Master Thesis

Graph-Based Deep Reinforcement Learning for Maintenance-Conditioned Wind Farm Wake Steering Control

ME54035: Master Thesis

Bas van Berkel

Delft University of Technology

TU Delft

Master Thesis

Graph-Based Deep Reinforcement Learning for Maintenance-Conditioned Wind Farm Wake Steering Control

by

Bas van Berkel

ME54035: Master Thesis

in partial fulfilment of the requirements for the degree of

Master of Science
in Mechanical Engineering

at the Department Maritime and Transport Technology,
Faculty Mechanical, Maritime and Materials Engineering,
Delft University of Technology

To be defended publicly on Monday, October 14th, 2024, at 10:00 AM

Student number:	4916778	
MSc Track:	Multi-Machine Engineering	
Report Number:	2024.MME.8990.pdf	
Project duration:	February 12, 2024 – August 16, 2024	
Supervision:	Prof. dr. X. Jiang,	TU Delft
	Prof. dr. E. Chatzi,	ETH Zürich
	Dr. P.G. Dominguez,	TU Delft
	Ir. G. Duthé,	ETH Zürich
	Ir. G. Arcieri	ETH Zürich

Cover: Condensation in the wake-affected Horns Rev wind farm. Photo by Christian Steiness; courtesy of Vattenfall under CC BY-NC 2.0 (Modified)

An electronic version of this thesis is available at http://repository.tudelft.nl/.

TU Delft

Abstract

Wind energy is growing to be an essential part of the transition towards sustainable energy sources. To facilitate this, it is crucial that wind farm operators can offer competitive energy prices compared to fossil fuel sources; effective and efficient wind farm operations are therefore imperative. However, many full-scale wind farms deal with wake effects, e.g. the disturbed air that travels downstream of a turbine and potentially ends up in other neighbouring turbines. The result of this lower-velocity, higher-turbulence flow field is both decreased power production and increased fatigue loads. One proposed solution for this problem comes in the form of 'wake steering': the yawing (rotating) of upstream turbines' nacelles to facilitate a degree of control over the deflection of wakes. By doing so, the problematic disturbed flow fields can be strategically guided between downstream turbines to maximise collective power production. However, the yawing action itself brings some adverse effects: the upstream turbines, now no longer directly facing the wind, feel decreased power production and increased fatigue loads themselves. These fatigue effects, accumulating over time, can eventually cause increased maintenance costs and nullification of any revenue gains through power optimisation. The problem thus becomes a farm-wide collective revenue optimisation task. This thesis investigates how long-term revenue in wind farms can be maximised, considering both profits through power optimisation and maintenance costs through fatigue-induced component failures due to wake steering control. First, a realistic wind farm simulation environment is constructed, based on a Graph Neural Network (GNN) surrogate wind farm simulation model, to facilitate efficient reinforcement learning training. Next, the environment is used to train fully centralised reinforcement learning agents based on a GNN architecture, resulting in agents that can generalise across all wind conditions and unseen wind farm layouts. Ultimately, the results show that an 'informed' agent that considers all profits and costs involved manages to significantly reduce the cost of energy compared to 'greedy' (power optimisation only), 'risk-averse' (damage minimisation only) and 'baseline' (zero-yaw) policies, furthermore considerably maximising long-term wind farm revenue by as much as 20%. Altogether, this thesis shows promising results in using graph-based reinforcement learning to train maintenance-conditioned, inflow-agnostic, and layout-agnostic wake steering controllers for wind farm revenue optimisation.

All code for this thesis can be found in the accompanying GitHub Repositories for the IEA37 3.4MW turbine surrogate[1], the GNN-based farm-level simulation surrogate[2], the simulation environment WakeWISE[3] and the wake steering controllers[4].

[1] https://github.com/bdvanberkel/IEA37_Surrogates
[2] https://github.com/gduthe/windfarm-gnn/tree/yaw-control-extension
[3] https://github.com/bdvanberkel/WakeWISE
[4] https://github.com/bdvanberkel/WakeWISEControllers

Preface

What a journey it has been! Spending six months working at the Structural Mechanics and Monitoring (SMM) group at ETH Zürich has been a true privilege. It has been incredible to work alongside so many intelligent and high-performing individuals. Their work - often truly excellent and state-of-the-art research - has been incredibly motivating, pushing me to get the most out of myself. I sincerely enjoyed the many interesting discussions during lunch or group meetings, sparking new ideas and inspirations that guided me towards my final results. Working on this thesis in Zürich has taught me a lot, from hard skills in distributed computing and wind farm modelling to working with Graph Neural Networks and Reinforcement Learning. I am always looking to learn new things, and working on this thesis has provided me with that and so much more. I could not have thought of a better way to conclude my academic career - the cherry on the cake if you will - than what the last seven months have been. I feel proud of what I delivered and I am happy with the final results, which will hopefully spark new ideas and concepts that others can build on. I look forward to seeing what the future brings, both to me as well as to wind farm research. There is so much left to discover, it is a matter of time until we see the rest of the endless possibilities and techniques. Looking back at my work - all of the bachelor, my year at the TU Delft Hydro Motion Team, the master and this thesis - I find that I always aimed to get the most out of myself. I therefore find the following Latin phrase very fitting to conclude this preface:

Nil satis nisi optimum
Nothing but the best is good enough

Bas van Berkel
Delft, September 2024

Acknowledgements

I would like to express my gratitude to many individuals who have been instrumental during the making of this thesis. They deserve to be acknowledged by name for their contributions and supervision they have so kindly provided me during the process. Without them, I would not have been able to get to where I am now, for which I am very grateful.

First of all, I would like to express my gratitude to Gregory Duthé and Giacomo Arcieri, who have provided me with excellent daily supervision during my stay at ETH Zürich. From the weekly update meetings, cracking our brains over the (in-)variance of reward functions to optimisation horizons, to troubleshooting the endless woes of RL training. My research was very much inspired by their incredible work on surrogate modelling and reinforcement learning. When talking about daily supervision, I must also mention Pablo Morato Dominguez, who was there for many of the weekly meetings to provide his thoughts and feedback and who participated in many interesting discussions. Similarly, I would like to thank Xiaoli Jiang for the supervision from Delft, and the excellent feedback she has given me throughout the thesis. Of course, my time at ETH Zürich would never have been possible without Prof. Eleni Chatzi who kindly hosted me at the Structural Mechanics and Monitoring (SMM) group. I am grateful for her hospitality in allowing me to work with her group, and for providing me with very excellent feedback during our update sessions. Then there are also the many incredibly intelligent and friendly people of the SMM group, which I would like to thank for the amazing Friday afternoon soccer games, the lunchtime get-togethers and their fascinating research which sparked many interesting discussions. I also want to mention Imad Abdallah who, together with Gregory, I got in touch with first for the proposed thesis. Finally, I would like to thank Kyriakos Chondrogiannis and Iasonas Soukas - my officemates at ETH, along with Giacomo - for the great banter during the many hours spent working alongside each other.

Of course, a special appreciation goes to my parents, without whom my six-month stay in Zürich would not have been possible. From the moment I mentioned a potential stay abroad to the conclusion of the thesis, they were nothing but supportive and enthusiastic. Furthermore, I would like to express my appreciation for the Justus and Louise van Effen Research Grant which I was kindly given by the TU Delft. Finally, my appreciation goes to the great maintenance team of the ETH Zürich Euler computing cluster for providing me with the computing power required for my research. I could not have done my research without it.

All of you - even those not explicitly mentioned by name - have contributed to this final product in some shape or form. I sincerely appreciate it!

Bas van Berkel
Delft, September 2024

Contents

List of Figures

List of Tables

Nomenclature

Abbreviations

Abbreviation	Definition
AEP	Annual Energy Production
ATC	Average Turbine Cost
ATP	Average Turbine Power
ATR	Average Turbine Reward
AYC	Active Yaw Control
BR	Baseline Removal
COE	Cost Of Energy
CTCE	Centralised Training with Centralised Execution
CTDE	Centralised Training with Decentralised Execution
DDPG	Deep Deterministic Policy Gradient
DEL	Damage Equivalent Load
DOF	Degree Of Freedom
DRL	Deep Reinforcement Learning
DTDE	Decentralised Training with Decentralised Execution
DTMC	Discrete-Time Markov Chain
DWM	Dynamic Wake Meandering
FEM	Finite Element Method
GAN	Generative Adversarial Network
GD	Gradient Descent
GEN	GENeralized graph convolution
GNN	Graph Neural Network
GS	Grid Search
GT	Game Theory
HAWC2	Horizontal Axis Wind turbine simulation Code 2nd generation
HRL	Hierarchical Reinforcement Learning
IEA	International Energy Agency
KPI	Key Performance Indicator
LCOE	Levelized Cost Of Energy
LDEL	Lifetime Damage Equivalent Load
LES	Large Eddy Simulation
LUT	Look Up Table
LSTM	Long Short-Term Memory
MAE	Mean Average Error
MAPE	Mean Average Percentage Error
MARL	Multi-Agent Reinforcement Learning
MDP	Markov Decision Process
MLP	Multi-Layer Perceptron
MSE	Mean Squared Error
MTBF	Mean Time Between Failures
NREL	National Renewable Energy Laboratory
NTM	Normal Turbulence Model
OpEx	Operational Expenditure
PAR	Probabilistic Auto-Regressive
POMDP	Partially Observable Markov Decision Process
PPO	Proximal Policy Optimisation

Abbreviation	Definition
PSD	Power Spectral Density
RANS	Reynolds-Averaged Navier-Stokes
RL	Reinforcement Learning
RMSE	Root Mean Squared Error
RMSPE	Root Mean Squared Percentage Error
ROSCO	Reference OpenSource Controller
WakeWISE	Wakesteering Windfarm Interactive Simulation Environment

1

Introduction

As the world is transitioning to the usage of sustainable energy sources in an attempt to tackle climate change, wind energy is growing to be an essential area of energy production. In a recent report by the International Energy Agency (IEA), they forecast that by 2028, as much as 12.1% of all energy will come from wind energy alone, making up more than a quarter of all sustainable energy produced (IEA, 2024). This indicates a growth of nearly 9% per year on average. Additionally, the IEA found that the majority of newly installed wind energy sources can provide competitive Levelized Cost Of Energy (LCOE) prices compared to fossil fuel sources, often even outperforming them entirely. It is thus evident that wind energy will play a vital role in the energy transition and that effective and cost-efficient operations are essential to replace the current fossil fuel energy system.

Wind farms often deal with the effects of wakes, e.g. the disturbed air that flows downstream of a wind turbine. In the process of generating energy out of the atmospheric boundary layer, wind turbines leave behind more turbulent and lower-velocity wind, which consequently propagates downstream and possibly ends up in neighbouring turbines. This tends to be a significant problem, as these velocity deficits cause decreased performance of downstream turbines and, thereby, reduced power production for the wind farm as a whole (Wu and Porté-Agel, 2015; Barthelmie et al., 2007). This problem often worsens as wind farm size increases due to an increasing number of turbines being aligned relative to the wind direction. Despite wind farm operators' best efforts to ensure the most optimal wake-effect-mitigating wind farm layouts under dominant wind directions, many wind conditions will still yield wake problems and, consequently, sub-optimal wind farm performance. Wake effects in wind farms can cause as much as a 40% efficiency loss in cases where turbines are aligned relative to the wind (Barthelmie et al., 2009). Additionally, the increased flow field turbulence in wakes can cause significant increases in aeroelastic loads in the downstream turbines (Lee et al., 2018). Naturally, this called for the development of methods to mitigate these effects, barring adjustments to the wind farm layout, which is assumed to be fixed.

In recent years, one proposed solution to this problem came in the form of wake steering. Researchers realised that the turbines' yawing capability, e.g. rotating the nacelle relative to the tower, could be used to adjust the propagation direction of the produced wakes. Under default circumstances, where turbines are yawed to face the incoming wind directly, the wakes propagate parallel to the wind direction. However, assigning a yaw offset relative to the wind to upstream turbines facilitated a degree of control over the direction in which the wakes propagated through the farm. Using this wake steering technique, problematic wakes can be steered away from downstream turbines, thereby mitigating the velocity deficit and increased turbulence effects, which would have otherwise resulted in significant adverse effects downstream. Despite the yawing action causing sub-optimal power production for the upstream turbines, the improved performance of downstream turbines collectively ensured a net benefit for the wind farm. Wake steering techniques facilitated several per cent performance increases compared to the naive wake-affected 'do-nothing' zero-yaw policy typically used in wind farms (Kadoche et al., 2023). Figure 1.1 and Figure 1.2 show the flow field and Annual Energy Production (AEP) effects of the do-nothing and optimised strategy, respectively.

However, wind energy engineers quickly pointed out that the yaw misalignment in wake steering led to undesired secondary effects. Generally designed for zero-yaw operating conditions, turbines can

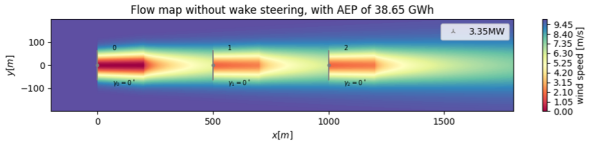

Figure 1.1: Flow map of a three-turbine wind farm without wake steering.

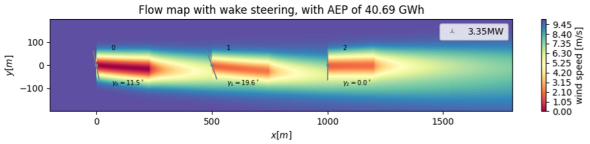

Figure 1.2: Flow map of a three-turbine wind farm with wake steering.

experience increased loads due to the yawing action (Damiani et al., 2018). What initially seems like beneficial performance through power optimisation with wake steering turns out to be more nuanced. The resulting increased loads, cyclic in nature, lead to accelerated fatigue degradation of wind turbine components. Fatigue failure, typically the dominating factor in turbine lifetime design, comes with significant maintenance costs, often requiring the complete replacement of components. What is more, the acceleration of fatigue accumulation only becomes visible in the long term when components start to fail, which can be as much as 20 years after the commissioning of the wind farm. The positive consequences of wake steering are instantaneous, whereas the adverse effects can linger in the background unnoticed for many years, after which they can cause significant monetary losses.

The problem of revenue optimisation thus becomes twofold. On the one hand, instantaneous power optimisation through wake steering allows immediate increases in energy production and, thereby, profit through energy sales. On the other hand, long-term fatigue effects coupled with wake steering for power optimisation cannot be ignored as they can potentially eventually nullify any revenue gains obtained. To keep the cost of energy low and to optimise the yield of sustainable energy sources, the typical wake steering controllers must be adjusted to account for all effects involved. This thesis will explore and further investigate this to find a suitable and effective method of constructing a maintenance-conditioned wake steering controller that accounts for both power optimisation and maintenance costs. Such a controller might then be used by wind farm operators to maximise the revenue they extract from a given farm, which ultimately translates to the electricity price presented to the customer. By ensuring a more cost-effective operation of the farm, wind energy will continue to provide competitive energy prices compared to fossil fuel sources and thereby facilitate a faster transition towards sustainable energy.

1.1. Problem Statement

Wake steering offers a feasible solution to tackle performance deterioration in wind farms due to wake effects. By introducing a yaw misalignment to various upstream turbines, the problem of downstream-propagating velocity deficits and turbulence increases ending up in downstream turbines can be partially mitigated. This facilitates nearly instantaneous increases in power production and, thereby, in short-term profit. However, as a result of wake steering actions, the accumulation of fatigue damage in turbines can be accelerated, causing long-term effects unaccounted for in power-only optimisation. If these effects are not considered, the instantaneous revenue gains can eventually be nullified by the introduction of more component replacements due to fatigue failure. The problem at hand is thus to develop an automated controller that determines the optimal set of yaw angles for the turbines in a wind farm, considering both the tasks of immediate power optimisation as well as the effects of long-term fatigue degradation, to optimise long-term revenue effectively.

1.2. Approach

The goal of this thesis is to design a wake steering controller that can determine the optimal set of yaw angles given the instantaneous environmental conditions. This work will cover the usage of Reinforcement Learning to obtain the controller with the desired behaviour. Through the usage of RL, a controller can be trained automatically without providing it with any ground truth knowledge. This method has several benefits. One is its ability to obtain a neural network controller that during inference takes mere milliseconds to run, which is ideal in a wind farm environment under constantly changing environmental conditions. Additionally, it can enjoy certain adaptability properties facilitating generalizability under changing environments, as will become evident later in this thesis. Furthermore, it paves the way for other variants of optimisation under long-term effects, a problem which is only effectively solved through RL optimisation techniques. To design the perceived wake steering controller, a two-step process is

used, as outlined below. For the perceived application of wind farm control, *training* of the controller happens a priori, ensuring the majority of computation time can be done outside real-time operation. Then, *inference* of the controller happens in real-time, taking only milliseconds to obtain the ideal set of yaw angles.

1. **Step one** is constructing a realistic simulation environment for the reinforcement learning agent to train in. Since reinforcement learning typically requires many millions of timesteps to be evaluated, a real-life wind farm is infeasible for it to learn efficiently. Instead, a virtual environment that mimics a realistic wind farm must be created. This environment must include and model wake effects, environmental conditions, component degradation, profits and costs.
2. **Step two** is to train the perceived reinforcement learning agent in the constructed environment. This requires careful construction and tuning of the agent, its architecture and the training algorithm to ensure it learns optimal maintenance-informed behaviour properly.

chapter 3, chapter 4, and chapter 5 will cover the first step of this process by covering the simulation model, fatigue theory and the construction of the environment, respectively. Next, chapter 6 and chapter 7 cover the training and evaluation of the trained agents in the environment, respectively. With this two-step process then concluded, chapter 8 and chapter 9 will discuss and conclude the work in this thesis. Central to the first part of the thesis is thus the creation of the environment. This requires several modelling choices for each of its modules. To ensure that these design choices align with the goals of the environment, a set of design criteria is formed by which the available options shall be judged. These will form the foundational principles upon which the environment is constructed and are listed below in no particular order.

- **Speed** - Sample efficiency is crucial as the environment will be used to train reinforcement learning models. As such, any choices concerning modules to be included in the environment should explicitly consider the required runtime. To ensure training feasibility, the step duration of the environment shall not exceed the order of milliseconds.
- **Accuracy** - A common problem in designing software in virtual simulation environments for later use in practice is the simulation-to-reality gap. To deal with this as best as possible, any choices concerning modules to be included in the environment should explicitly consider their accuracy. To ensure an accurate environment that models reality as closely as possible, accuracies should ideally be high in the ninety per cent.
- **Realistic** - To ensure any produced results (i.e. trained controllers) are judged fairly and realistically, the environment itself should be as realistic as possible. Aside from numerical accuracy, environmental behaviour should follow realistic trends. Consequently, any choices made concerning modules to be included in the environment should produce realistic inputs and follow real-life behaviour as closely as possible.

1.3. Research Questions

In this thesis, the following main research question will be investigated:

> **How can long-term profit be optimised in a wind farm by using Deep Reinforcement Learning for Wake Steering?**

To answer this main research question, it is broken up into several smaller sub-questions, each concerning a unique part of the thesis. Each of these makes up one chapter in this report. This means that it is structured as follows:

- chapter 3 on *How can the impact of wake steering on power production and fatigue loads be efficiently modelled in the context of a multi-turbine wind farm?* - Choosing and developing a suitable model that can determine both farm-level and turbine-level effects, for usage in a simulation environment.
- chapter 4 on *How can fatigue loads experienced by turbine components be mapped to accumulated lifetime consumption?* - Choosing and developing a method by which instantaneous fatigue loads can be converted to a contribution to long-term degradation effects.

- chapter 5 on *How can the environment be constructed in which the reinforcement learning agent is trained, to accurately represent a realistic wind farm?* - Development of the simulation environment to accurately and realistically represent a real-life wind farm, allowing the reinforcement learning algorithm to learn quickly and effectively.
- chapter 6 on *How can the architecture of the reinforcement learning agent be designed for effective wind farm control, and what methodologies should be used in training to ensure efficient learning?* - Setup and tuning of the reinforcement learning architecture and algorithm and the development of methodologies to facilitate training.
- chapter 7 on *To what extent does the performance of the trained reinforcement learning agent surpass or differ from that of various baseline controllers, and what insights can be gained from these comparative analyses?* - Performance comparison and evaluation of the trained controller(s) with baseline behaviour.

1.4. Contribution

This report is about several main contributions that made up the thesis project, each contributing to the final results. These contributions are as follows:

- **IEA Turbine-Level Load surrogates** - Development of power, thrust coefficient and load surrogates for the NREL-developed 3.4MW reference wind turbine, considering operating regions, operating modes, wind conditions and yaw angles.
- **Graph Neural Network Farm-Level Surrogates** - Development of an all-in-one combined layout-agnostic wind farm simulation surrogate, considering operating modes, wake effects and local power & loads, extending the work by Duthé et al. (2023).
- **WakeWISE Environment** - Development of a fast, efficient and realistic simulation environment for training wind farm wake steering controllers using reinforcement learning.
- **Wind Farm Wake Steering Controllers** - Development of generalisable graph-based wind farm wake steering controllers to optimise power, maintenance or a combination of both.

2 Background

In this initial chapter, first, some essential background knowledge is provided which will be at the centre of many chapters in this thesis. Section 2.1 covers key information about Damage Equivalent Loads (DELs) which form the basis of load and fatigue modelling within the simulation environment of this work. Next, section 2.2 covers Graph Neural Networks (GNNs), key to both surrogate modelling of wind farm mechanics and construction of the reinforcement learning agents in the latter half of this thesis. Finally, section 2.3 covers (Multi-Agent) Reinforcement Learning, theory which is essential for the latter half of the thesis regarding the training of wake steering controllers. Additionally, this chapter covers a review of wake steering literature and subsequently concludes with a summary of research gaps this work will cover.

2.1. Damage Equivalent Load (DEL)

The Damage Equivalent Load (DEL) is a measure often used in the context of wind turbines to quantify cyclic (fatigue) loads. Wind turbines deal with varying load cycles during operation, contributing to an accumulation of fatigue damage over time. In general, accumulated fatigue damage is assessed by investigating the load history of an area under interest and carefully bookkeeping the various number of cycles and respective load magnitudes. A typical method of consistently differentiating between different load cycles is, for example, the Rainflow Counting method T et al. (1974). These cycles can then be fed through an (often) empirically derived equation to determine the added damage each cycle has on the total accumulated lifetime fatigue. Though this serves well to perform a detailed load analysis, it is often hard to distinguish between the consequences of two load histories without tediously performing the aforementioned cycle counting and fatigue damage conversion. This gave rise to the defining of the DEL values, effectively summarising entire load histories into a single, more representable number.

The DEL effectively provides a hypothetical load magnitude that, when applied at a specific reference frequency over the same time period, results in an equal accumulation of fatigue damage as the original cyclic load time series it was based on. In other words, it represents the often complex and varying load spectrum as an equivalent single frequency single magnitude load. Doing so facilitates a direct comparison between two-time series loads without tediously investigating the load cycles involved and helps quantify fatigue effects over longer time intervals. In the wind turbine industry, the DEL is often calculated over a 10-minute load time series to effectively quantify the fatigue effects under certain wind conditions in a single representable number. A higher DEL means a higher degree of damage accumulation and, thus, very fast turbine degradation under the given circumstances. The DEL is thus often used during the design phase of a wind turbine with regard to fatigue analysis.

The theory behind the calculation of the DEL is based on Palmgren-Miner's linear fatigue damage accumulation rule (Miner, 1945) and the fatigue curve relations. The fatigue curves relate a given magnitude of a cyclic load to the number of cycles which would lead to failure. Typically, these curves are expressed in terms of stress, but in the field of wind energy, they are generally expressed in terms of loads or moments. Logically, there exists some load magnitude that would lead to direct failure at the first load cycle. This is the ultimate strength of the component or material in question. On the other

end, however, there exists some cyclic load magnitude that can essentially exert an endless number of load cycles without ever causing the component or material to break due to fatigue failure. This is typically called the fatigue limit or endurance limit. Anyhow, using this relation between cycles-to-failure, a given cyclic load magnitude and the number of cycles endured under that load, the increase in fatigue damage ΔD can be determined using Palmgren-Miner's linear damage accumulation rule. It states that the fraction of the number of cycles till failure endured is the added damage ΔD and that the fatigue damage effects of multiple cyclic loads can be linearly added together. Given the fatigue curve relation $N(DEL_i)$ mapping load magnitudes DEL_i to the number of cycles till failure and the number of cycles endured at each load n_i, the added fatigue damage is:

$$\Delta D = \sum_i \frac{n_i}{N(DEL_i)} \tag{2.1}$$

Furthermore, going by the definition of $N(DEL_i)$ derived from work by Freebury and Musial (2000) and NREL, it can be determined using the Wöhler exponent m characterising the gradient of the fatigue curve and the ultimate material strength DEL_u, as follows:

$$DEL_i = DEL_u \cdot N(DEL_i)^{-\frac{1}{m}} \tag{2.2}$$

$$\frac{DEL_i}{DEL_u} = N(DEL_i)^{-\frac{1}{m}} \tag{2.3}$$

$$\left(\frac{DEL_i}{DEL_u}\right)^{-m} = N(DEL_i) \tag{2.4}$$

$$\left(\frac{DEL_u}{DEL_i}\right)^{m} = N(DEL_i) \tag{2.5}$$

This can be merged with Palmgren-Miner's linear damage accumulation rule from earlier:

$$\Delta D = \sum_i \frac{n_i}{\left(\frac{DEL_u}{DEL_i}\right)^m} \tag{2.6}$$

$$\Delta D = \sum_i n_i \cdot \left(\frac{DEL_i}{DEL_u}\right)^m \tag{2.7}$$

This damage D, accumulated for each component individually based on their load history, measures how much of the fatigue lifetime has been consumed so far. Per Miner's rule, the component will experience fatigue failure when this damage state D reaches or exceeds a value of 1, indicating a complete consumption of available fatigue lifetime. What remains is to define how a time series cyclic load signal is converted and summarised to a single DEL value to be used in the above computations. Realising that any isolated constant amplitude cyclic load is effectively a DEL for the same number of cycles and that the DEL should result in an equivalent amount of damage:

$$f_{eq} T_s \left(\frac{DEL}{DEL_u}\right)^m = \sum_i n_i \left(\frac{L_i}{DEL_u}\right)^m \tag{2.8}$$

$$f_{eq} T_s DEL^m = \sum_i n_i L_i^m \tag{2.9}$$

$$DEL^m = \frac{\sum_i n_i L_i^m}{f_{eq} T_s} \tag{2.10}$$

$$DEL = \sqrt[m]{\frac{\sum_i n_i L_i^m}{f_{eq} T_s}} \tag{2.11}$$

Where f_{eq} is the reference frequency of the DEL signal to be calculated (often set to 1Hz), T_s is the length (in seconds) of the signal that is being converted, n_i & L_i the cycle count and amplitude of each

cycle in the original signal and m & DEL_u the Wöhler exponent and ultimate load of the component subject to the loads. Given this theory, any time series cyclic load signal can be converted to a DEL to summarise the effects of fatigue effectively.

2.2. Graph Neural Networks

Graph Neural Networks (GNNs) (Scarselli et al., 2009) extend the properties of neural-network-based machine learning to graphs. Graphs are defined using a set of nodes $\mathcal{V}$ and a set of edges $\mathcal{E}$ connecting them, using the $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ notation. Graphs typically have an adjacency matrix $\mathcal{A}$ of shape $|\mathcal{V}| \times |\mathcal{V}|$ indicating with a value of 1 at index (i, j) if node i and j are connected. Consequently, edges can be unidirectional or bidirectional; in the latter case, $\mathcal{A}$ is symmetrical as edges exist in both directions. Additionally, each node in $\mathcal{V}$ and edge in $\mathcal{E}$ can have a feature vector $x_i \in X$ and $e_{ij} \in E$ respectively. Finally, a graph might have a global feature vector U that holds global graph features. Altogether, a graph with $k = |\mathcal{V}|$ nodes, $l = |\mathcal{E}|$ edges, n node features, m edge features and o global features can be represented as $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{A}, X, E, U)$ where $\mathcal{A} \in \mathbb{R}^{k \times k}$, $X \in \mathbb{R}^{k \times n}$, $E \in \mathbb{R}^{l \times m}$ and $U \in \mathbb{R}^{o \times 1}$.

GNNs implement the feed-forward step of traditional neural networks on graphs through the message-passing paradigm. Rather than feeding values forward through succeeding network layers, it is propagated in place over the graph between connecting nodes. The input embedding of nodes and edges is thereby transformed into a different output embedding on a similar, if not identical, graph structure. Inherently, this means that graph topology plays a vital role in how and where information is shared over the graph. This dependency of information sharing on topology facilitates important prior information and dependencies to be implicitly encoded in the network architecture. In recent years, this powerful method of dealing with problems containing a high degree of variable or topology interdependency has seen numerous applications in novel research. Examples come in the form of learning physical processes on graphs (Duthé et al., 2023), performing anomaly detection in multivariate signals (Qian et al., 2023) and predicting relationships between social networks as part of recommender systems (Fan et al., 2019).

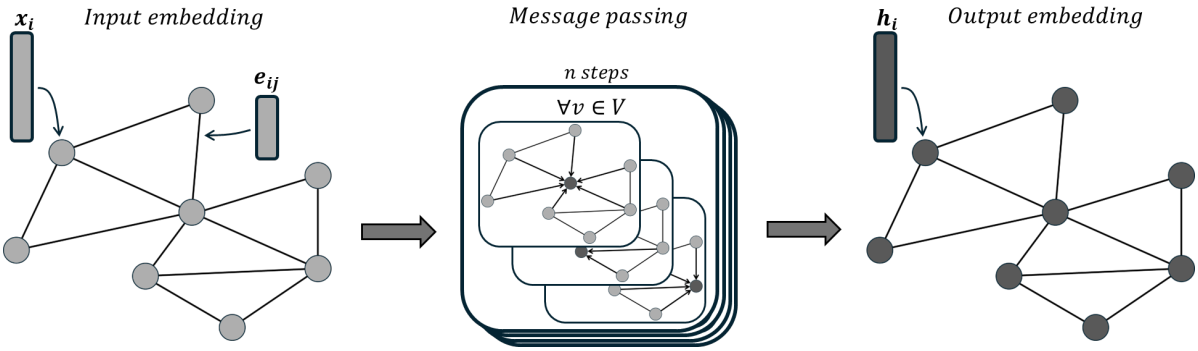

Figure 2.1: Example of the GNN process; the input embedding is fed through n in-place message passing blocks to obtain the output embedding.

Figure 2.1 shows the typical process of a GNN. An input embedding in the form of a graph with nodes, edges, node features and edge features is fed through one or multiple message-passing blocks. These blocks propagate information (features, values) between nodes, thereby updating all nodes on the graph. The output embedding of message-passing layer κ produces a graph embedding with hidden states h_i^κ of the nodes, which is consequently used as the input for layer $\kappa + 1$. The messages (values) being passed between nodes can be a function of the node features, edge features, global features or a combination thereof. This differs between the various message-passing techniques that can be found in literature. The aggregation of messages during the propagation step can be done using various methods, such as taking the mean or sum. A review of GNN architectures and techniques can be found in Zhou et al. (2021).

2.3. (Multi-Agent) Reinforcement Learning

Reinforcement Learning (RL) is the machine learning paradigm concerned with finding a decision-making policy for an agent to take actions in an environment and maximising the cumulative reward. It

does this by learning a decision-making policy $\pi(s_t)$ to infer an action $a_t \in \mathcal{A}$ at each timestep t based on the environment's state $s_t \in \mathcal{S}$. The environment then transitions to a new state s_{t+1} depending on the state transition function $P(s_t, a_t)$. Each timestep, the agent observes a reward r_{t+1} based on its behaviour and the state s_t. Favourable decisions are rewarded with higher rewards; bad decisions are penalised with lower rewards. The agent's training is done by letting it interact with the environment and discover which actions under which circumstances result in favourable returns. More formally, a reinforcement learning problem has a set of states $\mathcal{S}$, a set of actions $\mathcal{A}$ and a reward function $R(s_t, a_t)$. The goal is to maximise the cumulative reward R_t from timestep t and onwards:

$$R_t = \sum_{\tau=1}^{T} r_{t+\tau} \tag{2.12}$$

In finite-horizon problems, this sum of rewards will be bounded. However, in infinite-horizon problems, this is not necessarily the case. To deal with these cases, a discount factor γ can be added that discounts future rewards, or more simply said, ensures short-term rewards hold more value than long-term rewards. Adding this ensures convergence of the sum of values:

$$R_t = \sum_{\tau=t}^{T} \gamma^{\tau} r_{t+\tau} \tag{2.13}$$

Multi-Agent Reinforcement Learning (MARL) extends the paradigm of reinforcement learning to deal with environments that include multiple agents. Broadly speaking, there are three types of MARL: Centralized Training with Centralized Execution (CTCE), Centralized Training with Decentralized Execution (CTDE) and Decentralized Training with Decentralized Execution (DTDE). Furthermore, MARL problems can be cooperative or competitive in nature, depending on whether the agents share a common goal or not. The main differences with the single-agent case are that the state transition function P is now dependent on the joint action of all agents, there can be an action space $\mathcal{A}$ for all agents individually, and the reward could provide individual rewards for each agent depending on the formulation of the problem. In CTCE, one policy is trained to control all agents jointly, e.g., there is a single policy mapping (a collection of) observations to a joint action space. In CTDE, each agent holds their own policy, but training is done in a centralized way, allowing information to be shared between agents. This sharing of information is then omitted during testing time. In DTDE, each agent is trained and tested fully individually, meaning each agent learns completely independently.

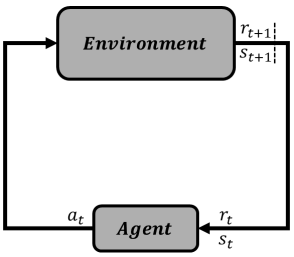

Figure 2.2: Single-agent training: a single agent interacts with the environment through one policy.

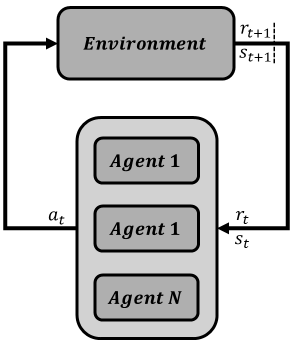

Figure 2.3: CTCE training: all agents are included in one policy.

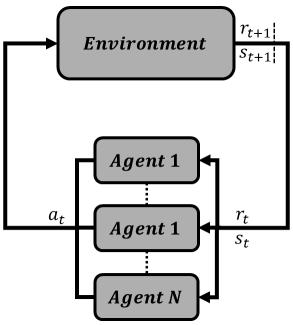

Figure 2.4: CTDE training: each agent has their own policy; information is shared.

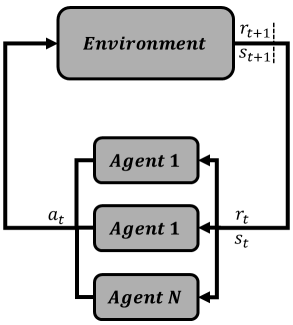

Figure 2.5: DTDE training: each agent has their own policy; no information is shared.

2.4. Literature Review: Wake Steering

At its core, the concept of leveraging wake steering for power optimisation in wind farms has been covered extensively in the literature. Over the last few years, various algorithms, methods and modelling principles have been applied for this purpose. Given the increased demand for sustainable energy

and the quest for efficiency as a result thereof, this field of research has become increasingly popular. Early work in wake steering can be traced back to 2005, in research by Medici (2005) investigating the effects of Active Yaw Control (AYC). More recently, Howland et al. (2019) conducted experiments on a six-turbine in-line wind farm, using analytic gradient ascent methods to optimise their yaw strategy. Using a custom-developed and calibrated analytical model for farm-level interactions, they employed the gradient descent search to find optimal set points. For selected wind directions, they showed that performance increases in power yield of the range $7 - 12\%$ are feasible. These results align well with earlier work highlighting experimental results from field tests using controllers trained in simulations (Fleming et al., 2017, 2019), showing performance gains of several per cent. In both cases, they developed wake-steering optimised yaw angle lookup tables using gradient descent and FLORIS (National Renewable Energy Laboratory (NREL), a) as a simulation model. Zong and Porté-Agel (2021) aimed to reproduce these aforementioned results in a controlled wind tunnel environment, measuring streamwise velocity deficits and turbine power outputs. Their results show agreement with prior research and experiments. Furthermore, they continued by calibrating analytical wake models to match experimental results, facilitating future work on active yaw control optimisation. Other examples of wake steering optimisation include the usage of Game Theory (Gebraad et al., 2016) and SNOPT, an optimiser for highly nonlinear, constrained problems (Gebraad et al., 2017). However, an inherent problem with simulation-based optimisation approaches that need transferring to real-life applications is the agreement between reality and simulation, or sometimes rather the lack thereof. Gori et al. (2023) investigate this sensitivity of simulation model choices on resulting optimal yaw strategies, highlighting a high degree of variability in output strategies. Furthermore, they find additional sensitivity concerning the employed optimisation algorithm. It is thus evident that care must be taken when using simulation-based optimisation strategies and picking optimisation algorithms.

With the increased popularity of reinforcement learning being applied to optimisation in the decision-making domain, wake steering has also seen ample RL-based research in recent years. Stanfel et al. (2020) employed decentralised multi-agent Q-learning for power optimisation in a three-turbine wind farm under a fixed wind condition, using FLORIS (National Renewable Energy Laboratory (NREL), a) as a steady-state flow simulation environment. They argue that wake steering based on Look Up Tables (LUTs) suffers from irregularities that are hard to model and promote the idea of using model-free control methods, such as Reinforcement Learning. In their environment, they explicitly account for time delays in wake propagation by 'locking' turbine yaw angles in place until wakes have propagated. They argue that this essentially solves the credit assignment problem in multi-turbine systems. Under the single wind condition they considered, they achieved a clear increase in wind farm power production. Similarly, Bui et al. (2020) grouped a 15-turbine wind farm in clusters of size three for information sharing and solved the optimisation problem using decentralised Double-Deep Q Learning. Rather than using a pre-made wake simulation environment, they calculated wake effects and power production using analytical and empirical relations. Using their MA-DRL approach under specific wind conditions, they achieved a power increase ranging from 1.99% to 4.11% depending on the layout of the three-turbine cluster. Dong et al. (2021), on the other hand, chose to make use of high-fidelity Large Eddy Simulations (LES) using SOWFA (National Renewable Energy Laboratory (NREL), b) to optimise power production in a six-turbine wind farm. Using Deep Deterministic Policy Gradient (DDPG), they found a 15% increase in power compared to a zero-yaw baseline under a fixed flow field. The works mentioned above typically only train their algorithms with small-sized wind farms, which are far from realistic; in real life, most wind farms will have many more turbines. Kadoche et al. (2023) and Padullaparthi et al. (2022), on the other hand, extend this to work with as many as 151 turbines. Both works use multi-agent RL, using either PyWake or FLORIS as wind farm simulators, and find power increases as high as several per cent depending on the farm size. In contrast to other work, Kadoche et al. (2023) train their algorithm on an environment with (limited) dynamic wind conditions rather than the fixed conditions the majority of other papers in this section made use of. Additionally, Padullaparthi et al. (2022) extend their simulation to include a measure of fatigue to effectively find a trade-off between power optimisation and component longevity. They include this in the reward function employing an arbitrarily chosen weighted sum between power production and fatigue loads.

The trade-off between power optimisation and fatigue loads can also be seen in several other papers from recent years. This is an important factor to consider, as yawing action can significantly impact fatigue loads on components (Damiani et al., 2018). Interestingly, yaw control itself can be used to minimise fatigue loads on the turbine (Kragh and Hansen, 2014; Fleming et al., 2015; Shen et al.,

2011). However, Ke et al. (2018) and Jeong et al. (2013) also show that applying yaw offsets to wind turbines can lead to detrimental effects, such as significantly decreasing fatigue lifetime. These effects can thus not simply be neglected in favour of immediate power optimisation. Lin and Porté-Agel (2020) investigated the trade-off between fatigue loads and power optimisation through wake steering and found that in some cases, the benefits of power production increase are neglected by the increased loads' negative effects. In the same way that Padullaparthi et al. (2022) modelled the trade-off between loads and power in their reward function, work by Ennis et al. (2018), Bossanyi (2018) and van Dijk et al. (2017) attempted to solve the problem as a multi-objective optimisation problem by giving relative importance to power optimisation and load minimisation. In many of these papers, only a single DEL (fatigue load) is considered. Ennis et al. (2018), for example, solely consider flapwise loads, whereas Bossanyi (2018) only incorporate tower base loads in their objective function. To address this issue, He et al. (2023) consider a total of five different fatigue loads to characterise the wake-fatigue effects as well as possible. However, all of these works addressing the trade-off between power and load treat the problem as a multi-objective optimisation problem, neglecting the true monetary losses resulting from maintenance due to fatigue degradation.

	Environment		Degradation		Solution	
Paper	Farm Size	Random Wind	Included	Components	Objective	Method
Howland et al. (2019)	6	≈	×	-	Power	GD
Fleming et al. (2017)	6	√	×	-	Power	GD
Fleming et al. (2019)	5	√	×	-	Power	GD
Zong and Porté-Agel (2021)	3-80	√	×	-	Power	GD
Gebraad et al. (2016)	6	×	×	-	Power	GT
Gebraad et al. (2017)	60	√	×	-	Power	GD
Stanfel et al. (2020)	3	×	×	-	Power	RL
Bui et al. (2020)	15	×	×	-	Power	MARL
Dong et al. (2021)	6	×	×	-	Power	RL
Kadoche et al. (2023)	4-151	≈	×	-	Power	MARL
Padullaparthi et al. (2022)	21	×	√	1	Power+Damage	MARL
Lin and Porté-Agel (2020)	3	×	√	1	Power+Damage	GS
Bossanyi (2018)	6	≈	√	2	Power+Damage	GS
van Dijk et al. (2017)	9	√	√	2	Power+Damage	GD
He et al. (2023)	2	×	√	5	Power+Damage	GD

GD: Gradient Descent, **GT**: Game Theory, **(MA)RL**: (Multi-Agent) Reinforcement Learning, **GS**: Grid Search

Table 2.1: Overview of reviewed literature

2.5. Research Gap

Following the literature review, some research gaps can be identified that need to be addressed in future research. These gaps allow innovation to proceed and answer questions that were left from prior research. The following gaps can be identified:

- **Maintenance & Long-term effects** - A significant gap that can be identified in the current research landscape is the evident gap in explicitly addressing monetary losses due to maintenance effects. Applying yaw angles to wind turbines causes (often significant) changes in load conditions and, therefore, in fatigue-induced component failure rates. These effects are either not accounted for or only included through some multi-objective optimisation goal with arbitrary coefficients of importance. Considering these effects when employing wake steering strategies is essential to avoid unexpected long-term revenue decreases whilst power production is seemingly optimised in the short term. Padullaparthi et al. (2022) attempt to address this but rely on arbitrarily chosen weights to quantify the 'cost' of fatigue loads. Instead, long-term effects must be expressed in a way that they can be jointly optimised with energy yield, making no arbitrary assumptions on relative importance.
- **Large layouts** - Very few papers have addressed the problem of wake steering optimisation in large-scale wind farms, i.e. with more than a handful of turbines. Clearly, the majority of works discussed in section 2.4 considered scenarios with only a few turbines, often in straightforward layouts such as single rows. When it comes to papers that apply reinforcement learning to the

problem, larger layouts are seen more frequently. Still, there is plenty of room for research towards wake steering in wind farms with a more realistic number of turbines.

- **Generalisable Controllers** - Many discussed works in section 2.4 consider only a single wind condition and a fixed layout in their research. This effectively means that the controller is trained on very specific and fixed wind conditions, lacking any generalizability towards arbitrary inflow conditions. In other words, it likely strongly overfits the conditions it is provided with. When it comes to applying trained wake steering controllers in real life, it is of significant importance to have inflow-agnostic controllers, e.g. controllers that are not constrained to particular conditions. In more recent work by Kadoche et al. (2023) and Padullaparthi et al. (2022), this is already partially addressed by training on a small but more diverse set of wind conditions. Still, there is yet plenty of room for more research. Additionally, the trained controller only works on the specific layout it was trained on; being able to transfer such a controller would make it much more practical and applicable. The latter concerns the creation of layout-agnostic controllers and too constitutes an open area of research.

Given this information, there seems to be a big opportunity in research towards maintenance-informed wake steering controllers for long-term revenue optimisation in large wind farms, robust to varying inflow conditions and wind farm layouts. As such, this work will attempt to cover this research gap and explore its possibilities.

2.6. Chapter Recap

This chapter covered some essential background knowledge on the topics of DELs, GNNs and MARL. These topics will be used extensively throughout this thesis and thus were given a separate initial introduction. Next, a literature review on work and publications in the field of wind farm wake steering was done to investigate the current research landscape. From this, three main research gaps became evident: modelling of maintenance & long-term effects, dealing with large layouts and generalising controllers across farm topologies and wind conditions. With this knowledge of essential topics and research gaps, the next chapter can proceed with the initial steps towards constructing the simulation environment as required for step one of this thesis.

3

Modelling of Turbines and Farms

An essential property of the simulation environment is its ability to accurately represent the farm-wide wake interactions and their impacts on local loads and power production. Inherently this means that a model capable of such calculations should be at the heart of this environment. A suitable model must thus be chosen that explicitly considers these effects and calculates them for any arbitrary multi-turbine wind farm layout, given any arbitrary global wind inflow condition and any arbitrary set of yaw angles. In this chapter, several options for such a model are explored, compared and contrasted in section 3.1. Next, based on the fundamental design criteria, an informed choice is made and further developed for usage inside the simulation environment in sections 3.3 and 3.4.

3.1. Model Choice

There are a variety of models available for farm-level modelling of wake effects, as well as for modelling turbine-level loads and power as a function of local wind conditions. Fundamentally, the available options differ in their levels of approximation/fidelity and, therefore, in the accuracy of calculated results. Choosing which model to use generally boils down to comparing the specific characteristics of each model and determining which aligns best with the environment objectives. As such, in the following section, different categories of models are compared and contrasted by their characteristics to make an informed decision effectively. An important distinction is the difference between models capable of determining farm-level effects and those solely capable of modelling local (turbine-level) effects. As will be evident in the following subsection, these will have to be treated individually and later merged together. As such, both of these categories are covered separately.

3.1.1. Farm-Level

The primary goal of modelling farm-level effects is quantifying the interactions between turbines resulting from produced wakes. The accuracy of these calculations is directly related to the methods with which they are done, which typically scale with model fidelity. Several research institutes and individual developers have released (open-source) software packages for these farm-level simulations, as highlighted in Figure 3.1. As mentioned, as model fidelity (and thus accuracy) increases, the computational effort (and hence runtime) also increases. Additionally to the a priori available software packages, a fifth option *Custom Surrogate* is added to the list. This is a proxy for a surrogate model yet to be trained, consisting of some neural network setup trained on a dataset generated by one or more farm-level simulators.

The first category of comprehensive models encapsulates both FLORIS (National Renewable Energy Laboratory (NREL), a) and PyWake (Pedersen et al., 2023), two packages developed by different developing teams but almost identical in their methods. These low-fidelity models are based on empirical relations and fitted formulas, calculating wake effects through wake deficit and wake deflection models derived from literature. The power of these relatively simple models is precisely their simplicity, as they inherently run much faster than the higher-fidelity models, e.g. in the order of fractions of a second. However, their simplicity might also have implications for the accuracy of the results they

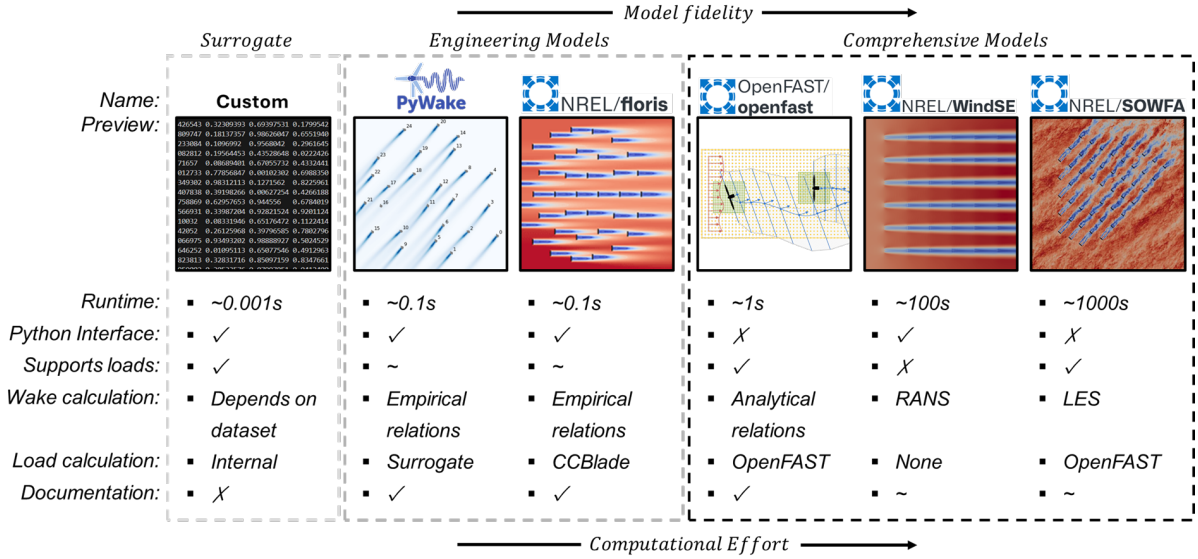

Figure 3.1: Comparison of several available farm-level simulation models.

produce. Nonetheless, as is evident from comparisons by Meyer Forsting et al. (2023); Fischereit et al. (2022); PyWake Development Team (2024), PyWake can yield satisfactory results for engineering applications. Similarly, FLORIS has shown good agreement with field results (Fleming et al., 2021) and wind tunnel tests (Campagnolo et al., 2022). Both models in their standard form, however, have limited native support for calculating turbine-level loads in their simulation and thus likely need additional local load computing modules.

The second category contains higher-fidelity models, considering additional mechanisms and mechanics compared to low-fidelity models like FLORIS and PyWake. The first of these, FAST.FARM (National Renewable Energy Laboratory (NREL), 2024b), uses Dynamic Wake Meandering (DWM) models to compute farm-level wake effects. Additionally, it includes an OpenFAST turbine-level aeroelastic simulation module to calculate local effects like power and loads. In essence, FAST.FARM has increased fidelity in flow physics, as well as the addition of a comprehensive aerodynamic and structural calculation module for each of the individual turbines involved. Aside from being able to compute influences on power production as a result of wakes, it facilitates comprehensive load analysis. Due to its trade-off between relatively simple flow physics and comprehensive turbine-level analysis, FAST.FARM can run in the order of seconds over multiple computing cores. Its farm-level DWM wake calculations have shown excellent agreement with Large Eddy Simulations (LES) as shown by Rivera-Arreba et al. (2023) and Jonkman et al. (2018), as well as good agreement in structural loads with real-life experiments (Kretschmer et al., 2021).

The last two models, NREL's WindSE (National Renewable Energy Laboratory (NREL), 2024a) and SOWFA (National Renewable Energy Laboratory (NREL), b), rely on (un-)steady Reynolds-Averaged Navier-Stokes (RANS) and Large Eddy Simulations (LES) simulations respectively. These two models determine farm-level wake effects through comprehensive high-fidelity aerodynamic calculations. It is important to note that WindSE does not have native support for turbine-level load calculations and thus needs additional modules; SOWFA, on the other hand, has a version of OpenFAST embedded for this purpose. WindSE can still be run with limited hardware, albeit with a significant runtime in the order of minutes. SOWFA, on the other hand, requires many more computing cores and can take up to several hours to simulate. Their results, however, align well with experimental real-life measurements (Churchfield et al., 2015; Shaler et al., 2023), though performance increase seems to be relatively limited compared to models like FAST.FARM or PyWake/FLORIS, especially for engineering applications.

Given these options, usage of the higher-fidelity category of comprehensive models already seems infeasible due to their relatively long runtime requirements and required computational effort, violating the environment's *speed* objective. Besides, it is unlikely such highly detailed levels of flow physics, such as RANS or LES, are required from an engineering perspective, as is the case in the environment. Additionally, the lack of load calculations in SOWFA requires augmenting it with additional mod-

els, further increasing its complexity. This leaves three options from a farm-level perspective: PyWake, FLORIS, or a self-trained surrogate model. Given the environment design objectives, specifically the *speed* requirement, both PyWake and FLORIS seem unsuitable, as they will undoubtedly function outside the millisecond order of magnitude. Fortunately, prior research has shown that PyWake's calculations can be successfully imitated by means of a Graph Neural Network surrogate, retaining accuracy whilst drastically reducing model runtime (Duthé et al., 2023). Its ability to infer results within milliseconds whilst retaining the satisfactory accuracy that PyWake provides makes a GNN-based custom-trained surrogate ideal for inferring farm-level results. The most suitable and feasible option is thus to develop a custom surrogate model that models the farm-level effects with limited runtime.

3.1.2. Turbine-Level

Similarly to farm-level models, the research community has provided several turbine-level models for computing local effects like power and loads. The available options are summarised in Figure 3.2. Interestingly, PyWake has initial experimental support for turbine-level power, thrust coefficient and load determination in the form of surrogate models. These surrogate models consist of Multi-Layer Perceptrons receiving local flow conditions as input and determining local variables for each turbine. PyWake has divided each of these surrogates into three separate networks, one for each operating region of below cut-in, regular operation and above cut-off. This was done as input-output relations differ significantly between regions, and training them separately yields a higher overall output accuracy. However, investigating the metadata of the model files reveals that these surrogates may perhaps not be very accurate. For one, most of the networks consist of a single hidden layer with only 15 nodes, which is seemingly insufficient to model complex relations. Furthermore, these surrogates were trained on as little as 2000 samples over all three operating regions combined; for comparison, Haghi and Crawford (2023) used a dataset consisting of 32768 samples, further using a three-hidden-layer MLP for training. Finally, none of PyWake's surrogates considers yaw misalignment, meaning the effects of yaw misalignment with the oncoming wind are not modelled. Since yaw misalignments are explicitly part of the trade-off in the environment, these surrogates are seemingly unsuitable without further adjustment.

Another two options are OpenFAST (National Renewable Energy Laboratory (NREL), 2024b) and HAWC2 (DTU Wind, Technical University of Denmark), developed by NREL and DTU respectively. Both are similar in their computations and capabilities and can model the turbine response to an incoming flow field. OpenFAST is open source and freely available on their GitHub repository, whereas HAWC2 is commercial software for which a license must be obtained. Both perform comprehensive aerodynamic and structural computations and do not rely on underlying surrogates or rough approximations to infer their results. This means they are significantly more accurate than the aforementioned experimental surrogates implemented in PyWake. However, their runtime is generally in the order of minutes in order to simulate a time-series response, as is required for DEL calculations. CCBlade (NREL, 2024b) is another comprehensive aerodynamic and structural computation model focused on blade calculations only. It was also developed by NREL, mainly for gradient-based airfoil optimisation. However, importantly, it cannot simulate blade responses under complete turbulence fields and does not support providing turbulence intensities as inputs, which is an essential part of wake calculations. Furthermore, its inability to simulate time-series turbulence fields makes it unsuitable for DEL calculations.

Additionally, there is the option of developing custom surrogates for this purpose. Similarly to the custom surrogate option in the farm-level case, this is a proxy for a surrogate model yet to be trained. Such a model would require a dataset generated by some more comprehensive model such that it can learn to imitate its calculations. Prior research has shown that MLP-based surrogates are excellent function approximators to imitate the calculations of comprehensive aerodynamic and structural solvers, as is evident in work by Haghi and Crawford (2023) and Dimitrov et al. (2018) using training sets generated by OpenFAST and HAWC2 respectively. These papers show that MLP-based surrogates can achieve excellent accuracies whilst operating with inference speeds no more than a few milliseconds.

Given these options, it becomes apparent that no readily available model satisfies the environment design objectives when it comes to inference speed. Both OpenFAST and HAWC2 significantly exceed the runtime design requirement by several orders of magnitude, and CCBlade is incapable of processing time series and unsuitable for tower load analysis. However, it is still possible to incorporate the power and load dynamics directly in the farm-level model; this would mean that, at runtime, only the

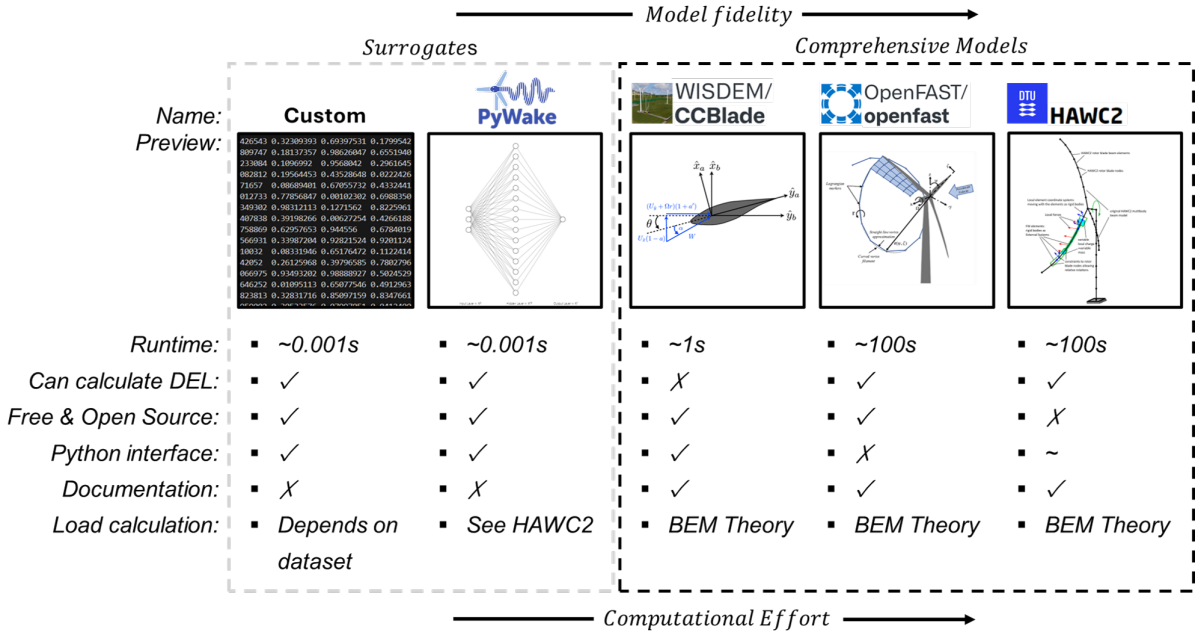

Figure 3.2: Comparison of several available turbine-level simulation models.

inference speed of the GNN surrogate would be relevant. During dataset generation, inference speed would be less critical, as that can all be done beforehand. For dataset generation of the farm-level surrogate, a turbine-level simulation must be done for each turbine in the farm layout. Given that several thousand setups must be run to generate the training dataset, as mentioned by Duthé et al. (2023), individually running comprehensive turbine-level simulations for each of the turbines in each sample farm becomes yet again infeasible. From a practical perspective, it is thus preferred to use a model with fast inference speeds, being the aforementioned custom turbine-level surrogates. This surrogate can be trained a priori on a set of arbitrary comprehensive simulation results and then jointly applied with PyWake to the dataset generation for the combined farm-level and turbine-level GNN surrogate. This means that turbine-level calculations are initially decoupled from farm-wide calculations for dataset generation. Afterwards, they are combined to form a dataset, from which the farm-level surrogate learns both simultaneously. Note that this is where the approach in this thesis deviates from Duthé et al. (2023), as they used the turbine-level surrogates present in PyWake itself. Instead, this thesis chooses to develop custom surrogates to deal with the issues outlined earlier this section.

3.2. Approach

From the previous sections, it has become clear that a custom GNN-based surrogate that simultaneously takes care of both farm-level and turbine-level calculations seems most suitable for the environment. To train this surrogate, first a comprehensive and elaborate dataset must be generated that includes ground truth calculations for various wind farm layouts, yaw angles and inflow conditions. For this, PyWake will be used to model farm-level effects combined with a custom turbine-level surrogate for turbine-level effects. This turbine-level surrogate, in turn, is to be trained on a dataset of its own. For this, OpenFAST will be used, as it is a free, open-source and relatively fast comprehensive wind turbine calculation tool capable of processing time-series data. First, the following assumption shall be made about the modelling of turbine-level variables:

Assumption 1 *Turbine power, thrust coefficient and loads can be modelled as a function of local wind conditions (disk-average wind speed, turbulence intensity, shear exponent) and yaw angle, i.e. as* $f(V_w, I, \alpha, \gamma)$

This assumption allows the turbine-level surrogates to accept single values for each input variable. By doing so, the effects of half-waked conditions are only partially accounted for, though it benefits from

significantly simplifying the creation of the surrogate model. The approach to creating the final wind farm surrogate model is thus as follows:

1. **Turbine-Level Dataset Generation** - Use OpenFAST to simulate a wind turbine under various wind conditions and yaw angles to create a dataset for training the turbine-level surrogate.
2. **Turbine-Level Surrogate Training** - Create a surrogate model framework and train it on the previously generated dataset, and evaluate its performance.
3. **Farm-Level Dataset Generation** - Use the previously trained turbine-level surrogate model in combination with PyWake to simulate various wind farm layouts under various inflow conditions and with various yaw angles to create a dataset for training the farm-level surrogate.
4. **Farm-Level Surrogate Training** - Create a surrogate model framework and train it on the previously generated dataset, and evaluate its performance.

Each of the surrogates, being farm-level and turbine-level, will be created according to the following steps:

1. Determine which variables are included as inputs, determine their interdependencies and define their lower- and upper bounds.
2. Generate samples that cover the input space sufficiently.
3. Determine the correct setup for the simulation software.
4. Run simulations for each of the input samples.
5. Post-process the results, if necessary.
6. Determine the architecture of the surrogate model.
7. Train the surrogate to predict the outputs given the input samples.
8. Evaluate the performance of the trained surrogate.

In the following sections, each of these steps is discussed and - where necessary - further subdivided into multiple subsections.

3.3. Turbine-Level Model

The turbine-level model's main purpose is to map inflow conditions and yaw angles to corresponding power, thrust coefficient, and load values at each turbine. However, some design requirements must first be formed to guide the surrogate creation process. As such, this initial section will discuss the requirements that are put on the model.

First, a specific turbine type must be chosen around which the surrogate model is built. Since the dataset generation will require simulations to be run in OpenFAST, sufficient and proper model files must be readily available to avoid unnecessary preparatory work. For this purpose, the reference wind turbines developed by IEA Wind Task 37 are very suitable as they provide ready-to-use OpenFAST model files combined with a turbine-level controller. They provide three wind turbine models: the 3.4MW, 10MW and 15MW versions. However, it has been recognised by the developers of OpenFAST that there exists an inexplicable edgewise instability in the 10 and 15MW turbines at certain high yaw angles (Jonkman, 2019), making simulating them with OpenFAST notoriously tricky. The 3.4MW turbine does not seem to suffer from this issue and is thus the more reliable turbine to deal with. The first requirement is, therefore, that the model shall be valid for this 3.4MW turbine. Next, since the environment shall be as realistic as possible, a wide range of wind conditions might occur. It is thus essential that the model shall function outside rated operating conditions as well; this forms the second requirement. Similarly, given that maintenance might occur every now and then, the model must be able to simulate a 'parked' turbine, e.g., a turbine that is out of operation and thus has a standstill rotor.

Regarding input variables, the allowable yaw angles must cover a sufficient range to allow for enough wake steering freedom. Since this is impossible to quantify a priori, a range of $[-30, +30]$ is chosen as a requirement. Additionally, this requirement agrees with the findings of Zong and Porté-Agel (2021), highlighting the maximum optimal yaw angles they computed in their setup. For the inflow variables, however, the IEC standard on design requirements for wind energy systems can be followed. This standard prescribes operating conditions under which turbines must be designed; as part of this, it defines the inflow conditions and relations among them. As such, following these inflow conditions

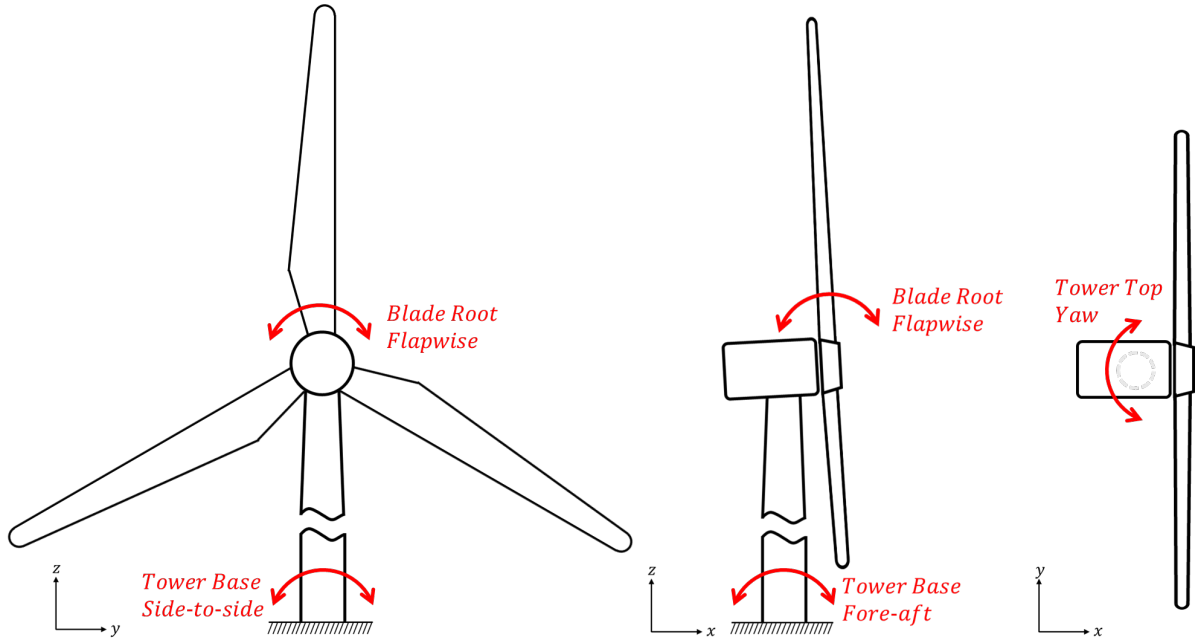

Figure 3.3: Conventions for the five turbine DELs.

is an additional requirement. Furthermore, since fatigue modelling requires Damage Equivalent Load (DEL) values, the model must output DEL values from post-processed time-series data. One final question that remains is of which components the DEL loads shall be predicted. Given the environment's purpose of modelling the fatigue effects of wake steering, three components are likely to be affected: the tower base, yaw system and blades. All of these three components are subject to alternating bending loads due to aerodynamic loading. Carroll et al. (2016) and Dao et al. (2019), furthermore, find that these are the structural components that are most often subject to fatigue failure. Additionally, the IEA37 3.4MW design paper highlights these areas in their fatigue study as well, indicating they were likely bottlenecks in the turbine design. As such, the model should consider tower base loads both from side-to-side and fore-aft, the loads on the yaw system and the edgewise and flapwise loads on the blade root. Each of these loads is a moment, and their conventions and locations are highlighted in Figure 3.3. The local power production and thrust coefficient must then complement the loads as output values to facilitate power production analysis and wake computations in PyWake. This summarises the following set of requirements:

1. The model shall be valid for the IEA37 3.4MW Reference Turbine
2. The model must be able to work in all three operating regions, e.g. below cut-in, at rated and above cut-out.
3. The model must be able to model 'parked' conditions
4. The model must be able to accept yaw angles in the range $[-30, +30]$
5. The input variables must comply with the IEC 61400-1 standard on design requirements for wind energy systems
6. The model must output DEL values for the tower base side-to-side and fore-aft, yaw system torsional and blade root edgewise and flapwise loads.
7. The model must output local (i.e. for each turbine) produced power and thrust coefficient.

3.3.1. Dataset Generation

The dataset must comprise pairs of inflow conditions and output variables from which the surrogate model can learn. The sampling of inflow conditions is first covered in subsubsection 3.3.1.1. Next, subsubsection 3.3.1.2 covers how OpenFAST, the comprehensive aerodynamic and structural calculation tool, should be set up correctly to calculate the turbine-level loads necessary for the dataset. Finally, subsubsection 3.3.1.3 covers what post-processing steps are required to obtain the final ground truth target values for the dataset.

3.3.1.1. Inflow Sampling

A full-field turbulence file must be generated to simulate the aerodynamic and structural response of a wind turbine in OpenFAST. This file contains the variations and turbulence of the wind in a 3D grid, where the x-axis essentially corresponds to the time dimension. OpenFAST will then translate this field over the wind turbine during the simulation to obtain the right time-dependent wind conditions at each point of the turbine. NREL provides a tool for generating such turbulence fields in the form of TurbSim (National Renewable Energy Laboratory (NREL), 2024b). Looking at the IEC 61400-1 standard and the TurbSim documentation, the three input variables needed to characterise inflow conditions are as follows:

- **Mean Wind Speed** V_m - The average over a given time-series of wind speeds.
- **Turbulence Intensity** I - The turbulence intensity, in fractions of the mean wind speed, of a given time series of wind speeds. Defined as $I = \frac{\sigma_w}{V_w}$, where σ_w is the standard deviation of wind speeds over the time series.
- **Shear Exponent** α - The exponent of the wind profile power law $\frac{u}{u_r} = \left(\frac{z}{z_r}\right)^{\alpha}$, relating wind speed u_r at a reference height z_r to the wind speed at any height z.

Note that wind direction does not yet play a role, as that can be characterised by the final input variable of **Yaw Misalignment** γ, the yaw angle of the turbine relative to the incoming wind.

For the **Mean Wind Speed** V_m, a uniform distribution over the range $[0, 30]$ is chosen. The IEA 61400-1 standard does not explicitly mention a range of wind speeds to be modelled; however, the selected range covers regions well outside the typical operating region of the IEA37 3.4MW turbine ($[5, 25]$) and should cover the vast majority of wind speeds that a regular wind farm will ever encounter in its lifetime. The reference height of V_m is the 3.4MW turbine's hub height. Next, for the **Turbulence Intensity** I, the 61400-1 standard prescribes, according to the Normal Turbulence Model (NTM), the relation $\sigma_1 = I_{ref}(0.75V_{hub} + b)$, where $b = 5.6m/s$ and I_{ref} is a parameter dependent on the wind turbulence characteristics, independent of wind turbine class. To enable the most extensive range of turbulence intensities to create the most robust model possible, the largest value (class A+ turbulence) of 0.18 is chosen. The relation defines σ_1, or the 90% quantile of wind speed standard deviations, and thus provides an upper bound for turbulence intensities. To obtain the intensity as a fraction, the relation can be divided by V_{hub} to obtain $I = I_{ref}(0.75 + \frac{b}{V_{hub}})$. Note that since the mean wind speed V_w above is defined at hub height, $V_{hub} = V_m$. Wind speed must be sampled prior to turbulence intensity, as one is dependent on the other. Following Haghi and Crawford (2023), a lower bound of 0.04 is set to ensure no perfectly smooth wind conditions occur, which would be unrealistic to begin with. Additionally, an upper limit of 0.5 is imposed, as the given relation's output will explode when approaching very low wind speeds.

Finally, the IEC 61400-1 standard prescribes a fixed and constant **Shear Exponent** α value of 0.2. However, since most loads depend on wind speeds at different heights above ground, the power law distribution of wind speeds significantly impacts results. As such, fixing α results in a surrogate model that is hardly robust to changes in wind shear. Instead, it is chosen to adopt the lower- and upper bounds defined by (Dimitrov, 2019), as they cover a wider range. Additional constraints are added in the form of a lower bound of -0.3 and an upper bound of 2.5 to tackle exploding values as wind speeds approach zero. Similar to turbulence intensity, the shear exponent is a function of the mean wind speed and must be sampled afterwards. The **Yaw Angles** γ are chosen uniformly in the range $[-30, +30]$. This is in accordance with the findings by Zong and Porté-Agel (2021), quoting maximum wake steering yaw angles to be about 30 degrees. Finally, since random turbulence fields must be generated using TurbSim, an additional seed value must be chosen. This seed value determines the random state of the processes within TurbSim. Its lower and upper bounds will be set to 0 and 2^{16}, respectively. An overview of the distributions of input variables is shown in Table 3.1.

A quasi-random SOBOL sequence is used to ensure that the samples are uniformly distributed throughout the input space and no bias is accidentally introduced. This allows for a better distribution of sample points over the input domain, eventually aiding in faster convergence during surrogate training. SOBOL sequences generate sets of values in the range $[0 - 1]$ for each variable to be sampled; together, these sets of values provide a proper distribution over the sample space. Since the input variables are being sampled uniformly between their lower and upper bounds, these SOBOL samples'

Symbol	Name	Lower Bound	Upper Bound
V_m	Mean Wind Speed	0	30
I	Turbulence Intensity	0.04	$min\left(0.5, I_{ref}\left(0.75+\frac{b}{V_{hub}}\right)\right)$
α	Shear Exponent	$max\left(-0.3, \alpha_{ref,LB}-0.23\left(\frac{V_{max}}{V_{hub}}\right)\left(1-\left(0.4log\frac{R}{z}\right)^2\right)\right)$	$min\left(2.5, \alpha_{ref,UB}+0.4\left(\frac{R}{z}\right)\left(\frac{V_{max}}{V_{hub}}\right)\right)$
γ	Yaw Misalignment	-30	30
ζ	Seed	0	2^{16}

Notes:
$I_{ref} = 0.18$ and $b = 5.6$, according to IEA61400-1 standard
$\alpha_{ref,LB}$ and $\alpha_{ref.UB}$ are 0.23 and 0.40 respectively, following (Dimitrov, 2019)
V_{hub} and V_{max} are hub-height and maximum possible wind speed respectively
R and z are rotor diameter and hub height in meters, and 130 & 110 for the 3.4MW turbine respectively.
All samples are chosen uniformly between lower- and upper bounds

Table 3.1: Overview of distributions of input variables

values can be applied according to $x_{i,lb} + s_i(x_{i,ub} - x_{i,lb})$, where $x_{i,lb}$, $x_{i,ub}$ and s_i are the lower bound, upper bound and SOBOL sample value for variable i respectively. SOBOL sampling maintains its uniform properties as long as the number of samples being drawn is a power of 2; as such, taking inspiration from the convergence study by Haghi and Crawford (2023), $2^{15} = 32768$ samples are generated. The input variables' distribution, bounds and relations are shown in Figure 3.4.

Given the sampled input parameters, turbulence fields can now be generated using TurbSim. For the output variables of local wind speed, turbulence intensity and power, average values of any time-series length can be used. For the DEL values, however, the time series length is an important parameter to pick. Usually, the DEL is determined based on a 10-minute time-series load history. However, given the large number of simulations and turbulence files that must be generated, it is chosen to limit the simulations to five minutes. This will save a significant amount of time and disk space when generating the dataset. This leads to the following assumption:

Assumption 2 *Damage Equivalent Loads (DELs) can be determined from 5-minute time-series calculations instead of 10-minute calculations without significantly compromising accuracy.*

In Appendix D the influence of running shorter time-series simulations is investigated through a small DEL-calculation convergence study over four different runs. From this quick study, it seems reasonable to assume that five-minute simulations to calculate the DEL will produce very similar results as running 10-minute simulations, and the assumption, therefore, seems acceptable. Anyhow, to remove any transient conditions at the start of the simulation, the first 20 seconds of all results are discarded, requiring an increased simulation time of 320 seconds. Several other parameters that were set in the TurbSim model are shown in Table 3.2. These stem from either the definitions outlined earlier, or from the simulation files provided by IEA Wind Task 37. A Python script was used to automatically and systematically run all sampled input parameter sets through TurbSim, using multiprocessing on 64 CPU cores. This process took approximately 12 hours and yielded 0.9TB of binary turbulence field files ready for processing in OpenFAST. With this, all input sampling has been done for the simulation step.

	Parameter	Value	Description
Fixed	WrADTWR	True	Whether tower time-series data points are created. Necessary for tower loads.
	NumGrid_Z	24	Vertical grid-point matrix dimension. Trade-off between generation time and resolution.
	NumGrid_Y	24	Horizontal grid-point matrix dimension. Trade-off between generation time and resolution.
	AnalysisTime	320	Length of time series in seconds, 300s (5min) plus 20 seconds of run-in period.
	HubHt	110	Turbine hub height in meters, 110 for the 3.4MW turbine.
	GridHeight	150	Grid height in meters, must be larger than the rotor diameter (130m).
	GridWidth	150	Grid width in meters, must be larger than the rotor diameter (130m).
	TurbModel	IECKAI	Turbulence model to use; Kaiman turbulence model according to IEC 61400-1
	IEC_WindType	NTM	Turbulence type to use; Normal Turbulence Model according to IEC 61400-1
	WindProfileType	IEC	Wind profile type to use; power law on rotor disk and log law elsewhere, according to IEC 61400-1
	RefHt	110	Reference height for mean wind speed; at hub height, as per aforementioned definition.
Variable	RandSeed1	ζ	Random seed to use
	IECTurb	$I \cdot 100$	Turbulence intensity, in percent
	URef	V_w	Mean wind speed at reference height
	PLExp	α	Power law exponent

Table 3.2: TurbSim parameters

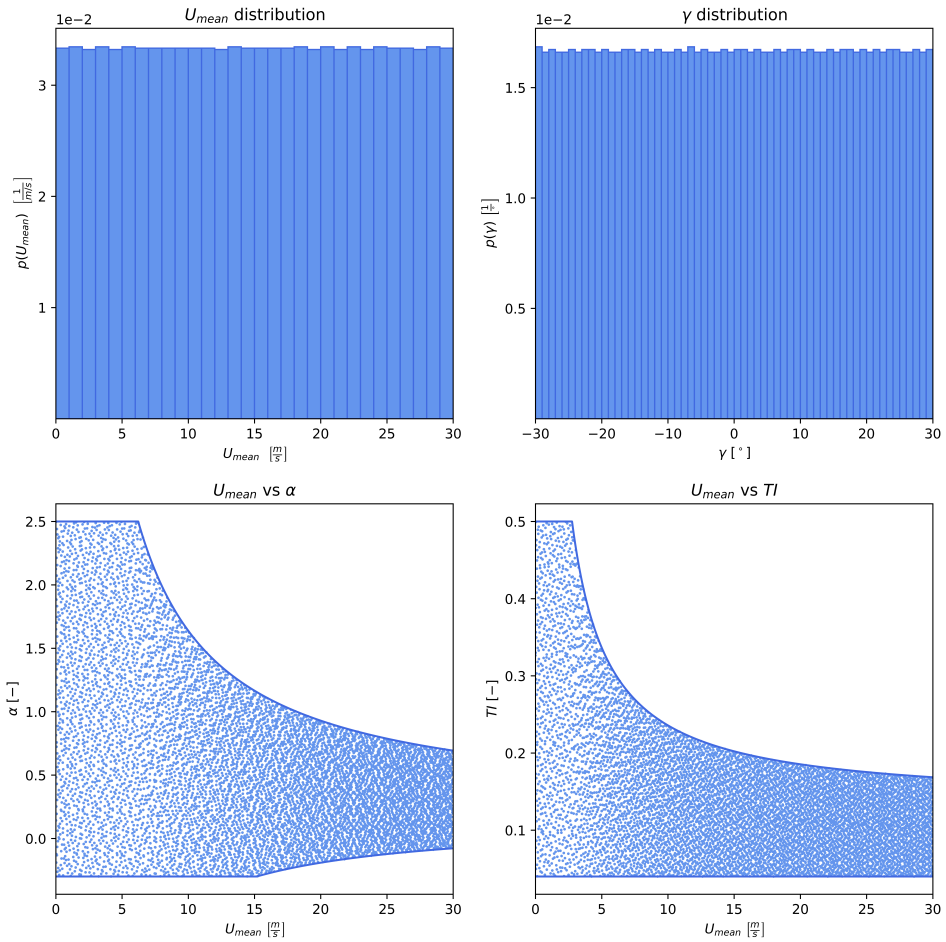

Figure 3.4: Distribution of input variables obtained using SOBOL sampling

3.3.1.2. OpenFAST Model Setup

The next step is to ensure a correct, accurate and proper working setup of the OpenFAST model. Fortunately, the files provided by Wind Task 37 are already set up correctly for running them through OpenFAST. Note that the IEA37 repository mentions that OpenFAST version 3.5.1 should be used, as will be done in the following section. The files provided in the wind turbine repository are the same files used to do the structural load studies for the design of the turbine itself and can be assumed to be accurate and realistic. However, a few things were changed or double-checked to ensure an accurate model.

First of all, a wind turbine controller is required to ensure the turbine's efficient operation. Typically, this is done through pitch control of the turbine blades to ensure that the turbine operates as closely to the ideal operating curve as possible. Fortunately, such a controller is readily available for OpenFAST in the form of ROSCO (NREL, 2024a), and the authors of the IEA37 3.4MW turbine have provided tuned parameters for said controller. ROSCO version 2.7.0 is used for the simulations as it is the supported version for the tuned parameters provided. All that remains is to provide the correct path to the ROSCO binary file in the `IEA-3.4-130-RWT_ServoDyn.dat` file.

As for the `IEA-3.4-130-RWT_ElastoDyn.dat` file, multiple changes were made. First of all, all three blade Degrees Of Freedom (DOF) were enabled to allow for the most realistic simulations. Next, the initial pitch angles for all three blades were set to 8 degrees. In the initial stages of the simulation, this might lead to some transient behaviour, but that will be removed by the initial 20-second window of data that is thrown away before postprocessing. Similarly, the initial rotor speed is set to 11.5 RPM. Both values were chosen arbitrarily to be in the middle of the limits of operation; ideally, both values would be set close to the steady-state values specific to the simulation conditions, but that is difficult to know beforehand. Instead, some transient behaviour is allowed at the beginning, which will balance

out towards steady state behaviour as the simulation proceeds. Finally, the file allows the provision of a list of output parameters that should be stored in a file after simulation. Here, it is crucial to provide all variables that could be of interest for load or power calculations. Note that a complete list of available output parameters and corresponding descriptions can be found in the OpenFAST documentation. The most important output parameters that were used and their descriptions are shown in Table 3.3. Note that the loads are still time-series values and thus need postprocessing to obtain the DELs.

Parameter	Description	Unit
RotPwr	Rotor Power	$[kW]$
HSShftPwr	High-speed shaft power	$[kW]$
RootMxb1	Blade 1 blade root edgewise moment	$[kNm]$
RootMxb2	Blade 2 blade root edgewise moment	$[kNm]$
RootMxb3	Blade 3 blade root edgewise moment	$[kNm]$
RootMyb1	Blade 1 blade root flapwise moment	$[kNm]$
RootMyb2	Blade 2 blade root flapwise moment	$[kNm]$
RootMyb3	Blade 3 blade root flapwise moment	$[kNm]$
YawBrMzn	Tower-top (yaw bearing) yaw moment	$[kNm]$
TwrBsMyt	Tower-base pitching (fore-aft) moment	$[kNm]$
TwrBsMxt	Tower-base roll (side-to-side) moment	$[kNm]$

Table 3.3: `ElastoDyn` output parameters

One small change was made in the main simulation input file `IEA-3.4-130-RWT.fst`. The runtime of the simulation, TMax, was set to 320 seconds to correctly match the length of the time-series turbulence field input file. Finally, some other minor changes that were made included the addition of the RtAeroCt output parameter in the `IEA-3.4-130-RWT_AeroDyn15.dat` file to capture the thrust coefficient and setting the TurbSim binary output file type as the correct input in `IEA-3.4-130-RWT_InflowFile.dat`. All other files and parameters were kept identical to the ones provided in the official IEA37 3.4MW reference wind turbine repository.

Now, to simulate all cases generated earlier, sets of input files and TurbSim turbulence files must be coupled together, and the correct file paths and parameters must be set. The `openfast-toolbox` Python package was used to do this automatically. For each case, and therefore each TurbSim turbulence file, a copy of the input files was made and adapted accordingly. Changes made to these input files were mainly ensuring the parameter for the turbulence field input file in `IEA-3.4-130-RWT_InflowFile.dat` pointed to the correct TurbSim file and that the correct turbine yaw angle (NacYaw) was set in the file `IEA-3.4-130-RWT_ElastoDyn.dat`. Furthermore, outside the regular operating region (4 to 25 $\frac{m}{s}$ wind speed), several additional parameters were changed. First of all, the pitch control, variable-speed control, generator and generator DOF were turned off to simulate parked conditions. Secondly, for parked conditions and wind speeds higher than 25 $\frac{m}{s}$, the blade pitch was set to 'fully feathered', e.g. 90 degrees, as is customary during such conditions to protect the turbine. Finally, below cut-in speed (4 $\frac{m}{s}$), the blade pitch was set to zero. `openfast-toolbox` generates and adjusts all input files where necessary and places them in a separate folder. All cases were simulated using OpenFAST version 3.5.1 using the terminal command `openfast <case_file_name>.fst`, running 64 processes in parallel. This took about 24 hours and yielded several gigabytes of binary output data containing all time-series outputs for each of the 32.768 simulations.

3.3.1.3. Postprocessing

The obtained time-series values of the simulations now need post-processing to turn them into a set of usable ground truth values for surrogate training. Looking back at the requirements, DEL values for five different locations on the turbine must be determined. These DEL values are derived from the time-series load values and can thus be seen as a summary of the history of load cycles. Generally, this involves using rainflow counting to determine the number of cycles and their respective magnitudes. Fortunately, the `openfast-toolbox` Python package mentioned earlier provides functions that can perform this rainflow counting and DEL calculation. First of all, as mentioned earlier, the first 20 seconds of the simulation are discarded to remove any transient startup effects. This leaves 300 seconds, or five minutes of time-series data to be analysed. The inflow conditions and yaw angle are derived from the input data and filenames; these will form the input vector X, which needs no further processing.

The second part of the dataset is the ground truth outputs, denoted as Y. For power production, the mean power over the entire time series is considered the best way to obtain a singular representative

value. The same method is also applied to the thrust coefficient for the same reason. For the DELs, `openfast-toolbox`'s `equivalent_load` function is used. The function takes the time-series load data together with the Wöhler exponent m as input. The Wöhler exponent is typically a material parameter and is related to how steep the fatigue curve is. A typical Wöhler exponent for steel materials is 4; for composite materials, this value is 10. As such, for the tower-base side-to-side and fore-aft DELs, $m = 4$ is used; for the blade root edgewise and flapwise DELs, $m = 10$ is used instead. For the tower top yaw bearing, however, a value of $m = 7$ is used. Further explanation for the choice of these values can be found in chapter 4. Finally, since DELs are always defined at some reference cyclic speed, the default value of $1Hz$ is used as such. This means the output DEL indicates a load magnitude that, when applied at $1Hz$, applies the same amount of damage as the entire load history over that period. Since there are three blades, the DEL value is calculated for each of their load time series separately; the three resulting values are then averaged. All in all, there are seven output values: the power, thrust coefficient and DEL values for each of the five (sub-)components. This is summarised in Table 3.4.

	Variable	Unit	Description
Input (X)	V_w	$[m/s]$	Mean wind speed over 5-minute window
	I	[-]	Turbulence intensity
	α	$[-]$	Shear exponent of power-law wind speed distribution
	γ	$[^\circ]$	Yaw angle of the turbine relative to the wind
Output (Y)	P	$[kW]$	Mean turbine power over 5-minute window
	C_t	$[-]$	Mean thrust coefficient over 5-minute window
	$DEL_{br,ew}$	$[kNm]$	Three-blade mean blade root edgewise DEL
	$DEL_{br,fw}$	$[kNm]$	Three-blade mean blade root flapwise DEL
	$DEL_{tt,yaw}$	$[kNm]$	Tower top yaw DEL
	$DEL_{tb,ss}$	$[kNm]$	Tower base side-to-side DEL
	$DEL_{tb,fa}$	$[kNm]$	Tower base fore-aft DEL

Table 3.4: Dataset description for turbine-level surrogate

All postprocessed outputs are shown in Figure 3.5 and Figure 3.6, plotted against wind speed and yaw angle, respectively. Note that power and thrust coefficient values are only valid inside the operating region; outside of this, the turbine is either parked or standing still, resulting in no power production. Indeed, the dependency on the yaw angle clearly shows that certain loads can increase or decrease as the yaw angle changes.

3.3.2. Surrogate Model

Now that the dataset is ready the architecture and training of the surrogate model itself must be defined. Both of these are essential to facilitate effective training and yield accurate models. As such, subsubsection 3.3.2.1 will first cover the architecture chosen for the surrogate model and the accompanying hyperparameters used in the training process. Finally, following training using the chosen architecture and parameters, subsubsection 3.3.2.2 covers evaluation metrics highlighting the accuracy of the trained surrogate model.

3.3.2.1. Architecture & Hyperparameters

The neural network architecture follows the typical architecture of a fully connected MLP, consisting of several hidden layers with multiple hidden nodes with activation functions. The network has three hidden layers with 32, 64 and 32 nodes, respectively. There are four input nodes corresponding to the four input parameters highlighted in Table 3.4; furthermore, there is one output node. This means that each variable has its own network, rather than all outputs sharing a common network. The activation function used in the nodes is the `LeakyReLu` function. The output layer does not have an activation function and is, therefore, linear. Furthermore, three networks are trained for each variable: one for the below cut-in wind speeds, one for the operating region's wind speeds, and one for parked conditions. The latter is shared between the above cut-out wind speeds and 'true' parked conditions where the turbine is shut off for maintenance. These last cases share the same OpenFAST model setup and can, therefore, be inferred from the same surrogate model.

All input and output values are first normalised using `min-max` scaling, meaning they are mapped and, therefore, scaled to the region $[0, 1]$. The dataset is split into a training and validation set, with

20% of the data points going towards the validation split. Mean Squared Error (MSE) loss is used as a loss function, combined with Adam as optimiser. All training is done using PyTorch. The learning rate is set to 0.001, and the training is left to run for 3000 epochs. The network is saved if and only if the validation loss decreases, preventing overfitting on the training dataset. All hyperparameters are shown in Table 3.5; the number of hidden nodes, the activation function, the learning rate and training epochs were tuned manually to obtain the lowest training loss. The loss function, min-max scaling and optimiser were chosen as the typical default options for PyTorch neural networks. The training of the surrogates takes approximately one minute on an RTX A100 laptop GPU.

Parameter	Value
Number of input nodes	4
Number of hidden layers	3
Number of nodes in hidden layers	[32, 64, 32]
Number of output nodes	1
Activation function	LeakyReLu
Input/Output transform	Min-Max Scaling
Train / Validation Split	80% / 20%
Learning Rate	1e-3
Loss function	MSE
Optimiser	Adam
Framework	PyTorch
Epochs	3000
Learnable Parameters	4385
Early Stopping	Validation loss

Table 3.5: Overview of model architecture and hyperparameters for the turbine-level surrogates.

3.3.2.2. Evaluation

After training, the turbine-level surrogate is used for inference on the entire input dataset to investigate its performance visually; this is shown in Figure 3.7. It is evident that the plots showing the relations between the output variables and wind speed or yaw angle match those plotted from the input dataset in Figure 3.5 and Figure 3.6 nicely. Furthermore, the figures plotting the ground truth values versus the predictions indicate a close match with the $y = x$ line, which indicates good performance; this is further proven by the calculated r^2 values shown in the plots. Furthermore, the distributions of absolute errors are centred around zero and show low absolute errors.

To properly quantify the performance of the trained surrogates, some performance metrics are calculated in Table 3.6. The surrogates for power and thrust coefficient score very well, with Mean Average Percentage Errors (MAPE) of only two per cent. Generally, the surrogates have more trouble predicting the DEL values, as these evidently show higher relative errors. However, judging by the low magnitude of the Mean Average Error (MAE), most of the contribution of the relative errors comes from cases where the ground truth value approaches zero, and any non-zero prediction will cause an explosion of the respective relative error. Nevertheless, their performance seems quite reasonable and aligns with work by Dimitrov (2019). The trained turbine-level surrogates are now suitable for inclusion in the dataset generation for the farm-level surrogate.

Model	r^2 score	RMSPE	RMSE	MAPE	MAE
P	0.998	0.067	50.388	0.020	25.787
C_t	0.999	0.030	0.008	0.022	0.005
$DEL_{br,ew}$	0.992	0.144	43.483	0.053	27.002
$DEL_{br,fw}$	0.981	0.230	256.816	0.092	167.69
$DEL_{tt,yaw}$	0.974	0.350	219.263	0.134	122.06
$DEL_{tb,ss}$	0.960	0.178	1631.687	0.117	924.066
$DEL_{tb,fa}$	0.943	0.139	2299.924	0.093	1093.924

Table 3.6: Performance metrics for all seven surrogates averaged over all operating regions.

3.4. Farm-Level Model

Now that a suitable turbine-level surrogate has been trained, it can be combined with the farm-level model (PyWake) to generate a dataset to train the farm-level surrogate. Similar to the turbine-level surrogate, some requirements are first defined:

1. The model shall be valid for the same inflow conditions as the turbine-level surrogate
2. The model must work in all three operating regions, e.g. below cut-in, at rated and above cut-out.
3. The model must be able to model 'parked' conditions
4. The model must be able to accept yaw angles in the range $[-30, +30]$
5. The model must generalise to any wind farm layout

3.4.1. Dataset Generation

Following the same process from the turbine-level surrogate, first, the inflow conditions and input farm layouts are generated in subsubsection 3.4.1.1. Next, in subsubsection 3.4.1.2, the correct model setup for PyWake is chosen to produce the most accurate and realistic results. Finally, in subsubsection 3.4.1.3, the outputs from the simulations are then cast into the right input and output format for learning on graphs.

3.4.1.1. Inflow & Layout Sampling

Dataset generation for the farm-level surrogate requires sampling both inflow conditions and farm layouts. Fortunately, inflow sampling can use the same method as prescribed in subsubsection 3.3.1.1, meaning a new set of inflow conditions can be generated relatively quickly. The major difference with before is that, in this case, no turbulence files need to be generated, and the three-variable inflow information is sufficient as-is. The focus in this section is, therefore, on the generation of the random farm layouts.

A large variety of wind farm layouts must be generated for the training dataset to ensure a robust and layout-agnostic farm surrogate model. This ensures that the model generalises as best as possible to any farm layout. To do so, it must consist of various shapes, numbers of turbines, inter-turbine spacing and arrangement. Furthermore, some constraints that enforce, for example, a minimum inter-turbine spacing and/or a minimum number of turbines per layout must be imposed. Layout generation begins by randomly sampling several parameters that will be used in the layout-building process; these parameters and their bounds are shown in Table 3.7. These values are inspired by the values chosen by Duthé et al. (2023) and based on typical values for real-life wind farms. Similarly to inflow generation, sampling uses uniform sampling with SOBOL sequences to ensure an even spread over the available input domain.

Symbol	Name	Lower Bound	Upper Bound
n_p	Number of candidate turbines to spawn	25	100
D_{min}	Minimum turbine spacing, expressed in rotor diameters	2	8
r_{lw}	Length-over-width ratio for candidate turbine field generation	0.5	4

Notes:
Since D_{min} is expressed in rotor diameters, the actual spacing is $D_{min} \cdot R$ m
All samples are chosen uniformly between lower- and upper bounds

Table 3.7: Overview of bounds of input parameters

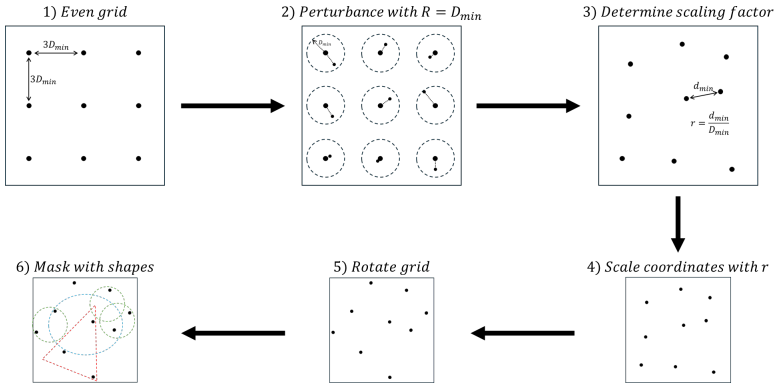

Figure 3.8: Steps of generating the rectangular domain of randomly perturbed points.

Layout generation proceeds by initialising n candidate turbines over a rectangular domain characterised by a length-over-width ratio r_{lw}. This will create evenly spaced points that are placed three times the minimum turbine spacing apart along vertical and horizontal lines over the rectangular domain. Next, the previously generated points are perturbed to a position within a radius of D_{min} uniformly at random around their original position. To enforce the minimum turbine spacing, the ratio $r = \frac{d_{min}}{D_{min}}$ is calculated, where d_{min} is the actual closest distance between any two points in the perturbed rectangular candidate turbine field. All coordinates are then scaled by this factor r to ensure d_{min} matches D_{min} exactly. The entire grid of points is then rotated by some random angle β, sampled randomly from the range $[-45, 45]$. All previous steps combined yield a rotated grid of perturbed points; this grid is, however, still relatively even and rectangular. To introduce more random shapes into the dataset, this grid is then masked using three types of masks: a 'circles' mask, an elliptical mask and a triangular mask. The 'circles' mask consists of three circles at different positions and radii; the triangular mask consists of a single triangle at some random position with random size and rotation, and the elliptical mask of an ellipse at a random position with random rotation and size. Masking the previously generated points with these masks then provides three different layouts. The grid generation process is shown in Figure 3.8; the masking process in Figure 3.9.

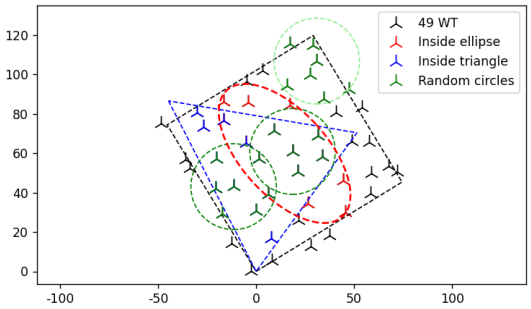

Figure 3.9: Process of masking the rectangular grid with different shape masks.

Now that the layout has been generated, each of the turbines in the layout can be given a yaw angle. Additionally, each turbine is given an operating mode, being either in operation or parked to simulate maintenance. Yaw angles, similarly to inflow sampling in subsubsection 3.3.1.1, are sampled uniformly using SOBOL sequences from the range $[-30, +30]$. Operating modes, however, are sampled using a categorical distribution with probabilities. For this layout generation, the probability of any turbine being parked is set to 5%, meaning, on average, 5% of all turbines are parked. This operating mode will cause the turbine to generate no wakes, generate no power and have loads associated with a parked turbine when the mode is set to non-operating. This allows the surrogate to learn the effects of one or more turbines being turned off, allowing it to simulate turbines under maintenance in the simulation environment. Using these settings and methods, 5000 different layouts are generated with ten inflow conditions each, yielding 50,000 different training cases. Similarly, 15,000 cases are generated for the test, and the validation is set separately.

3.4.1.2. PyWake Model Setup

The next step is to find the correct model setup for PyWake. Unlike the case for the turbine-level simulations using OpenFAST, no ready-to-use setups are available for this specific case. Instead, these must be chosen based on the aforementioned requirements and constraints. There are essentially four setup choices that have to be made: the wake deficit model, the deflection model, the superposition model and the turbulence model. Each of these modules has a variety of options to choose from; internally, in PyWake, they are used to model the wake effects in the farm.

The deficit model calculates the deficit in wind speed within the wake effects, and PyWake offers a wide variety of models to choose from. Fortunately, the same IEA Wind Task 37 project that produced the 3.4MW reference turbine also conducted research into farm-wide power optimisation (Quaeghebeur et al., 2021), which can be consulted for their wake modelling choices. In their work, they made use of the wake deficit model proposed by Bastankhah and Porté-Agel (2014), which happens to be implemented and available in PyWake as the `BastankahGaussianDeficit` module. For the turbulence model, four options are available. From these options, the `CrespoHernandez` module, implemented according to Crespo and Herna´ndez (1996), is chosen, following Duthé et al. (2023). Furthermore, the superposition model adds up the overlapping wake effects of multiple turbines. For this, PyWake's `SquaredSum` module is chosen.

The wake deflection module determines the deflection of the wake effects due to yaw angles. This module is vital for the simulation, as these deflections are precisely what wake steering is based on. PyWake offers three different wake deflection models, of which two have citations referring to the work according to which they were implemented. Since the underlying relations must be valid across the entire proposed region of available yaw angles $[-30, +30]$, the underlying assumptions and regions of validity must be studied. This already eliminates the `FugaDeflection model`, as the PyWake documentation provides no citation to the underlying paper. Judging from the papers behind the other two options, Larsen et al. (2020) and Jiménez et al. (2010), it becomes apparent that the former (`GCLHillDeflection` in PyWake) was only derived from experiments with a maximum yaw angle of 17.5. The latter, `JimenezWakeDeflection`, was derived from experiments with maximum yaw angles of 30 degrees. The only suitable wake deflection module is thus PyWake's `JimenezWakeDeflection`.

Furthermore, PyWake's internal experimental surrogate load models are replaced by the turbine-level surrogates created in section 3.3. PyWake's default turbine model was replaced by a custom one, with an additional 'operating mode' option to facilitate the simulation of parked conditions. Since PyWake's original surrogates already had support for a three-region surrogate, minimal adjustment was necessary to support the newly trained surrogates. All 80,000 generated cases can now be simulated using the PyWake model. Using parallel processing over 8 CPU cores, this process takes several minutes. This yields as output the local power, wind speed, turbulence intensity and the five fatigue loads for each turbine in the farm, considering wake effects and free-stream wind conditions. This data can now be post-processed to obtain the training dataset for the farm-level surrogate.

3.4.1.3. Postprocessing

The GNN is trained on graphs, but the previously generated data is not yet represented as such. The inputs and outputs must, therefore, be represented as an input graph and an output graph. To do so, each turbine is represented as a node on a fully connected graph. Multiple connectivity types are possible, but the fully connected type will likely propagate wake effects over the graph the best, at the cost of more network parameters. For the input graph, each node is given a feature vector consisting of its yaw angle, operating mode and the free-stream inflow conditions characterised by wind speed, wind direction, turbulence intensity and shear exponent. For the output graph, each node is given a feature vector consisting of the local power, wind speed, turbulence intensity and the five fatigue loads. The edges between nodes of the input graph each get a feature vector containing a polar representation of the vector between them (e.g. angle and distance) and the angle of the wind direction relative to the angle from the polar representation. For more information on this edge representation, the work by Duthé et al. (2023) on which this GNN surrogate is based can be consulted. Note that this thesis adds additional node features in the form of yaw angle and operating mode, and thus deviates slightly from their work. Since no edge features need to be predicted by the GNN, output graph edge features are irrelevant and, therefore, omitted. What results is a dataset consisting of input graphs with node- and edge features and output graphs with only node features, ready for training.

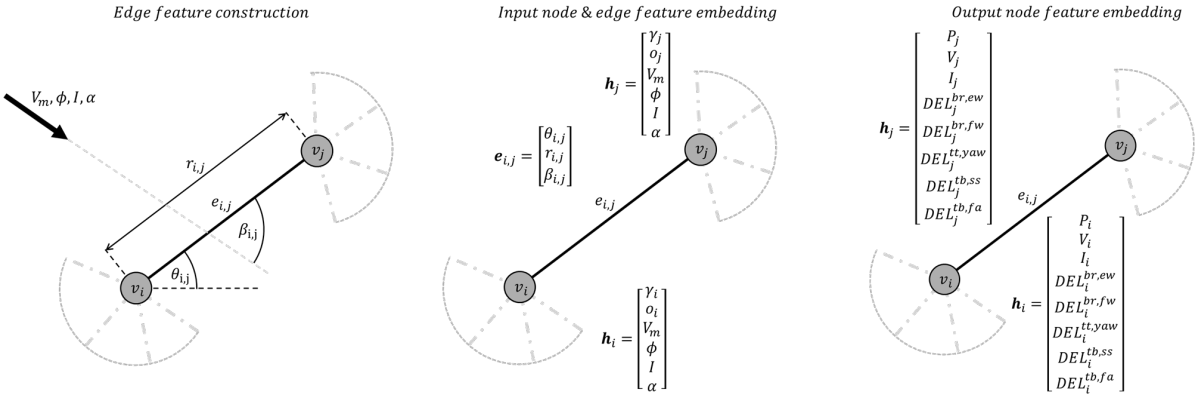

Figure 3.10: Graph embedding of dataset

3.4.2. Surrogate Model

With the farm-level dataset now generated and embedded in the form of graphs, all is ready for the training of the farm-level surrogate model. First, the architecture and hyperparameters for the model are chosen and defined in subsubsection 3.4.2.1. Following training using the provided architecture and parameters, subsubsection 3.4.2.2 covers the evaluation of the trained surrogate through evaluation metrics highlighting its accuracy.

3.4.2.1. Architecture & Hyperparameters

Following the architecture from work by (Duthé et al., 2023) that the surrogate is based on, the network architecture is built upon an encode-process-decode principle. Node and edge features are encoded into a higher-dimensional latent space, better suited for graph aggregation, using MLP-based encoders. Next, several layers of a graph convolution block propagate features over the graph using message-passing techniques. Finally, a node decoder decodes the resulting output latent space back into an interpretable node feature vector.

The node encoder consists of a LeakyReLu-activated 3-hidden-layer MLP with 256, 512 and 256 hidden nodes, encoding node features onto a latent vector of length 256. This node encoder is based on the turbine-level surrogates developed earlier in this chapter, as they are likely to largely perform the same type of calculations. It is, therefore, logical to employ a similar architecture. The edge encoder consists of a ReLu-activated 2-hidden layer MLP with 256 hidden nodes in each layer, encoding edge features onto a latent vector of length 256. Once the latent state has been encoded on the graph, four layers of GENeralized Graph Convolution (GEN, Li et al. (2020)) with SoftMax aggregation propagate features over the graph using message passing. Finally, a ReLu-activated node decoder MLP with two hidden layers of size 256 decodes the latent state into the output vector of size 8.

Furthermore, all inputs (node and edge features) are standardised using mean-standard deviation standardisation. The mean and standard deviation are calculated over the entire training dataset. The learning rate is set at 0.0005 initially but is adjusted using Cosine Annealing over 100 training iterations, after which it stays fixed. 150 training epochs (iterations) are done, with a batch size of 100. The Adam optimiser is used to update the network parameters using a Mean Squared Error (MSE) loss function. All hyperparameters are summarised in Table 3.8. All hyperparameters were chosen according to Duthé et al. (2023) and further manually tuned to achieve the lowest validation loss.

Parameter	Value
Node feature dimension	2
Node encoder hidden layers	3
Node encoder layer dimensions	[256, 512, 256]
Node encoder activation function	LeakyReLu
Node latent dimension	256
Edge feature dimension	3
Edge encoder hidden layers	2
Edge encoder layer dimensions	[256, 256]
Edge encoder activation function	ReLu
Edge latent dimension	256
GNN layer type	GEN
GNN layer aggregation	SoftMax
Number of GNN layers	4
Node decoder hidden layers	2
Node decoder layer dimensions	[256, 256]
Node decoder activation function	ReLu
Node output dimension	8
Input/Output transform	Mean-Std Standardisation
Number of graphs in train set	50000
Number of graphs in test set	15000
Number of graphs in validation set	15000
Initial learning rate	0.0005
Learning rate scheduler	Cosine Annealing
Learning rate scheduler stop	100 epochs
Loss function	MSE
Optimiser	Adam
Framework	TorchGeometric+PyTorch
Epochs	150
Learnable parameters	1,650,440
Early Stopping	Validation loss

Table 3.8: Overview of model architecture and hyperparameters for the farm-level GNN surrogate.

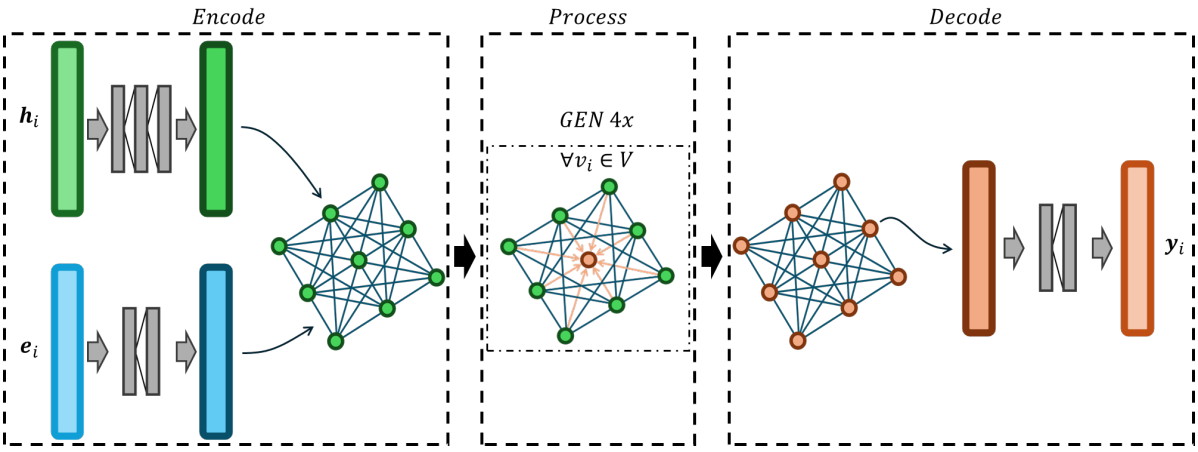

Figure 3.11: Overview of farm-level GNN surrogate, with the encode-process-decode paradigm.

3.4.2.2. Evaluation

Training the farm-level GNN surrogate takes about six hours running on an RTX4090 GPU with 24GB of RAM. The training curves, both training loss and validation loss, can be found in Figure 3.12.

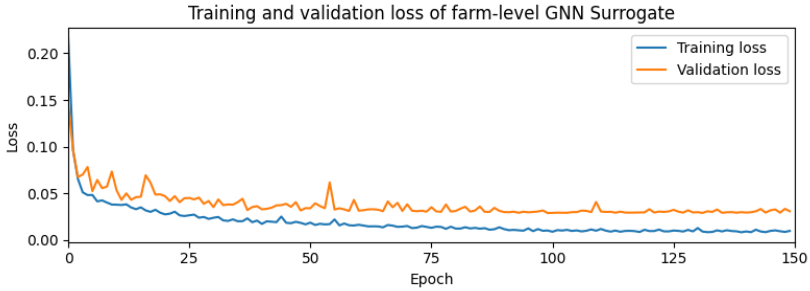

Figure 3.12: Training curves of the farm-level GNN Surrogate.

Following the same performance analysis from section 3.3, all predictions on the test set can be plotted for each output variable. This is shown in Figure 3.13. Note that all of the output variables are local to the turbines, e.g., they include wake effects from other turbines. Each point in these graphs is a value for a single turbine under a single wind condition in a single farm layout. For the most part, the same shape of the graphs can be seen that was also present for the turbine-level surrogate outputs. This is logical, as the surrogate still has to predict the same turbine structural loads; the only difference is the local flow condition inputs, which have to be implicitly determined due to wake effects. One interesting behaviour, though, is the behaviour around discontinuous sections. Around the cut-in and cut-out wind speeds of 4 and 25 m/s, respectively, the surrogate has difficulty modelling the immediate changes in structural behaviour due to the turbine transitioning to parked behaviour. It essentially fits a continuous curve on a discontinuity, causing some data points in the sudden transition to be predicted in the middle of the discontinuity.

This same behaviour can also be seen in the third column of figures, showing the predictions versus the ground truths. Furthermore, there is some noise in the training set, which gets smoothed out by the surrogate, causing several data points (which are outliers in the training set) to be predicted closer towards the majority of data points. Still, the excellent values for r^2 indicate that this merely concerns some outliers and that the vast majority of data points are still correctly inferred. A summary of the r^2 score, RMSE and MAE values is shown for all output variables in Table 3.9. Note that, unlike Table 3.6, the relative scores (e.g., percentages RMSPE and MAPE) are omitted here. This is due to the fact that, in data points where the ground truth is zero or approaches zero, the relative error tends to explode towards infinity. This yields mean relative errors that significantly misrepresent the true performance of the surrogate, as they overpower the 'true' relative errors of the rest of the dataset.

Variable	r^2 score	RMSE	MAE
P	0.996	25.957	21.929
$V_{W,l}$	0.995	0.121	0.104
TI_l	0.980	0.006	0.005
$DEL_{br,ew}$	0.995	47.463	39.760
$DEL_{br,fw}$	0.986	191.774	157.324
$DEL_{tt,yaw}$	0.986	103.754	85.778
$DEL_{tb,ss}$	0.975	601.581	464.686
$DEL_{tb,fa}$	0.984	902.543	722.481

Table 3.9: Performance metrics for the farm-level GNN Surrogate model on the test set; subscripts $_l$ indicate that wind speed V_w and TI are the ones local to the turbines, e.g. with wake effects. Relative errors are omitted due to value explosions when ground truth values approach zero.

3.5. Chapter Recap

In this chapter, a model to simulate entire wind farms under varying wind and yaw conditions was developed. To enjoy inference speeds that abide by the design objectives of the environment, a custom GNN-based surrogate model was developed. This model was trained using a dataset of various wind farm layouts under various inflow and yaw conditions, containing flow conditions as inputs and local turbine variables such as power and loads as output. To create this dataset, PyWake (for farm-level wake effects) was combined with a custom surrogate (for turbine-level power and load effects). This

custom turbine-level surrogate was trained using a dataset of various local flow conditions and yaw angles obtained by running the IEA37 3.4MW reference turbine through OpenFAST along with various flow fields. The result of this chapter is a fast and accurate GNN-based surrogate model that can hereafter be implemented in the environment to infer the effects of a given yawing strategy. As such, to answer the sub-question *How can the impact of wake steering on power production and fatigue loads be efficiently modelled in the context of a multi-turbine wind farm*: using a surrogate GNN farm-level model trained using a dataset constructed by a combination of PyWake and a custom turbine-level surrogate. Using the fatigue loads that can be determined using the surrogate model, the next chapter can proceed with the defining of a fatigue accumulation model.

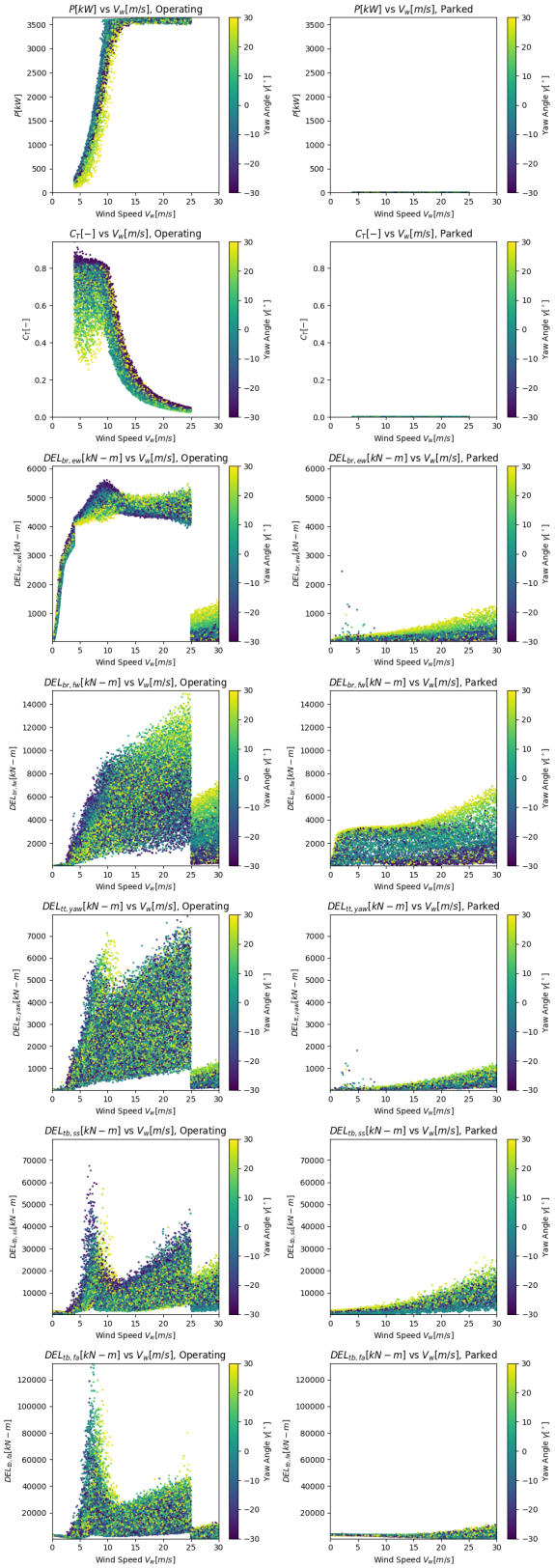

Figure 3.5: Outputs versus Wind Speed

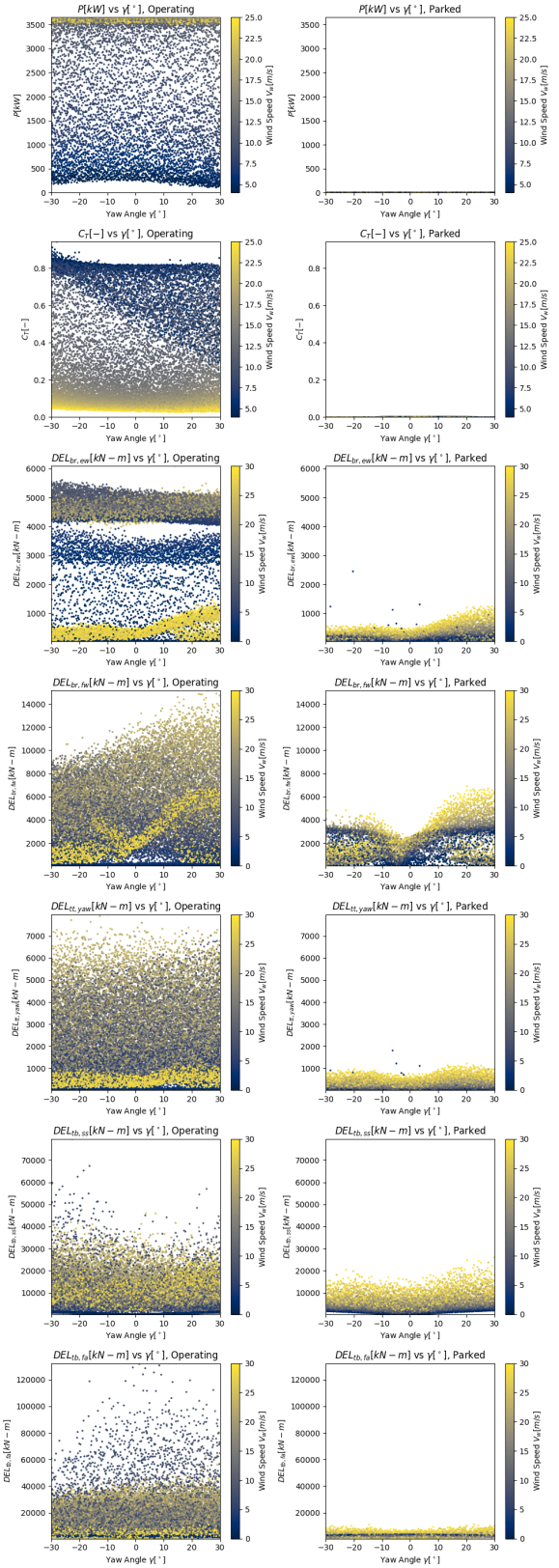

Figure 3.6: Outputs versus Yaw Angle

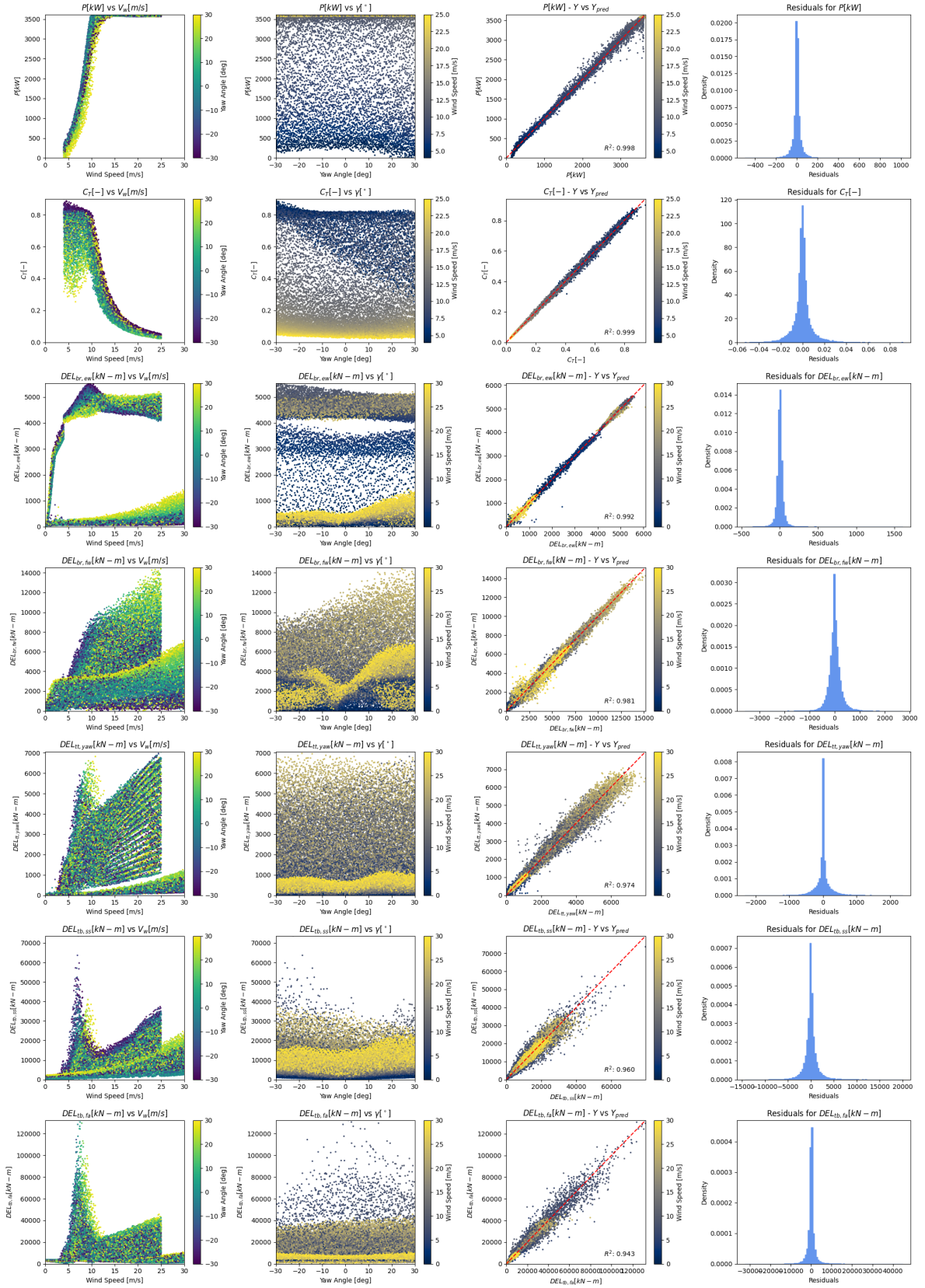

Figure 3.7: Output analysis of the trained turbine-level surrogate

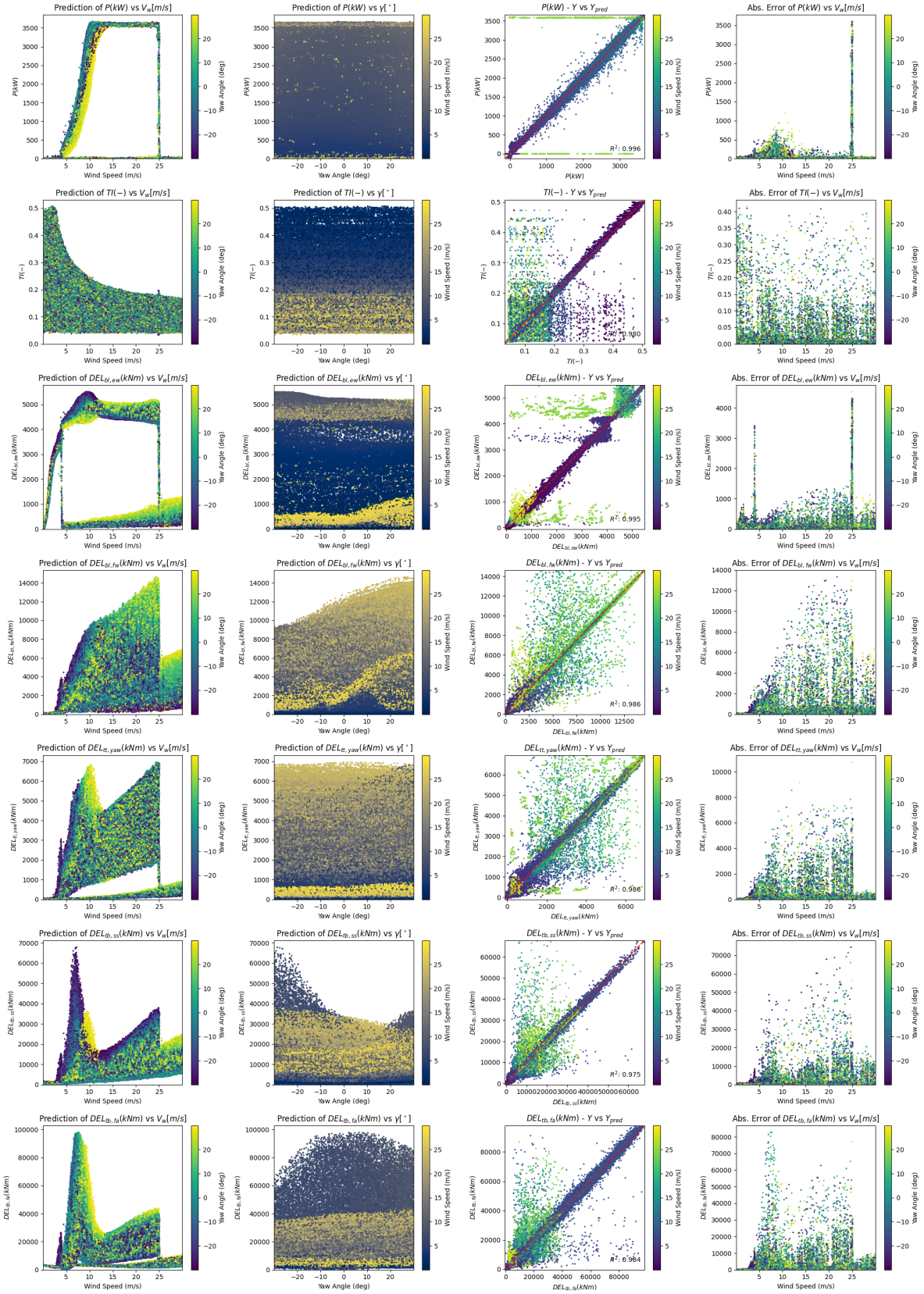

Figure 3.13: Output analysis of the trained farm-level surrogate on the test set.

4

Fatigue Modelling

Before the environment can be constructed, the method by which fatigue is modelled must first be determined. The surrogate model developed in the previous chapter provides as output the Damage Equivalent Load (DEL) for each of the five considered areas, given the local flow conditions around the turbine. The task at hand now is to map this 'fatigue load' to long-term effects in the form of accumulation of effects leading to component failure. In this chapter, section 4.1 will cover the theory used for the damage accumulation method, after which section 4.2 will describe the process of finding the required parameters to characterise the accumulation process.

4.1. Damage Accumulation Theory

For methods on how to accumulate fatigue damage, the IEC 61400-1 standard on wind turbine design requirements (International Energy Agency (IEA), 2019) can be consulted. The standard dictates that Palmgren-Miner's linear damage accumulation rule, or Miner's rule in short, can be used for fatigue analysis. Miner's rule can be used to determine the accumulation of 'damage' of a component; when this damage value reaches the limit state of 1, the component is assumed to have failed. This accumulation of damage is calculated in fractions, where each period of load cycles contributes to an increment of damage. Increments are dependent on both the magnitude of the cyclic load and the number of cycles.

Miner's rule can be formally defined as $\sum_i \frac{1}{N(S_i)}$, where S_i indicates the load range for a single load cycle i; $N()$ indicates the number of cycles to failure for a given constant amplitude loading with the range given by the argument. Miner's rule is typically expressed in terms of stress rather than loads. However, given that the previously created surrogates provide cyclic loads as output, Miner's rule can be rewritten to work with loads instead: $\sum_i \frac{1}{N(DEL_i)}$. As the DELs from previous chapters were defined as 10-minute equivalent loads at 1 Hz, sections of 600 cycles can be converted directly. More generally, this means that n cycles of a DEL_i can be converted as $\sum_i \frac{n}{N(DEL_i)}$.

What remains is to determine the parameter(s) of the function $N()$ that determines, for any given DEL, the number of cycles which would lead to failure. In prior research by NREL (Freebury and Musial, 2000), this issue was addressed by fitting a function of the form $M_a = M_u \cdot N^{\frac{-1}{m}}$ to fatigue testing data. Here, M_a is the amplitude moment of the load cycle, M_u the ultimate moment (e.g., the moment which would result in instant failure), N the allowable cycles to failure and m the Wöhler exponent. Since the DELs from the surrogate are defined as moments, this function is valid for DEL-based fatigue calculations as well. Indeed, this is what NREL implemented in their tool for fatigue analysis called MLife (NREL). MLife uses fatigue analysis rules outlined by the IEC 61400-1 standard, the same as referred to above. What remains is thus to determine what values for DEL_u (ultimate moment) and m to use for each component to characterise the fatigue curves fully.

4.2. Fatigue Curve Parameters

In the appendix of the design paper of the IEA37 3.4MW reference turbine (International Energy Agency (IEA), 2016) the results of several fatigue calculations can be found. Usually, with these values, the

parameters of the fatigue curves can be inferred by solving simple equations. However, closer inspection of the corresponding GitHub repository with up-to-date files and calculations reveals that they have done updated calculations, significantly increasing the load envelope. This is likely due to changes in the design or updated structural simulations with improved model setups. Consequently, any fatigue calculations found in the design paper use outdated values. In fact, the load envelope in the design paper seems to be an order of magnitude smaller in certain places compared to the updated values in the repository. Consequently, this means that the fatigue lifetimes and loads reported in the document's appendix are no longer valid and will, combined with determined turbine-level fatigue loads, result in unrealistic failure rates. It is thus evident that these fatigue values cannot be used to infer the parameters of the fatigue curve. However, one parameter that can still be taken from the design paper is the value for the Wöhler exponent m. This is fixed and set to 10 for blade-related fatigue loads and 4 for tower-related fatigue loads. Only the tower top yaw system Wöhler exponent is not listed; this is set to 7 to be consistent with the tower top torsional surrogate originally implemented in PyWake.

Assumption 3 *The Wöhler exponents for blade root flapwise and edgewise DELs is 10, that of tower top yaw system 7 and that of tower base fore-aft and side-side 4.*

Analytically determining the DEL_u parameters would mean that exact material properties and geometries should be known. FEM software can be used to numerically determine the load, moment or stress limits of the cross sections of interest. Modelling this would be a project on its own and is certainly outside the scope of this thesis. Instead, the curve parameters can be tuned automatically such that they result in realistic failure rates given typical operating conditions and strategies. Wind turbine components are typically designed for a fatigue lifetime that exceeds the desired operational lifetime of the farm. As such, this lifetime can be tuned to be approximately 22 years, assuming the wind farm is to operate for 20 years. Tuning these parameters thus comes down to determining what curve parameters will result in component breakdowns after approximately 22 years. This essentially performs fatigue lifetime 'backwards' without explicitly calculating material or fatigue properties. This will, however, suffice for this environment; after all, these values can be updated with more accurate ones if the environment is ever to be applied to training real-life controllers. This method of inferring fatigue curve parameters is very similar to work by Leonetti et al. (2017) and Allaix and Gijsbers (2016), who used Bayesian inference on real-life observations to find the best fitting curve parameters.

4.2.1. Numerical Fitting of Fatigue Curve Parameters

For fatigue curve fitting, several zero-yaw, 25-year-long simulations can be run in the environment. This represents the 'default' operation of a wind farm and serves as an approximation of the fatigue loads that the turbines will endure during their lifetime. All fatigue loads (e.g. DELs) during this simulation are recorded as an N-dimensional time series of length T, where N is the number of turbines, and T is the number of time steps. Next, an optimisation algorithm can be used that evaluates the lifetimes of all components given a set of curve parameters and tunes them accordingly to achieve the desired 22-year lifetime. For this, a simple gradient descent algorithm can be employed that aims to centre the distribution of lifetimes of all components in the turbine to be around 22 years.

To enable gradient descent to work, an objective function must first be defined. This function must align with the goal of the optimisation problem, being the alignment of the mean component lifetime with the target of 22 years. A simple objective function to use is the Mean Squared Error (MSE), measuring the average of squared residuals. Given a 'guess' of ultimate moment DEL_u, the objective function calculates the time step at which failure occurs for all turbines in the dataset. Next, the difference between these calculated 'failure' time steps (or 'lifetimes') and the desired 22 years of lifetime converted to time steps is squared and averaged over all values. The task at hand for the optimisation algorithm is thus to minimise this MSE given the freedom to adjust DEL_u.

One more thing to consider is that materials (and thus components) might not always be produced perfectly, and some variation in material parameters is thus likely to occur. This could affect the fatigue properties of components, where some components might last longer than others. To effectively model this, the fatigue curve parameter DEL_u can be modelled as a stochastic variable with a normal distribution, where for each component DEL_u is sampled from this distribution. The component maintains this sampled value throughout its lifetime and re-samples upon replacement. In this case of stochastic variables, the objective function must be adjusted. Instead of optimising for a single fixed value of component lifetime, it should instead optimise to match a distribution of lifetimes. From component

lifetime analyses by several authors (El-Naggar et al., 2021; Carroll et al., 2016; Dao et al., 2019), it is evident that the distribution of lifetimes can often be several years around the mean expected value. These studies make no distinction between failures caused by fatigue and failures caused by other factors, such as rain erosion or wear. As such, a reasonable choice to make is to model the distribution as a normal distribution centred around 22 years, with a standard deviation of 1 year. The objective function can then, given the mean and standard deviation of DEL_u, determine component lifetimes by sampling from the parameter distribution and processing the time-series fatigue load data. Since the mean is already known from the deterministic optimisation, only the standard deviation of the normal distribution needs tuning. The objective function can thus be defined as the MSE loss between the target and fitted standard deviation of the resulting lifetimes.

To make iterations a whole lot faster and to prevent caching unnecessary time series data in memory, a trick can be used to capture the entire time-series load history in a single value. Remembering that DELs are inherently already defined as a 'summary' of load cycles, a time series of DEL values itself can further be compressed into a single 'master' DEL that incurs the same damage over the load history. Instead of having to evaluate the entire time series for each gradient descent iteration, only the single master DEL needs evaluation with a number of cycles equal to the original time series. This 'master' DEL, or more formally the lifetime damage equivalent load (LDEL), can be calculated as follows:

$$i \cdot n \left(\frac{LDEL}{DEL_u}\right)^m = \sum_i n \left(\frac{DEL_i}{DEL_u}\right)^m \tag{4.1}$$

$$i \left(\frac{LDEL}{DEL_u}\right)^m = \sum_i \left(\frac{DEL_i}{DEL_u}\right)^m \tag{4.2}$$

$$\frac{LDEL}{DEL_u} = \sqrt[m]{\frac{1}{i}\sum_i \left(\frac{DEL_i}{DEL_u}\right)^m} \tag{4.3}$$

$$LDEL = \sqrt[m]{\frac{1}{i}\sum_i DEL_i^m} \tag{4.4}$$

This means that this lifetime DEL can be calculated without knowing DEL_u and can thus be used to compress the full load history into a single DEL for use in the gradient descent objective function. However, before doing so, it must be validated whether this LDEL can be used as an accurate approximation for the LDEL for any fraction of the time series. This convergence study is done in Appendix B. From this study, the following assumption can be safely made:

Assumption 4 *The full load history of a 25-year-long horizon can be compressed into a single DEL value, called the 'master' DEL or LDEL. Furthermore, this LDEL value is assumed to be equally valid when used to determine lifetimes for horizons shorter than the original 25 years with which it was calculated.*

Now, with this LDEL and a guess from the minimisation algorithm, the expected component lifetime in days n_d can be easily calculated given a guess for DEL_u. Realising that, since the LDEL is defined at 1Hz, the number of cycles per day is $n_d \cdot 60 \cdot 60 \cdot 24 = 86400 n_d$:

$$86400 n_d \left(\frac{LDEL}{DEL_u}\right)^m = 1 \tag{4.5}$$

$$n_d = \frac{1}{86400 \left(\frac{LDEL}{DEL_u}\right)^m} \tag{4.6}$$

The optimisation algorithm employed is Nelder-Mead, as implemented in Scipy's Minimize method. The Nelder-Mead algorithm aims to minimise a function $f(\mathbf{x})$, in this case the objective function described in the previous paragraph, using the variable(s) $\mathbf{x}$ within a given box constraint. Since DEL_u can only be positive, the box constraint is only bounded on the bottom edge, e.g. with a lower limit of 0, and has no upper bound. Similarly, in the case of optimising for the stochastic version, the standard

deviation of DEL_u is also bounded below by 0, as the standard deviation can never be smaller than 0. Using this gradient descent algorithm, it only takes a few seconds to find value(s) for DEL_u (and the standard deviation thereof) that minimises the objective functions. The loss curves of the gradient descent process can be found in Appendix C. The final mean and standard deviation values can be found in Table 4.1. Fitted values for two- and three-year target lifetime standard deviations are also provided.

DEL	Wöhler Exponent m	**Mean** $\mu_{DEL_u}[kNm]$	**1-year STD** $\sigma_{DEL_u}[kNm]$	**2-year STD** $\sigma_{DEL_u}[kNm]$	**3-year STD** $\sigma_{DEL_u}[kNm]$
Blade Root Edgewise	10	35634.785	148.750	310.000	477.163
Blade Root Flapwise	10	55370.247	0.000	421.245	697.891
Tower Top Yaw	7	63581.925	0.000	655.042	1127.499
Tower Base Side-to-Side	4	1962407.473	13526.565	41078.125	63800.537
Tower Base Fore-Aft	4	6163272.492	62470.703	137500.000	205311.889

Table 4.1: Fitted fatigue curve parameters.

One last note to make is that *technically* the tower base fore-aft and side-to-side loads are from the perspective of the rotating nacelle's frame of reference and are thus not fixed in the world's frame of reference. This means that if a world-fixed frame fore-aft and side-to-side damage value D were to be defined, the nacelle-rotating DELs must be projected onto the world frame first before damage can be accumulated. This is also the reason why the side-to-side DEL_u is significantly lower than the fore-aft DEL_u, whereas in reality, the tower base is often symmetric and would have identical values. For simplicity, in this environment, it is assumed that fore-aft and side-to-side Ds exist in the nacelle-rotating reference frame.

Assumption 5 *The tower base fore-aft and side-to-side fatigue damage can be modelled in the nacelle's rotating reference frame, using asymmetrical fatigue properties, without projecting them back into the world-fixed frame of reference.*

4.3. Chapter Recap

This chapter covered the modelling of component degradation due to fatigue. Taking inspiration from the IEA61400-1 standard on wind turbine design, damage accumulation was quantified as Palmgren-Miner's linear accumulation of damages due to load cycles each timestep. The DELs, as calculated using the farm-level surrogate, are used to determine ΔD for each timestep. ΔD is calculated according to fatigue curves, relating the DEL fatigue load to the number of cycles which would lead to failure. Gradient descent is used to determine the parameters that characterise these curves and find the set of parameters that result in 22-year design lifetimes under simulated zero-yaw conditions in a wind farm. The Wöhler exponents are assumed to be fixed for each of the five types of components. Gradient descent then provided the remaining parameter, DEL_u, thereby fully defining the curves. Furthermore, to possibly deal with stochasticity in material parameters, standard deviations were also fitted to obtain normal distributions of the parameter DEL_u to match a distribution in observed lifetimes. To answer the sub-question *How can fatigue loads experienced by turbine components be mapped to accumulated lifetime consumption?*: using Palmgren-Miner's rule and fatigue curves with parameters fitted based on average load histories. With the fatigue accumulation model finalised and the surrogate farm-level model ready, the next chapter can proceed to construct some remaining modules and complete the simulation environment.

5

Simulation Environment

This chapter will cover the further development of the simulation environment that will enable the training of the reinforcement learning controller. At the core of the environment is the simulation model developed in chapter 3; however, it requires many other modules that each model some aspect of the environment. As such, this chapter defines all other modules that collectively make up 'WakeWISE' (Wakesteering Windfarm Interactive Simulation Environment).

5.1. Main setup

The environment is implemented in Python according to the Gymnasium framework (Towers et al., 2023), a popular and well-known framework for building reinforcement learning environments. It implements a standardised way for a reinforcement learning algorithm to interact with the environment by exchanging rewards, actions and observations. This reward function and observation space are covered in section 5.7 and section 5.8 respectively. The action space consists of the yaw angles for each of the n turbines, implemented as a 'Box' space of dimension n with bounds $[-30, +30]$. Since the environment works with timesteps Δt of 10 minutes, the problem is constructed as a discrete control problem with ten-minute intervals. As the environment transitions from timestep t to $t+\Delta t$, the following assumption is made on the transition behaviour:

Assumption 6 *All sampled environment variables, originating from random processes, are assumed to remain constant between t and $t + \Delta T$. The same goes for the yaw angles, which are too kept constant.*

5.2. Time Keeping

Among the main design objectives for the simulation environment is the requirement for it to be as realistic as possible. This means that the modules embedded in it should present behaviour that mimics reality as closely as possible. Many real-life processes can have daily, seasonal or even yearly cycles or dependencies, and modelling such behaviour thus means explicitly considering the evolution of time during the simulation. As such, a timekeeping module will be included to keep track of and progress the simulated time. Other modules can then refer to this module to adapt their behaviour accordingly. The timekeeping module is implemented as a simple counter, keeping track of minutes, hours, days, months and years. Each timestep advances its counters by the configured timestep duration, cycling back to zero where necessary. Finally, each time the environment is reset, it takes a randomly sampled time state as an initial condition; this promotes more diverse trajectories during reinforcement learning training.

5.3. Inflow Conditions

A vital aspect of the simulation environment is the sampling of synthetic inflow conditions, e.g. the free-stream wind state around the wind farm. The wind conditions can considerably impact wake interactions in the farm and are precisely one of the main drivers behind the need for wake steering. For the

environment, a suitable inflow condition sampler should be chosen that can provide the environment with a new state every timestep as the episodes progress. Several options are available for sampling such conditions, but as always the environment design objectives should be satisfied. In this section, several options for wind samplers are compared and contrasted, and a suitable module type is chosen for embedding inside the environment.

5.3.1. Model Choice

A relatively simple but often utilised method of sampling inflow conditions is to choose simple distributions for each variable to be sampled. For each timestep, the module simply samples from each distribution to obtain a new wind state. Weibull distributions for wind speed and uniform distributions for wind direction are often used for this purpose. Though this method of sampling enjoys an extremely fast inference speed, often in the sub-millisecond range, its simplicity causes there to be no temporal coherency in the generated time series. Furthermore, it is unlikely that real-life wind conditions match perfect statistical distributions. On top of that, when univariate distributions are used, there is no coupling between the different variables; this seems unrealistic, as certain wind directions might bring stronger winds. Finally, the lack of seasonal or daily effects brings yet another factor to decrease realisticity as there is no coupling between time and behaviour. However, this can be partly overcome by conditioning the distribution parameters on time.

The aforementioned relatively simple approach can be expanded in the form of a Markov Chain. A Markov Chain models a finite set of states and the transition probabilities between them; in other words, given a state s_i of set with indices $i \in N$, it models the transition probabilities $P(s_j|s_i) \quad \forall \quad j \in N$. This adds an element of temporal coherence to the model, as the probability of transitioning to the next state depends on the previous state. Effectively, this means that when these probabilities are tuned according to real-life data, probable transitions will occur more often, according to reality. Furthermore, the probability of transitioning to different states can depend on time by determining transition matrices for different timeframes. All in all, this brings both temporal coherency and time dependency. Furthermore, joint distributions can be modelled by conditioning state transitions on other variables. Markov Chain models only require some parameter lookups and simple categorical sampling, meaning their runtime is effectively no slower than that of the simple distribution sampling mentioned earlier. Examples of Markov-Chain modelling of synthetic wind time-series can be found in work by Shamshad et al. (2005) and Sahin and Sen (2001). Similarly, Scholz et al. (2013) extended the Markov-Chain framework in the domain of wind turbine power production, modelling transition probabilities as Bernstein polynomials.

Deep learning can also provide ways of generating time-series data. One method of forecasting or generating such data would be to train an MLP based on a training dataset of real-life measured wind conditions. Simple supervised learning would be sufficient to obtain a model that can predict future trends given previous values. However, one major downside to this method is that the model overfits significantly on the provided dataset, learning to replicate the time series it was trained on. The inherent lack of stochasticity and randomness means the model is entirely deterministic, leading to a lack of unique time series outputs. Possible positive aspects, though, can be that its inputs can be augmented with additional variables like time or other wind parameters on which it should be conditioned. Models based on MLP will typically have inference speeds of around 1 to 2 milliseconds.

Generative Adversarial Networks (GANs) or Probabilistic Auto-Regressive (PAR) models can be used to overcome the issue of deterministic outcomes and lack of stochasticity in the generated signals. Both models consist of neural networks that are trained to imitate the time series in a dataset as closely as possible without exactly repeating it. Examples of such models are TimeGAN (Yoon et al., 2019) and PARNN (Panja et al., 2024), both of which can be used to generate synthetic time-series data. Both methods have shown excellent performance in imitating a variety of time series signal datasets whilst retaining the distributional properties of the input signal. However, their impressive performance comes with a runtime in the millisecond range; this typically is not a big deal unless inference speed is of importance, as is the case in this environment.

From these options, summarised in Figure 5.1, it is evident that the runtime becomes infeasible for the environment as soon as deep learning is used as part of the sampling process. Given that the surrogate model itself already runs in the millisecond range, adding an additional significant amount of runtime each step will cause a violation of the first design objective. Despite their accuracy and realisticity in imitating real-life conditions, the deterministic and probabilistic deep learning approaches are unsuited for this environment. Besides, the autoregressivity of signals is better learned by a Markov

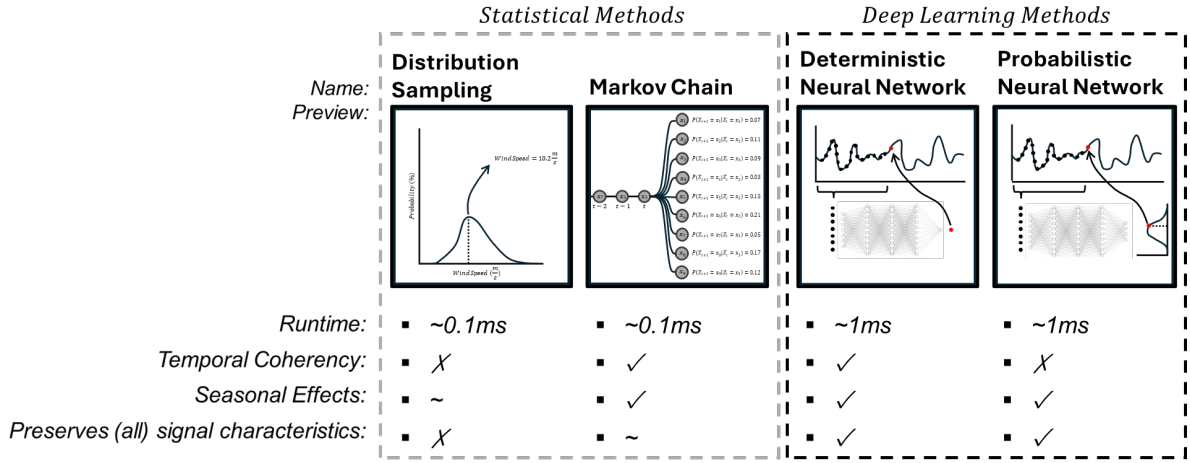

Figure 5.1: Comparison of the four options for inflow sampling.

Chain than through deep learning models. Given the fact that a Markov Chain improves on temporal coherence and time-dependency compared to the simple distribution sampling approach whilst not significantly increasing inference time, it seems like the best approach for the environment.

5.3.2. Model Fitting

The next step is thus fitting Markov Chain parameters to a dataset. For this, a Discrete-Time Markov Chain (DTMC) is used, consisting of a set of finite states $S = x_1, x_2, ..., x_n$ and transition probabilities $P_{ij} = P\{X_{t+1} = x_j | X_t = x_i\}$, where $i, j \in n$ and X_t indicates the state of the system at time t. The Markov property dictates that the probability of moving from state X_t to X_{t+1} is only dependent on the state X_t, and not on any of the states $X_0 ... X_{t-1}$ preceding it. This means that it has a probability matrix P of shape $n \times n$, where $\sum_{j \in S} P_{ij} = 1 \quad \forall i \in n$.

A suitable dataset can be found on the website of Danish wind energy company Ørsted. They offer, free and publicly available, a 2-year dataset of meteorological measurements of their Anholt offshore wind farm (Ørsted). This dataset includes wind speed and direction measurements at a 10-minute temporal resolution. Since the dataset contains measurements at ten different points in the farm, they are first averaged during pre-processing. This leaves, for every measurement time, a single (averaged) value for both the wind speed and wind direction. Additionally, invalid numbers (such as missing values) are replaced by interpolations of the data points around them. This data is now ready to infer the Markov Chain matrix from.

Since the Markov Chain works with discrete states, the wind direction and wind speed must first be digitised into a set of discrete values. The wind direction is discretized into $N_{wd} = 18$ bins of 20 degrees width, with the first bin centred around zero. The resulting discretised wind rose can be found in Figure 5.2. Wind speed is discretized into $N_{ws} = 40$ bins between 0 and 40 m/s and can also be found in Figure 5.2. Furthermore, note the definition of the number of months equals M.

The first tensor to cover is the transition tensor for wind directions. To enable temporal coherency in the synthetic time series, sampled wind directions must depend on previously sampled points. As these transitions are modelled according to a first-order Markov chain, the first dimension of size N_{wd} of the transition tensor is indexed by the previous wind state. Furthermore, to include a dependency on the month of the year for seasonal effects, another dimension of size M must be included along the second axis for indexing by month. The third dimension then becomes the vector of transition probabilities of size N_{wd}, one probability of transitioning to any of the possible wind direction states. In total, this will produce a tensor of shape $N_{wd} \times M \times N_{wd}$. The tensor is populated by iterating over each month, then iterating over each wind state and performing simple frequency counting of transitions between that state and any other state. These frequencies are then divided by the total number of transitions between that state and any other state to obtain the transition probabilities. By doing so, the transition probabilities are both conditioned on the current month of the year as well as the previous wind direction state.

The second tensor to cover is the transition tensor for wind speeds. Temporal coherency herein is

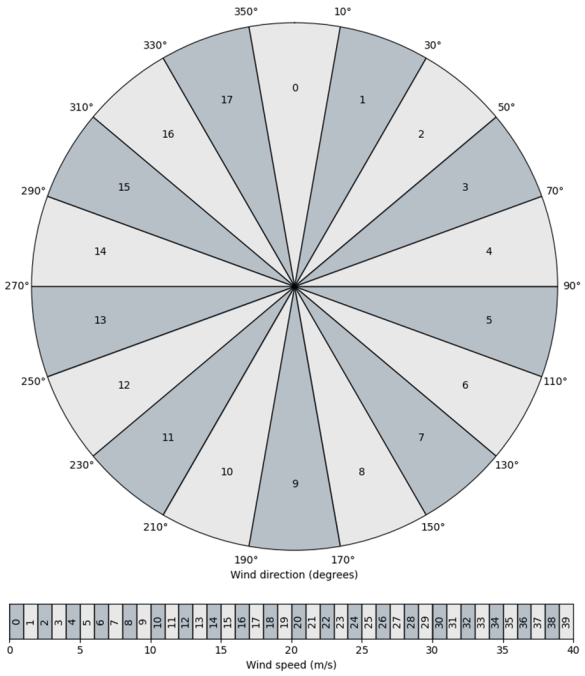

Figure 5.2: Digitisation, or 'binning', of wind direction and speed values.

achieved by the same method as was the case for wind directions, by indexing along the first dimension of size N_{ws} with the wind speed state. The second dimension, of size M, is used for indexing using the wind direction state. This allows the wind speed to depend on the wind direction; this adds to realisticity, as certain wind directions generally carry higher wind speeds. The final dimension of size N_{ws} becomes the vector of transition probabilities, one probability of transitioning to any of the possible wind speed states. In total, this will produce a tensor of shape $N_{ws} \times N_{wd} \times N_{ws}$. Here, too, the tensor is populated by looping over wind directions, then over all wind speed states and performing frequency counting of transitions between states. The probabilities are obtained by dividing by the total number of transitions. The result of both Markov-Chain inference steps are two matrices, shaped $N_{wd} \times M \times N_{wd}$ and $N_{ws} \times N_{wd} \times N_{ws}$, for determining transition probabilities between wind direction and wind speed states respectively.

To now start generating synthetic time series wind direction and speed, the sequences must first be primed using initial states. These states can be chosen uniformly among all possible wind direction and wind speed states. Next, the wind direction transition tensor can be indexed along its first and second dimensions using the previous wind direction state and current month of the year, respectively. This yields a $1 \times N_{wd}$ vector of transition probabilities. Using categorical sampling with probabilities, the next wind direction state can then be sampled quickly. For the wind speed sampling, a similar process applies; here, the first two dimensions of the transition tensor are indexed by previous wind speed and current (newly) sampled wind direction. The result is two discrete states for both variables; this is, however, a very non-smooth output. To tackle this, uniform noise between the boundaries of $[-10, +10]$ and $[-0.5, +0.5]$ can be added to wind direction and wind speed states, respectively. This process then repeats for as long as necessary to fill the synthetic time series.

5.3.3. Model Evaluation

In Figure 5.3 and Figure 5.4, a randomly generated sample of wind conditions is compared with a sample of respective wind direction and wind speed measurements in the dataset. Note that the goal is not to replicate the original signal but rather to imitate it. The synthetic signal seems to behave very similarly to the measurements in the dataset and exhibits similar behaviour over time. Also note that the jumps in wind direction, which can be seen in Figure 5.3, result from the continuity of the discretized wind direction bins, which loop around from 0 to 17 when the wind crosses 360 degrees. Though these discontinuities in the plot might seem unrealistic, they are perfectly normal when thought of in a radial

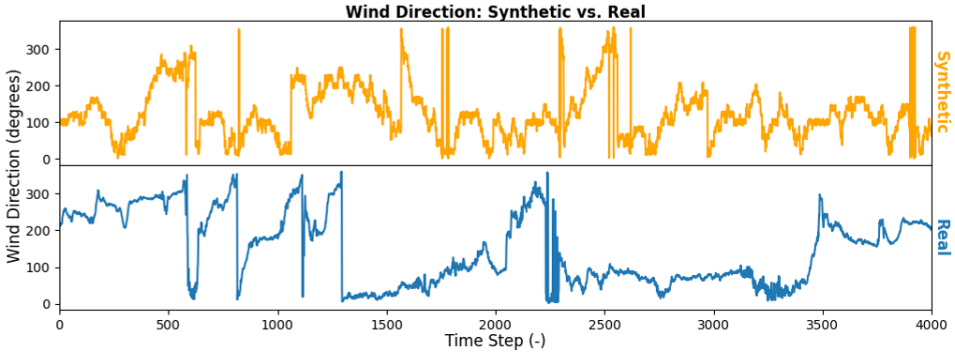

Figure 5.3: A sample of synthetic wind direction data versus the dataset.

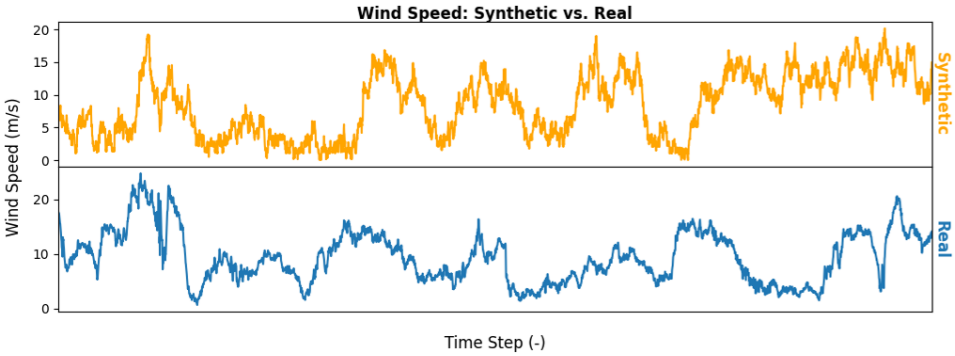

Figure 5.4: A sample of synthetic wind speed data versus the dataset.

sense. As is evident from these plots, the Markov Chain seems to produce realistic time series that exhibit realistic behaviour. Furthermore, the generation of these 4000 timesteps took 60 milliseconds, resulting in a per-timestep latency of only 15 microseconds.

More qualitative and quantitative analysis can be done to analyse the module's performance further and assess its accuracy compared to the input dataset. Figure 5.5 and Figure 5.6 plot the probability density plots of wind speed versus wind direction and wind direction only, respectively. As is evident from these figures, the distributions of different wind conditions show a good match with the actual measured data. Furthermore, the probabilities of each wind direction occurring match very well. Since the wind direction effectively loops around back to 0 when it exceeds 360, Figure 5.7 and Figure 5.8 cast aforementioned figures into a polar projection. Here, too, it is evident that distributions match well, or at least well enough, for applications where approximations are sufficient, such as this environment.

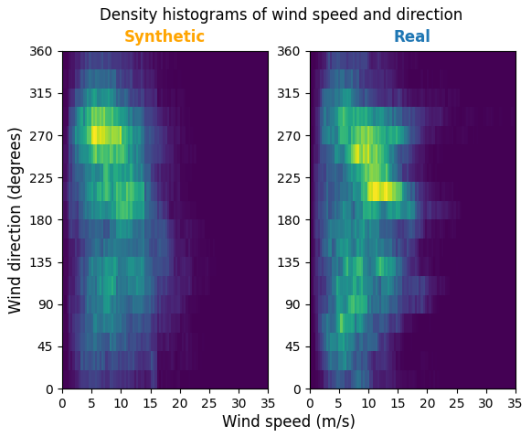

Figure 5.5: A sample synthetic wind signal's 2D density plot versus that of the real signal.

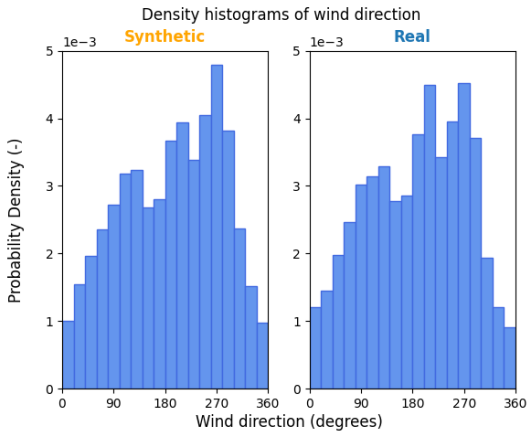

Figure 5.6: A sample synthetic wind signal's wind direction density plot versus that of the real signal.

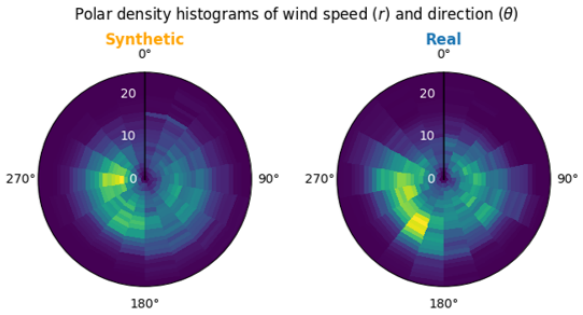

Figure 5.7: A sample synthetic wind signal's polar 2D density plot versus that of the real signal.

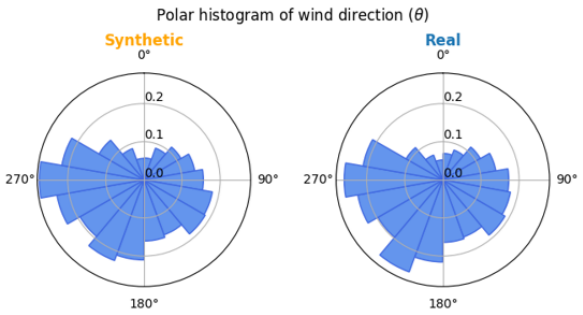

Figure 5.8: A sample synthetic wind signal's polar wind direction density plot versus that of the real signal.

Some more quantitative analysis can be done by using the specific values of each of the time signals. In Figure 5.9 and Figure 5.10, the auto-correlation and Power Spectral Density (PSD) of a synthetically generated signal and the real signal are shown. Overall, they show good agreement, indicating that the synthetically generated signal is reasonably realistic. The real signal tends to have a few deviating auto-correlation lag periods compared to the synthetic signal, but generally, they follow a similar shape. Regarding the PSD, there seems to be a larger discrepancy between the two signals. For lower frequencies, the signals match more closely; this can be attributed to the Markov chain process. The higher frequencies - which seemingly show more discrepancy - could result from the uniform noise added to the discretized states to obtain a smoother signal. This uniform noise likely has a higher noise frequency, as the random samples have no temporal coherency. All in all, for use in the environment, the wind direction and wind speed sampling module using the Markov Chain process has been shown to be a suitable and feasible choice.

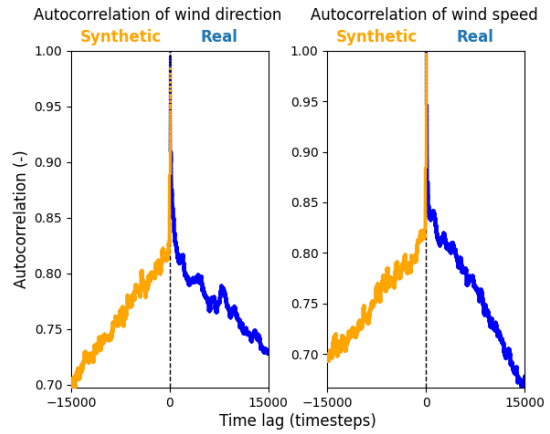

Figure 5.9: Autocorrelation plot of a sample synthetic signal versus that of a real signal.

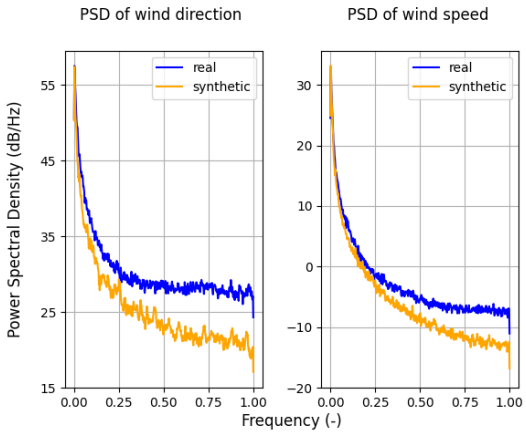

Figure 5.10: Power Spectral Density (PSD) plot of a sample synthetic signal versus that of a real signal.

Two more variables need sampling to complete the inflow sampling module: turbulence intensity and shear exponent. For the turbulence intensity, the method as outlined by the IEC 61400-1 standard on wind turbine design can be consulted. This was discussed earlier in subsubsection 3.3.1.1 and involves sampling from a uniform distribution between a lower and wind-speed dependent upper bound. Similarly, the shear exponent sampling derived from Dimitrov (2019), also discussed in subsubsection 3.3.1.1, can be reused. The free-stream wind conditions can be fully characterised using the Markov Chain process for wind direction and speed, combined with the aforementioned turbulence intensity and shear exponent sampling. Furthermore, the wind direction and speed are time-dependent, temporally coherent and have fast inference speeds, thereby satisfying all three environment design objectives.

5.4. Electricity Price

Another parameter that can be an essential input for wake steering policies is the current price at which electricity is sold on the market. A higher electricity price might justify a more extreme yawing policy, as the net gain in profit could significantly exceed the increased damage incurred as a result. The prices might have a significant impact on what policy to follow and are thus not to be neglected in the environment. This means that a sufficient module that models these prices is to be chosen. In this section, options for this module are compared and contrasted, and an informed decision is made based on the environment design objectives.

5.4.1. Model Choice

In the simplest case, a fixed value is chosen to model the electricity price. A time-based average or an arbitrary value that seems most realistic can be picked. Modelling electricity prices this way incurs a negligible increase in runtime and is undoubtedly the fastest way to implement such a module. However, using such a fixed value removes the trained agents' ability to deal with fluctuating electricity prices entirely, significantly impacting their ability to be used in practice. Instead, to add a bit more complexity, a probability distribution can be fitted on historical data of electricity prices. At execution time, samples can be drawn from this distribution at random. This allows the agent to encounter a broader range of electricity prices and thus be more robust against possible fluctuations it might encounter in practice. Sampling from simple distributions incurs a negligible time loss compared to the fixed value case, so that is not a problem. However, in both aforementioned cases, one relation is still not explicitly considered: the dependence on time. Oftentimes, the electricity price is heavily coupled with the time of day or seasons of the year. For example, in the summer, electricity demand might be lower, leading to an abundance in supply and, thus, lower overall prices. To address this, fixed values can be fitted based on both the time of day and month of the year. Sampling then comes down to finding the value corresponding to a given time in a large lookup table, indexed by hour and month. Once again, the increase in runtime is negligible, but it is much more realistic. All three options are summarised in Figure 5.11.

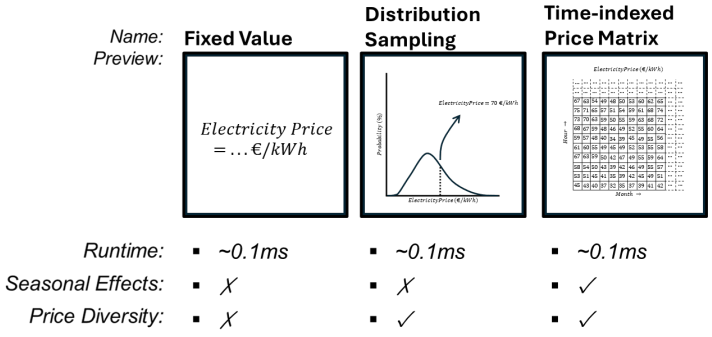

Figure 5.11: Comparison of the three options for electricity price sampling.

Given the design objectives for the environment, the third option seems to be the best fit. All three options have almost identical runtimes, being in the order of microseconds, meaning runtime plays no role in this decision. When it comes to accuracy, all three can be fitted based on real-life data with high accuracy. However, only the third one mimics realistic behaviour when it comes to the objective of being realistic. The first two options ignore seasonal or hourly effects, thereby lacking some coherent and time-dependent behaviour, making them somewhat unrealistic. The most suitable option is, therefore, the third option.

5.4.2. Model Fitting

The next step is to find a suitable dataset of electricity prices on which the model can be fitted. For this, the day-ahead prices of the Scandinavian energy market can be used; this data comes from the Nordpool spot market, which is a joint market between Norway, Sweden, Finland, Denmark and Germany. It is freely and openly available on Nord Pool Group and Energinet. From here, historical hourly data from the past years can be downloaded. Since 2020-2021 was heavily influenced by the

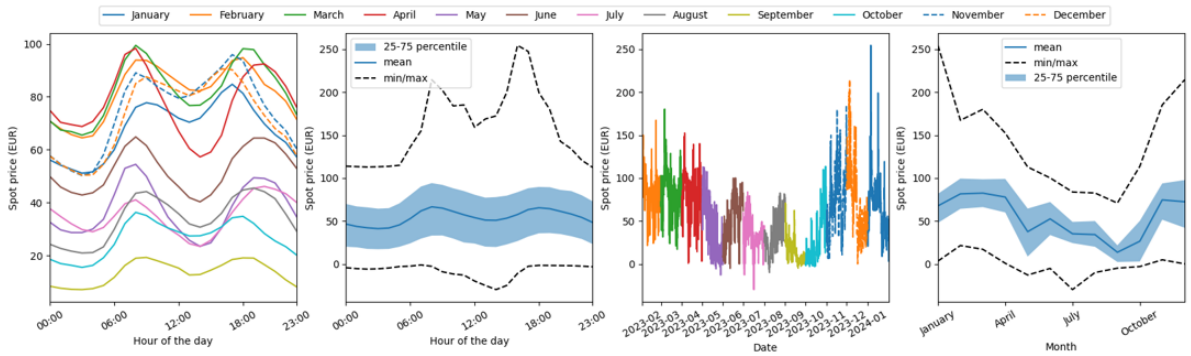

Figure 5.12: Dataset analysis of spot prices.

COVID-19 Pandemic and the period 2022-2023 by the conflict in Ukraine, only the spot prices for 2023-2024 were downloaded. Furthermore, the 'SYSTEM' price category is of interest, as it serves as a (stable) reference price for the whole market system.

To correctly model seasonal and hourly behaviour, prices in the dataset must be grouped by hour and month. This yields $12 \cdot 24 = 288$ sets of data points characterising the electricity prices at each hour and month. For each of these groups, the mean is calculated and stored in a 12×24 matrix. Figure 5.12 shows an analysis of the time-dependency. From this analysis, it is evident that both seasonal effects and hourly effects play an essential role in the determination of energy prices. In this data, it is, for example, visible that in winter, prices tend to be significantly higher and that in the morning and evening hours, the peak energy demand is likely to increase prices as well.

To proceed, the electricity price matrix must be fitted on this data. For this, the data as shown in the leftmost plot of Figure 5.12 is used. This data is averaged hourly for each month of the year. This is sufficient to construct the 12×24 matrix of average electricity prices, indexing the price by month and hour along the first and second dimensions, respectively.

5.5. Maintenance Costs

Maintenance costs form an essential part of the reward with which the performance of a wake steering policy can be assessed. In the environment, this cost is related to wind turbine components' degradation and breaking down and is therefore linked to maintenance actions. It is thus essential to define a model with which these maintenance costs are calculated given the (changes in) different states of the environment.

In essence, there are two ways of modelling these maintenance costs in the environment: using sparse costs and using dense costs. In the sparse case, maintenance costs are incurred exactly when components break down or require maintenance. This would, in fact, be the most realistic way of modelling it, as maintenance is generally paid for when it is necessary. Dense costs, on the other hand, spread these costs out over the component's lifetime. In the end, both models amount to the same total cost; they differ only in at which point(s) in time these costs are incurred. Numerical examples of both are shown in Figure 5.13 and Figure 5.14.

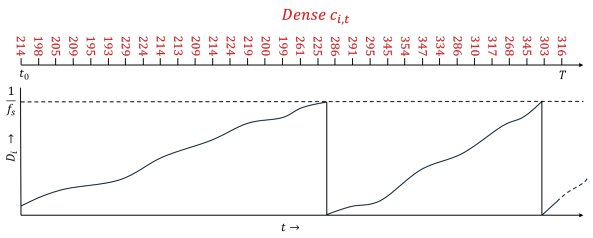

Figure 5.13: Dense maintenance costs

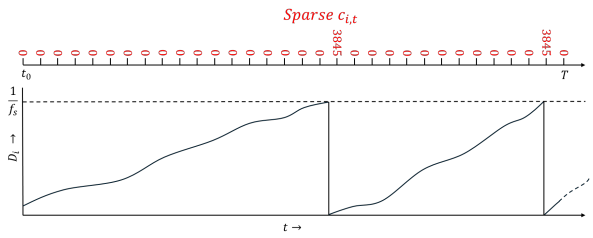

Figure 5.14: Sparse maintenance costs

For the dense cost case, the change in accumulated fatigue damage D, called ΔD, is used. ΔD directly indicates the 'consumption' of fatigue lifetime of a component in the current timestep and is,

therefore, an ideal way of measuring the continuous and momentary usage of a component. Since the component is to break down when a failure limit D_f is reached, which without a safety factor defaults to 1, all maintenance costs for replacement must be incurred over its lifetime. The costs incurred can thus be simply and directly derived from ΔD multiplied by the replacement cost.

$$c_i^t = \Delta D_i^t C_i \tag{5.1}$$

Where c_i^t is the dense maintenance cost for component i at time t, ΔD_i^t the change in accumulated fatigue damage of component i at time t and C_i the total cost of replacement for component i. If a safety factor is applied to the component's failure limit, e.g. instead of 1 its failure limit is $\frac{1}{f_s}$, the dense cost formula can be easily adjusted:

$$c_i^t = \Delta D_{i,t} C_i f_s \tag{5.2}$$

The sparse maintenance cost case works differently and is in fact much simpler. Instead of looking at changes in D_i, the actual damage state D_i itself dictates the cost. In all timesteps where D_i does not exceed the failure limit, c_i^t is zero; in the timestep that does exceed this limit, c_i^t is the full replacement cost C_i. In fact, this can be thought of as a subset of dense costs, where the function is essentially a Dirac-Delta function at 1 (or $\frac{1}{f_s}$ in the case a safety factor is applied):

$$c_i^t = \delta(D_i - \frac{1}{f_s})C_i \tag{5.3}$$

This formulation assumes that the damage state D_i will at some point directly match $\frac{1}{f_s}$ for the Dirac-Delta function to work correctly. In practice, however, D_i will likely jump over this value between timesteps. The formulation in code is, therefore, rather an exceedance check every timestep.

The actual maintenance costs C_i remain to be defined. Fortunately, the IEA37 3.4MW reference turbine repository (IEA Wind Task 37) can once again be consulted. In the Excel file `xlsxIEAonshore.xlsx`, the authors of the turbine design provide the computed costs for each of the components in the turbine. From here, costs for the replacement of blades, tower and yaw system can be taken. One additional aspect to consider is the downtime of a turbine as a result of maintenance; to ensure realistic maintenance behaviour in the environment, this too should be modelled. Unfortunately, this downtime is unavailable in any of the IEA37 3.4MW documents. Instead, inspiration can be taken from wind turbine reliability studies by Dao et al. (2019) and Carroll et al. (2016) and supplementing their findings with additional downtime for unaccounted aspects, such as travel and lead times. The final values are shown in Table 5.1.

DEL	Component	Cost (Euros)	Downtime (Hours)
Blade Root Edgewise Blade Root Flapwise	Blade	120901.57	300
Tower Top Yaw	Yaw System	125013.75	150
Tower Base Side-to-Side Tower Base Fore-Aft	Tower	829755.28	600

Table 5.1: Costs and downtime for replacement of each component.

Both versions, dense and sparse, have their upsides and downsides. Sparse costs, for example, are the more realistic case. Dense costs, on the other hand, provide much more information each timestep, which can benefit reinforcement learning. This begs the question of whether there is a version of modelling maintenance costs that sits somewhere between dense and sparse costs. A necessary property for such a cost distribution must at least be that the total cost over the entire component lifetime must be the total replacement cost. For this, it is easiest to relate the change in damage ΔD to some cost c_i^t for that timestep, similar to what was done for the dense cost case. Each ΔD shall then be used to calculate a cost factor f_i^t which is multiplied with C_i to obtain c_i^t. To start, a generic exponential function can be picked with a variable that can be adjusted to adjust the distribution of costs over the component lifetime. The following sections discuss two versions of these unevenly distributed cost functions.

In the first case, the simple formula $e^{cD} - p$ can be used. Here, c is an adjustable parameter, and p is a fixed parameter dependent on c. To ensure the area under the curve between 0 and 1 (e.g. the entire component lifetime) equals 1 (e.g. total replacement cost), the correct value of p must be calculated. To do so, the general integral must be calculated and set to 1:

$$\int_0^1 e^{cD} - p = 1 \tag{5.4}$$

$$\left[\frac{1}{c}e^{cD} - pD\right]_0^1 = 1 \tag{5.5}$$

$$\left(\frac{1}{c}e^{c} - p\right) - \left(\frac{1}{c}\right) = 1 \tag{5.6}$$

$$e^{c} - cp - 1 = c \tag{5.7}$$

$$e^{c} - c - 1 = cp \tag{5.8}$$

$$\frac{e^{c} - c - 1}{c} = p \tag{5.9}$$

However, this only holds true if $c <= 1.25$, as beyond that, the function will intersect with the D-axis (x-axis) within the region of $[0, 1]$. In that case, the intersection of $e^{cD} - p$ with the D-axis must first be determined in order to find the lower bound for the integral:

$$e^{cD} - p = 0 \tag{5.10}$$

$$e^{cD} = p \tag{5.11}$$

$$cD = \ln p \tag{5.12}$$

$$D = \frac{\ln p}{c} \tag{5.13}$$

Repeating the general integral function and equating it to 1, similar to what was done before, yields:

$$\int_{\frac{\ln p}{c}}^1 e^{cD} - p = 1 \tag{5.14}$$

$$\left[\frac{1}{c}e^{cD} - pD\right]_{\frac{\ln p}{c}}^1 = 1 \tag{5.15}$$

$$\left(\frac{1}{c}e^{c} - p\right) - \left(\frac{1}{c}e^{\frac{c\ln p}{c}} - p\frac{\ln p}{c}\right) = 1 \tag{5.16}$$

$$\frac{1}{c}e^{c} - p - \frac{1}{c}e^{\ln p} + p\frac{\ln p}{c} = 1 \tag{5.17}$$

$$\frac{1}{c}e^{c} - p - \frac{1}{c}p + p\frac{\ln p}{c} = 1 \tag{5.18}$$

$$e^{c} - cp - p + p\ln p = c \tag{5.19}$$

$$e^{c} - c = cp + p - p\ln p \tag{5.20}$$

$$e^{c} - c = (c+1)p - p\ln p \tag{5.21}$$

$$e^{W\left[\frac{c-e^{c}}{e^{c+1}}\right]} + c + 1 = p \tag{5.22}$$

Where $W[]$ is the real part of the Lambert-W function. Now that p is defined for both regions $c <= 1.25$ and $c > 1.25$ given a specified value for exponent c, the cost factor f_i^t can be determined. This factor is nothing less than the integral of the curve between D_i^{t-1} (the fatigue damage at the previous timestep) and D_i^t (the fatigue damage at the current timestep), where $D_{i,t} - D_i^{t-1} = \Delta D_t$. By adjusting the exponent value c, both dense costs ($c \to 0$) and sparse costs ($c \to \infty$) can be achieved, as well as anything in between ($c \in [0, \infty]$). Four examples of the function with different parameters for c are shown in Figure 5.15.

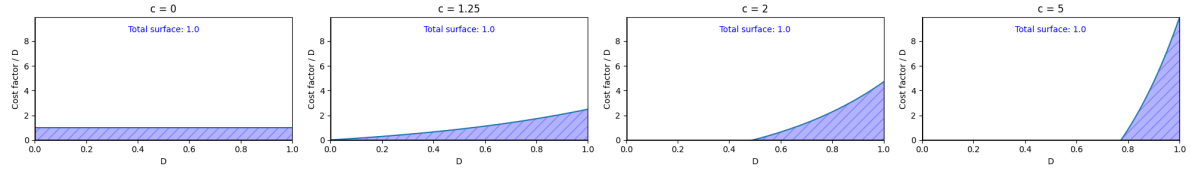

Figure 5.15: Examples of the non-linear cost function $e^{cD} - p$ with different parameters for c.

A second way of modelling these unevenly distributed costs is to employ a different version of the formula, being $e^{cD} - 1$. Here, it is not the parameter p that needs adjusting to ensure the area under the curve remains 1. Instead, the function is left as-is, and the calculated cost factors f_i^t are simply scaled by the true area under the curve. To do so, first, this 'true' area under the curve must be calculated:

$$\int_0^1 e^{cD} - 1 = \tag{5.23}$$

$$\left[\frac{1}{c}e^{cD} - D\right]_0^1 = \tag{5.24}$$

$$\left(\frac{1}{c}e^{c} - 1\right) - \left(\frac{1}{c}\right) = \tag{5.25}$$

$$e^{c} - c - 1 = A \tag{5.26}$$

$$\tag{5.27}$$

When calculating the area under the curve $e^{cD} - 1$ for determining the cost factor f_i^t, all that remains is to divide the obtained area by the 'true' area A. By doing so, the calculated cost factors are scaled such that they sum up to 1, as they should. Here, too, by adjusting the exponent value c, both dense costs ($c \to 0$) and sparse costs ($c \to \infty$) can be achieved, as well as anything in between ($c \in [0, \infty]$). Four examples of the function with different parameters for c are shown in Figure 5.16.

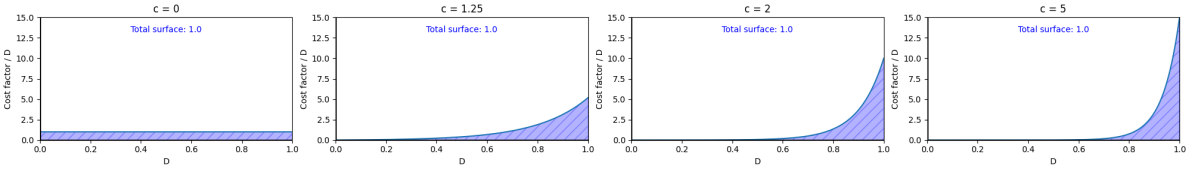

Figure 5.16: Examples of the non-linear cost function $e^{cD} - 1$ with different parameters for c.

5.6. Fatigue Accumulation

The fatigue damage calculation is done according to the methods dictated by the IEA61400-1 standard, e.g. using Palmgren-Miner's linear damage accumulation rule. This was discussed in chapter 4; here, too, the curve parameters, m and DEL_u, defining the fatigue curves, were determined. In this section, the calculation rules for determining fatigue damage and accumulation are briefly and explicitly repeated for completeness.

The surrogate in the environment infers, given inflow conditions and yaw angles, the five DEL loads for each of the components j for each of the turbines i. This DEL is defined at 1Hz; together with a timestep of 10 minutes, this means 600 cycles of the given DEL load are added as fatigue damage. Each timestep t, for each component j in each turbine i, the following calculations are done:

$$N(DEL_{i,j}^t) = \frac{1}{\left(\frac{DEL_{i,j}^t}{DEL_{u_j}}\right)^{m_j}} \tag{5.28}$$

$$\Delta D_{i,j}^t = \frac{600}{N(DEL_{i,j}^t)} \tag{5.29}$$

$$D_{i,j}^t = D_{i,j}^{t-1} + \Delta D_{i,j}^t \tag{5.30}$$

Furthermore, when a component is in the process of being replaced, no damage is accumulated for said component. Note that in the case stochastic variables are used for DEL_{u_j}, it is sampled from a distribution at the beginning of each new replacement for component j.

5.7. Reward Function

The reward function encapsulates both the money earned through energy production as well as the costs incurred due to maintenance. The maintenance costs $c_{i,j}^t$ at each timestep t for component j in turbine i were already defined in section 5.5. What's left is to define the money earned through energy production each timestep.

The GNN surrogate provides for each timestep t and each turbine i the power produced, P_i^t. However, to calculate the profit due to energy production, the energy yield in MWh needs to be determined; after all, the electricity prices are defined as units per MWh. To determine the amount of energy that turbine i has produced in timestep t, its power must be multiplied by 1000 to convert it to W and further multiplied by 600 (number of seconds each timestep). Finally, it must be divided by $3.6e9$ to obtain the energy yield in MWh. This can be simplified to the following formula:

$$E_i^t = \frac{P_i^t}{6.0e3} \tag{5.31}$$

To then determine the profit due to energy production, it can simply be multiplied by the electricity price per MWh at timestep t, R^t. This yields, for all turbines i at timestep t, the profit p_i^t. Given the profit per turbine p_i^t and the maintenance cost per component, per turbine $c_{i,j}^t$, the reward function then becomes:

$$r_t = \sum_i \left[p_i^t - \sum_j c_{i,j}^t \right] \tag{5.32}$$

5.8. Observation Space

The observation space dictates what variables the agent observes and can be used for decision-making. The state s_i^t is defined for each turbine $i \in 0, 1, ..., N-1$ at each timestep $t \in 0, 1, ..., T-1$. For reinforcement learning, it is essential to provide enough information in the observation for the agent to make well-informed decisions. Since maintenance is explicitly considered in the environment, the damage states $D_{i,j}^t$ of component j are part of the observation. Note that perfect knowledge of these damage states would be unrealistic; this will be discussed in chapter 8. Furthermore, the free-stream wind conditions (wind speed V_m^t, wind direction ϕ^t, turbulence intensity I^t and shear exponent α^t) are required to infer the wake effects present in the farm. Additionally, the price of electricity R^t is important, as high prices could justify more risky yawing strategies. Finally, since the end goal is to model the environment with a finite horizon, the fraction η^t of the time horizon that passed is also important. This means that there are 11 variables for each turbine, yielding a total observation of size $11 \times N$. A single vector s_i^t of that matrix, e.g. for a single turbine, is as follows:

$$s_i^t = \begin{bmatrix} V_m^t \\ \phi^t \\ I^t \\ \alpha^t \\ R^t \\ \eta^t \\ D_{i,br,fw}^t \\ D_{i,br,ew}^t \\ D_{i,tt,yaw}^t \\ D_{i,tb,ss}^t \\ D_{i,tb,fa}^t \end{bmatrix} \tag{5.33}$$

Since the wind farm can inherently be represented as a graph, such as the case for the farm-level GNN surrogate, it also makes sense to include a graph-like observation of the environment. Especially when wake effects are present, the topology of the farm has a significant impact on how wake effects propagate through the farm and how individual turbines influence each other. It is thus very logical that this geometrical information might benefit the agent's decision-making process. This opens up the possibility of using GNN-based agents in reinforcement learning. Constructing this graph-like observation actually allows many of the features of the surrogate's graph to be reused. To start with the graph observation, the farm-level surrogate's nodes, edges (connectivity) and edge features can be directly copied. This yields a fully connected graph with edge features already present. All that's left is to encode the observation space defined earlier to the nodes; this is simply done by extracting each turbine's state vector and adding it as a feature vector for the respective node on the graph. Similar to the input embedding in Figure 3.10 used in the farm-level surrogate, the observation embedding then becomes as shown in Figure 5.17.

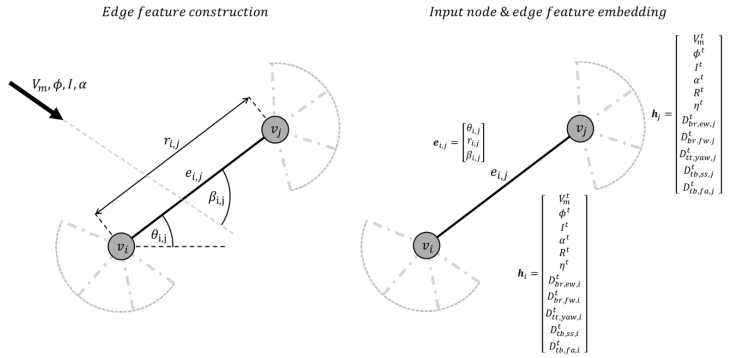

Figure 5.17: Embedding of observation of the environment on a graph, reusing edge features and connectivity from the farm-level surrogate.

5.9. Chapter Recap

This chapter covered the construction of modules that will work alongside the farm-level surrogate model to form the wind farm simulation environment. Suitable techniques and datasets were chosen to obtain models for the sampling of wind conditions and electricity prices in sections 5.3 and 5.4, respectively. Furthermore, a maintenance model was constructed that maps the damage states of components, or increments thereof, to maintenance costs. The accumulation of fatigue damage was formalised given the fatigue theory covered in the previous chapter. Finally, the defining of the reward function and observation space concluded this chapter. To answer the sub-question *How can the environment be constructed in which the reinforcement learning agent is trained, to accurately represent a realistic wind farm?*: through Markov-Chains for wind sampling, time-based values for electricity prices, fatigue accumulation with Palmgren-Miner's rule and maintenance costs based on damage. With the environment now done and ready, the next chapter can proceed with the second stage of this thesis: the training of controllers using reinforcement learning within the environment.

6

Multi-Agent Reinforcement Learning

Now that the environment is constructed, this chapter will proceed towards the next stage of this thesis. First, the control problem is formulated in section 6.1. Next, section 6.2 covers the selection of a suitable RL algorithm and corresponding model architecture. section 6.3 then covers baseline removal, an essential technique used to ensure stable learning of the agents. Finally, section 6.4 highlights some other techniques used in the training process and provides an ablation study to show how each of the applied techniques, including baseline removal, contributes to observed performance.

6.1. Control Problem Formulation

The reinforcement learning algorithm is concerned with the optimisation of the cumulative reward R over each episode. Based on the state s_t of the environment at timestep t, the agent takes an action a_t containing the yaw angles for all turbines in the environment, according to a policy π. It receives a reward r_t, based on a combination of costs c_t and profits p_t:

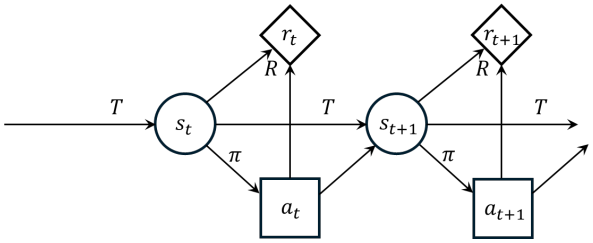

Figure 6.1: Diagram of the Markov Decision Process

The agent must learn to find the (near) optimal set of parameters θ for π, such that the policy π^*_θ maximises the expected cumulative reward R over each episode of length T. The discount factor γ discounts future rewards.

$$\pi^* = \arg\max_{\pi} E\left[\sum_{t=0}^{T} \gamma^t r_t\right] \tag{6.1}$$

The RL problem is therefore defined as the tuple $\langle S, A, R, P, \gamma \rangle$ with the state-space S, action space A, reward function R, transition function P implemented by the environment and discount factor γ.

6.2. Algorithm Selection

When it comes to reinforcement learning, there are many frameworks and algorithms from which to choose. Given the continuous nature of both inputs and outputs, an algorithm capable of inferring continuous actions must be chosen. Proximal Policy Optimisation (PPO) has historically proven to be a very capable algorithm that can deal with continuous action spaces (Mahmood et al., 2018; De La Fuente

and Guerra, 2024); furthermore, it is one of the most widely used algorithms for RL optimisation. Given the multi-agent nature of the problem, the next major decision to make is the training and execution paradigm. Multi-agent reinforcement learning allows both centralised and decentralised training and/or execution, fundamentally differing in how the agent(s) are designed. These training paradigms are explained in section 2.3. In general, the argument against using a CTCE paradigm is the algorithm's scalability with regard to the growing joint action space of all the agents combined. Furthermore, sometimes it can be impractical in reality to perform the decision-making through a central controller, as would be the case with CTCE. Instead, many works propose using the CTDE paradigm, moving execution to the individual agents whilst retaining the power of centralised learning. In this environment, however, a reasonable assumption is that centralised execution would not be a constraint as wind farms are already generally centrally controlled. Theoretically, CTCE should lead to higher performance, given the fact that information is perfectly shared through the network. As such, the CTCE paradigm will be used initially on a smaller farm with a reasonable amount of turbines, and scalability shall be judged later on to determine whether CTDE-based agents are required to scale it to larger farms.

PPO uses two distinct architecture parts: a policy network and a value network. The policy network transforms the state observation into an action, essentially forming the decision-making part of the algorithm. A policy clipping function is added to ensure that the policy network's subsequent updates will not result in wildly varying policy behaviours. As the name suggests, updates to the policy are kept within some proximity of the previous behaviour to ensure more stable updates. The value network, on the other hand, is used in the advantage function; the latter attempts to answer the question of whether taking one action is better than taking another. The advantage function is calculated based on subtracting a *baseline estimate* V from the *discounted sum of rewards* Q: V is the output of the value function, whereas Q follows from experience collected by the agent. The value function V is essentially the expected sum of rewards when starting at the current state s_t. An advantage value greater than zero is indicative of the action resulting in an expected higher cumulative return. The PPO architecture thus consists of two networks that need to be constructed: the policy network and the value network. subsection 6.2.2 will cover the design of the agent's deep learning architecture, which includes this policy and value network.

6.2.1. Agents

In this thesis, five different agents (or 'policies') will be constructed and compared with each other. Three of these will be trained using Reinforcement Learning, whereas the other two are simple heuristic policies. The following five agents will be constructed and compared:

1. **The zero-yaw (baseline) agent** - The zero-yaw agent essentially has a policy of $\pi(s_t) = 0$, thereby always giving a yaw offset of zero to all turbines. It can be considered the 'do-nothing' policy and serves as an example of what most wind farms nowadays will use.
2. **The random agent** - The random agent chooses a random yaw angle in the range $[-30, +30]$ for each of the turbines at each timestep. It serves as a simple reference of what random actions would mean for the farm's performance.
3. **The greedy agent** - The greedy agent optimises for power only; the 'greedy' part comes from the fact that it ignores component degradation due to loads and, therefore, only greedily optimises power. It effectively removes the cost c^t from the reward function defined in section 5.7. The majority of papers covered in section 2.4 used this 'greedy' technique of power optimisation.
4. **The risk-averse agent** - The risk-averse agent does not think about power at all and only considers component degradation. It attempts to minimise the degradation of components at all costs to ensure longer fatigue lifetimes. It removes the energy production profit p^t from the reward function defined in section 5.7 and uses the standard evenly distributed dense costs from section 5.5 to model degradation.
5. **The informed agent** - The informed agent optimises for power but also includes the cost of degradation in the reward. This is the default case highlighted in section 5.7, where both profits through energy production and costs due to fatigue degradation are included. It uses the evenly distributed dense costs from section 5.5. It is an adaptation of the works in the last section of section 2.4, replacing the arbitrary weights for power and loads with both terms expressed in monetary value as outlined in section 5.7. This agent effectively optimises long-term wind farm profit by considering all factors involved.

6.2.2. Agent Architecture

Architectures typically employed in RL are based on conventional neural networks or built on transformer-based models. However, given the importance of farm topology and its relation to the incoming wind, a GNN-based architecture makes much sense in this problem. Recently, the usage of such a graph-based architecture was demonstrated by Sheehan et al. (2024), indicating initial promising results. By explicitly encoding farm topology in the input of the architecture, it no longer needs to implicitly learn these relations as part of its training process. This can essentially boost training efficiency and shift the 'focus' of learning more to directly optimising actions. In section 5.8, a graph variant of the observation was already defined and derived, meaning it can directly be incorporated into a possible GNN inside the agent. The question then becomes how node and edge features are encoded, how graph convolution takes place and how actions are inferred from the output embedding of the graph.

Taking inspiration from the GNN surrogate developed in chapter 3, an encode-process-decode paradigm can be used. The problem at hand is very similar to the physical variable inference problem, requiring implicit modelling of the wake interactions between the turbines given the inflow conditions. As such, node and edge features are encoded on the graph using a 2-layer MLP encoder. The same GNN-type block, GEN, is used for graph convolution to process and propagate information over the graph. This creates a new graph embedding, yielding a new latent vector for each node on the graph. This combined encoding and processing step can be thought of as a state encoder and can potentially be shared between the value function and policy. The final part of the architecture is the transformation of the obtained hidden encoded state into the required outputs, being the parameters characterising the action distributions and the state-value function output.

To obtain the vector of actions, each node's latent vector is fed through a 2-layer MLP decoder, each outputting a mean and standard deviation of a normal distribution. During training, actions are sampled from these distributions to facilitate exploration; during inference, solely the mean is taken. All actions are then stacked to obtain the final action vector. For the value function, node pooling is used to obtain a single-vector graph representation by taking the mean along the node dimension. This is fed through a 2-layer MLP decoder to obtain a single-value function output. The former is called the policy head of the architecture, and the latter is the value head. The architecture is shown in Figure 6.2. Note that the first two steps, node & edge encoding and message passing, can be shared between the policy head and value head as a shared state encoder.

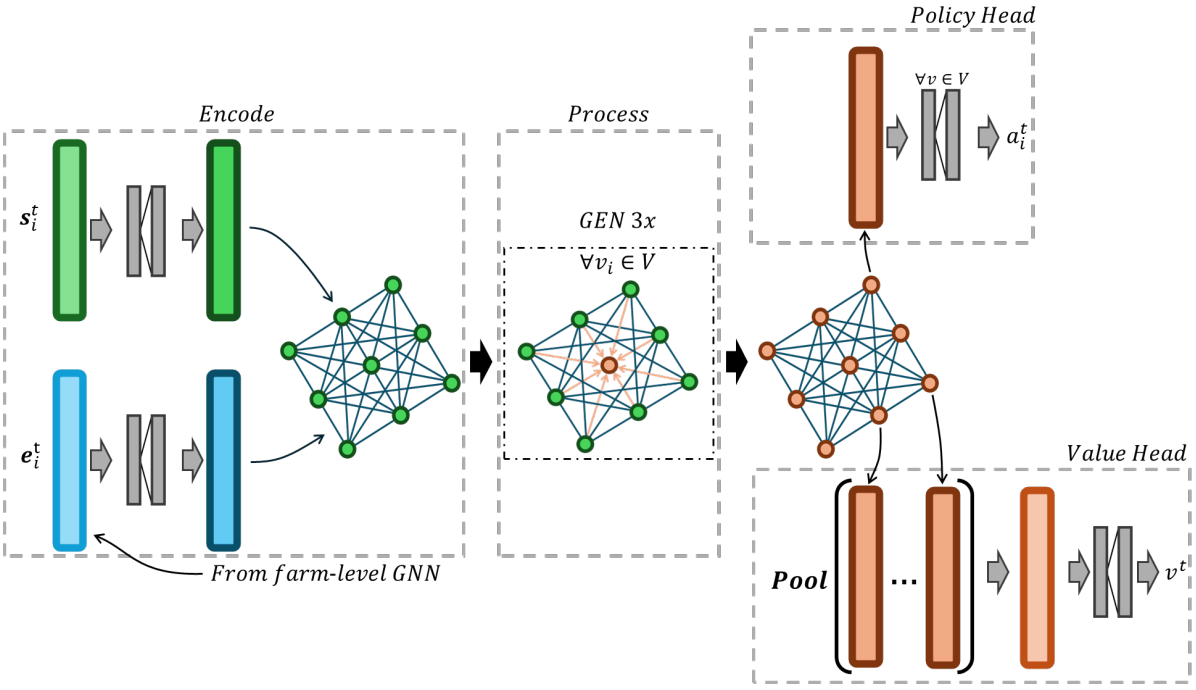

Figure 6.2: Overview of the agent's GNN architecture.

6.3. Baseline removal

The random processes in the environment, such as sampling wind conditions or electricity prices, introduce a lot of variance to the episodes encountered by the reinforcement learning agent. A single policy might, under various random circumstances in the environment, obtain rewards with a very high variance. Consequently, similar behaviour can result in very dissimilar rewards: the value of r_t does not strictly reflect the performance of a policy π, as it is also dependent on other processes in the environment. Such high variance in both environment and obtained rewards can destabilise training updates as the value function has a hard time correctly predicting future rewards, furthermore causing unstable policy updates. Dong et al. (2021) found similar problems in their work on RL for wind farm control, arguing that it is a significant training problem. The question then becomes how this variance can be reduced in order to stabilise training without introducing a bias that would change the underlying optimisation problem. Historically, this has been the topic of *variance reduction techniques* in reinforcement learning (Greensmith et al., 2001; Mao et al., 2019).

In order to reduce this variance, a baseline value can be removed from the reward obtained at each timestep. More specifically, a *state-dependent* baseline $b(s_t)$ can be removed (Schulman et al., 2018). Interestingly, subtracting such a state-dependent baseline from the reward function does not introduce a bias whilst significantly decreasing variance, as long as it is not a function of the action a_t taken (Seita, 2017). A logical choice for a baseline is the zero-yaw policy, which brings the additional benefit of ensuring the reward signal directly indicates how much better the trained policy is compared to the default policy. The baseline value $b(s_t)$, therefore, becomes the reward r_t the zero-yaw policy would have achieved during the same timestep. Since the stochasticity due to random processes in the environment affects the baseline as much as it would the agent, the reward signal gives a measure of *relative performance*, which can be assumed to have significantly less variance. Given the original reward from the environment $r(s_t)$ and the baseline reward at the same timestep $b(s_t)$, the transformed reward with baseline removal becomes as shown in Equation 6.2. The effectiveness of this technique will be investigated and discussed in subsection 6.4.1.

$$r'(s_t) = r(s_t) - b(s_t) \tag{6.2}$$

6.4. Training

Aside from baseline removal, several more techniques can be applied to ensure a more stable training process. Similarly to baseline removal, their effectiveness will be investigated in subsection 6.4.1. The following three adjustments are made:

1. **Observation normalisation** - All observations are normalised before being fed into the agent's network(s). Similar to supervised learning, networks perform better if the relative magnitudes of all input parameters are in the same range. Therefore, each variable v in the observation space is normalised according to $\frac{v-v_{min}}{v_{max}-v_{min}}$, where v_{max} and v_{min} are the maximum and minimum values of v respectively. This ensures each variable in the observation space is in the range of $[0, 1]$.
2. **Action normalisation** - The same principle applies to the output of the policy, e.g. the action space. All yaw angles, being between $[-30, +30]$, are normalised to a range $[0, 1]$. They are transformed back into the 'true' range when they are returned to the environment.
3. **Reward scaling** - Reinforcement Learning works better if the episode-wise rewards are not excessively big; this can cause unstable updates to the policy or value networks. As such, all rewards are scaled such that the per-episode sum of rewards is close to 1. For the 16-turbine farm case in subsubsection 7.3.2.1, all rewards are scaled by a factor 3840.

6.4.1. Training Technique Ablation Study

The aforementioned techniques in section 6.3 and section 6.4 all contribute to a stable and efficient training process. An ablation study is performed in this section to highlight the importance of each of these techniques. The training curves of various setups are shown in Figure 6.3. Each curve presents the mean and standard deviation range of three runs at the given setup. All runs were done using the 'greedy' objective, e.g. with only power optimisation as the goal. The abbreviation `MLP` stands for the usage of a regular Multi-Layer Perceptron architecture, whereas `GNN` indicates the usage of a

Graph Neural Network architecture. Furthermore, `BR` indicates the usage of Baseline Removal and `NORM` indicates observation, action and reward normalisation. Note that runs without baseline removal and normalisation had their rewards adjusted in post-processing to match the range of the other runs; this transformation simply involved removing the baseline reward and normalising it like it would have done in the other runs.

From the figure, it becomes evident that baseline removal is essential for the correct functioning of the algorithm. In fact, the runs without using baseline removal seem to not learn at all and likely stay stuck with the original randomly initialised weights of the MLP network. This network is initialised such that the outputs are centred around zero, meaning it tends to choose actions very close to the baseline zero-yaw policy. In some cases, it might thus slightly outperform the baseline, whereas in others, it under-performs. The rewards returned by the environment are too noisy to update the network correctly, and as such, training performance deteriorates, resulting in a total lack of learning. This also aligns with the hypothesis by Dong et al. (2021).

Introducing baseline removal ensures that learning at least starts to happen, albeit slowly and with a lower performance ceiling compared to the `MLP+BR+NORM` and `GNN+BR+NORM` runs. Further introducing the normalisation step allows the agents to achieve a greater performance. Finally, introducing the GNN architecture ensures faster convergence, as certain geometrical relations do not have to be learned implicitly inside the network but are encoded explicitly in the input. Note that the GNN architecture runs, over the same 24-hour runtime, could only run for two-thirds of the timesteps due to computational constraints. Both the `MLP+BR+NORM` and `GNN+BR+NORM` converge to the same final training reward. However, as will become evident later, the GNN architecture has some benefits compared to the MLP architecture. This will be discussed in subsubsection 7.3.2.4. All in all, baseline removal is especially essential for effective learning; normalisation significantly increases performance, and the GNN architecture allows for more efficient timestep-wise learning.

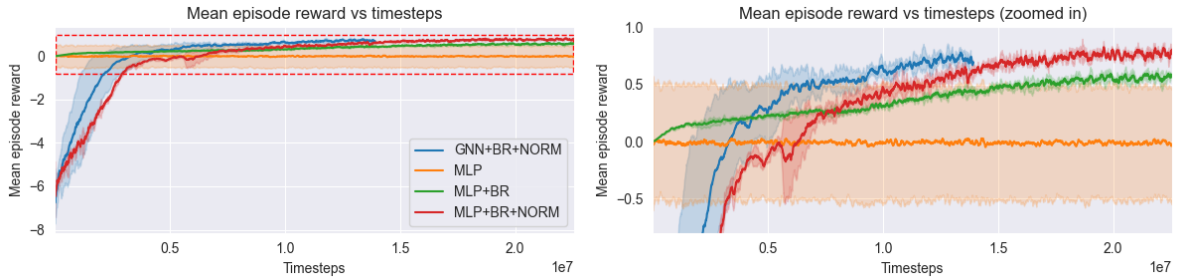

Figure 6.3: Training curves of runs with and without training techniques.

6.5. Chapter Recap

This chapter covered the setup of the reinforcement learning problem. First, the problem was formulated and a suitable RL algorithm was chosen. Next, a suitable model architecture for the agent was picked and constructed. Several techniques to improve learning were then presented, including a variance reduction technique called baseline removal. Finally, an ablation study was done to highlight the performance improvements that each training technique brings. Important is to note that from here on, *agent* refers to individual fully centralised controllers, e.g. neural network controllers. The *multi-agent system*, with multiple turbines ('agents'), is controlled by a single *actor* or *agent*. To answer the sub-question *How can the architecture of the reinforcement learning agent be designed for effective wind farm control, and what methodologies should be used in training to ensure efficient learning?*: using PPO with a graph-based architecture for the agent, and using variance reduction and normalisation techniques for reward signals. Now that the training setup is ready and training runs efficiently and effectively, the next chapter can proceed to evaluate the performance of the trained agents.

7

Policy Analysis

This chapter presents the evaluation of the agents trained using the Reinforcement Learning setup defined in the previous chapter. First, some Key Performance Indicators (KPIs) are defined in section 7.1; these are used to summarise the performance of the agents. Next, section 7.3 highlights the results of training and showcases various analyses of the performance of the trained agents. This includes an analysis of both an infinite horizon, as well as a finite horizon case; the difference between these two is highlighted in subsection 7.3.1. Furthermore, a generalizability study is performed to investigate whether the trained agents can generalise across farm topologies.

7.1. Key Performance Indicators (KPI)

To assess the performance of the trained agents, it is useful to define some KPIs. KPIs provide a single value with which the performance of several policies can be easily compared. For this, the following KPIs are defined:

- **Average Turbine Power (ATP)** - The average power produced by a turbine in the farm over a given time period. A higher value indicates better energy production.
- **Average Turbine Cost (ATC)** - The average maintenance cost for a turbine in the farm over a given time period. A lower value indicates fewer turbine costs.
- **Average Turbine Reward (ATR)** - The total sum of profit obtained through power production minus the maintenance costs incurred. It is the 'true' financial balance. A higher value indicates a better net profit.
- **Cost Of Energy (COE)** - Defined as the total maintenance cost divided by the total amount of kWh energy produced over the same period. It can be thought of as the price paid to generate each kWh. A lower value indicates better financial performance.
- **Mean Time Between Failures (MTBF)** - Defined as the mean time between failures of each component. In this environment, given the maintenance policies run components to failure, it is equal to the mean lifetime of each component. A higher value means longer lifetimes.

7.2. Layouts

Three different layouts will be used during training and evaluation; these are shown in Figure 7.1, Figure 7.2 and Figure 7.3. The Lillgrund and Horns Rev layouts are based on real-life wind farms.

7.3. Results

In this section, the results of training and evaluating the various agents in the environment are presented and discussed. Two cases will be discussed: the *'infinite'* horizon case, and the *'finite'* horizon case. The difference between these two will first be discussed in subsection 7.3.1. Next, subsection 7.3.2 encapsulates all training and evaluation done on an environment with effectively an infinite horizon. This includes investigating RL training and evaluation on the 16-turbine farm in subsubsection 7.3.2.1,

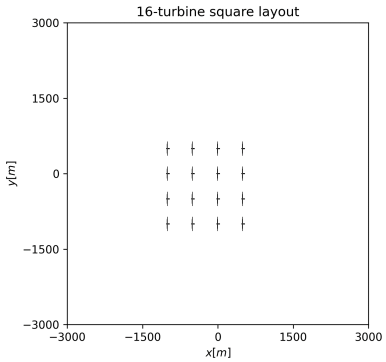

Figure 7.1: 16-turbine grid-aligned wind farm layout.

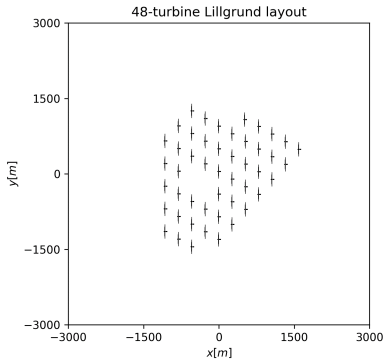

Figure 7.2: Layout based on the Lillgrund wind farm.

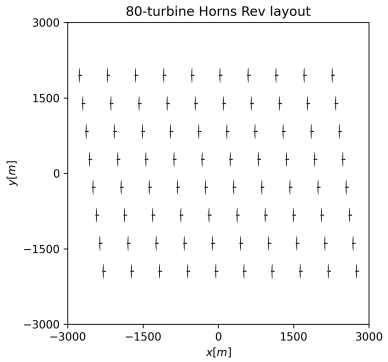

Figure 7.3: Layout based on the Horns Rev wind farm.

the Lillgrund farm in subsubsection 7.3.2.2 and the Horns Rev farm in subsubsection 7.3.2.3. Furthermore, in subsubsection 7.3.2.4, the ability of the trained agents to generalise to other farm layouts is investigated and evaluated. Finally, subsection 7.3.3 investigates the effect of evaluating the agents on a more realistic finite horizon environment with sparse costs.

7.3.1. Finite vs Infinite Horizon

There are two assumptions that can be made about the optimisation horizon of the reinforcement learning problem. The first case that will be covered is the *infinite horizon* case. It is concerned with optimising an objective function without explicitly considering a limited farm lifetime, essentially optimising for optimal static wind farm control. In the case of an infinite horizon, the expectation of the sum of sparse (realistic) costs is the same as the expectation of dense costs. As the time horizon approaches infinity, the costs incurred under either cost model within the same timeframe will be approximately identical. This is because the fraction of cost incurred under the final component replacement, which might only be utilised for a small time period, is insignificant compared to the rest of the simulation time. Consequently, the difference between the sparse cost model (not yet incurring cost for the final component) and the dense cost model (already incurring cost based on usage) is comparably small. This assumption can be further analytically proven by examining the worst-case scenario of deviation between dense and sparse costs, where a component is just on the end of the nth fatigue lifetime at the end of the horizon. The sparse cost model would have ensured that $n-1$ replacements have been paid for, whereas the dense cost model would have ensured that (approximately) n full replacements have been paid for. The difference between the two would be one component replacement cost; however, the relative difference between the total costs of both models is $\frac{c_{dense}-c_{sparse}}{c_{sparse}} = \frac{(n)\cdot C_{replace}-(n-1)\cdot C_{replace}}{(n-1)\cdot C_{replace}} = \frac{1}{n-1}$. Furthermore, $\lim_{n\to\infty} \frac{1}{n-1} = 0$. In this case, modelling the costs as dense costs hardly changes the underlying problem. It is thus safe to assume that the infinite horizon case can effectively be modelled as a short-term optimisation using dense costs, finding the optimal trade-off between component degradation and power optimisation. The policy is *static* with reference to wind farm age. This has the additional benefit of ensuring more frequent feedback for the agent, ultimately causing more stable and simpler learning of the optimal policy.

Assumption 7 *For the infinite-horizon case, the maintenance costs can be modelled as a dense, pay-per-use function based on fatigue damage accumulation without changing the underlying optimisation problem compared to sparse, pay-per-breakdown costs.*

When moving to a *finite* horizon case, the optimisation horizon is fixed at a certain length. This can be, for example, a 30-year window in which the farm is expected to be in use before full replacement. In this case, the dense cost model does not provide the discrete steps in maintenance costs which would otherwise allow for strategic planning of component failures. This could mean moving entire replacements just outside the optimisation horizon to save on large amounts of maintenance costs. Here, the sparse cost model would be the only realistic model and the assumption made above would no longer hold. In this thesis, the infinite horizon case is investigated and solved to obtain a static policy that maximises revenue regardless of the optimisation horizon. Note that the *fixed* episode lengths do

not change the *infinite* horizon assumption, as the inherently static policy means optimising for a shorter episode is equally as optimal as optimising till infinity. It is essentially optimising as if each episode was placed in sequence. The number of episodes used during training is there to improve diversity in encountered environmental conditions. The *infinite* horizon case is investigated later by evaluating the finite-horizon agents in it. Anyhow, in the following section the infinite horizon case with the dense cost model is investigated first.

7.3.2. Infinite Horizon

The infinite horizon optimisation problem is solved in this subsection for several objective functions. Each episode during training is 1000 timesteps long or approximately one week of simulated time. Each episode is assumed to be terminal, e.g. the agent learns to optimise for the one-week horizon; due to the assumption that was made earlier, this should not change the problem at hand. Each episode is initialised at a random point in the year to promote seasonal diversity and, thereby, variability in the encountered wind conditions and electricity prices. Table 7.1 highlights the hyperparameters used for the reinforcement learning algorithm. Since episodes are finite, a discount factor of 1 is chosen. A low entropy coefficient is picked to introduce exploration, without destabilising training. The learning rate, batch size, clip parameter and architecture hyperparameters were tuned manually to yield the highest average reward at the end of training.

Parameter	Value
Shared value/policy encoder	False
Node feature dimension	11
Node encoder layer dimensions	[64, 64]
Node encoder activation function	Tanh
Node latent dimension	64
Edge feature dimension	3
Edge encoder layer dimensions	[64, 64]
Edge encoder activation function	Tanh
Edge latent dimension	64
GNN layer type	GEN
GNN layer aggregation	SoftMax
Number of GNN layers	3
Policy head hidden layers	[64, 64]
Policy head activation function	Tanh
Policy head output dimension	2
Value head pooling method	Mean
Value head hidden layers	[64, 64]
Value head activation function	ReLu
Value head output dimension	1
Learnable Parameters	152,003
Learning rate	0.0001
Batch size (timesteps)	10000
Episode length (timesteps)	1000
Minibatch size (timesteps)	1000
Gradient descent iterations	20
Policy clip parameter	0.1
Entropy coefficient	0.001
GAE Lambda coefficient	0.1
Discount factor	1.0

Table 7.1: Overview of model architecture and hyperparameters for infinite-horizon reinforcement learning.

7.3.2.1. 16-turbine Case

In these first runs, the environment uses the 16-turbine layout as the wind farm to optimise for. Each of the agents was trained five times with different seeds; the training curves are shown in Figure 7.4, Figure 7.5 and Figure 7.6. Out of these five runs, the best agent is taken for each objective to use for evaluation. Evaluation takes place over 200 randomly initialised episodes, all five agents running with the same 200 different seeds. Each episode is 1000 timesteps long and runs with the dense cost model to provide cost statistics for the infinite horizon case, which - as was assumed earlier - can be modelled as such. The financially-oriented KPIs - ATP, ATC, ATR and COE - are shown in Figure 7.7, Figure 7.8, Figure 7.9 and Figure 7.10 respectively.

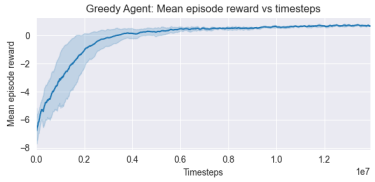

Figure 7.4: Training curve(s) of the naive agent with a one standard deviation confidence interval.

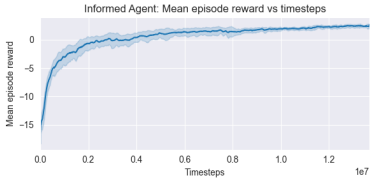

Figure 7.5: Training curve(s) of the informed agent with a one standard deviation confidence interval.

Figure 7.6: Training curve(s) of the risk-averse agent with a one standard deviation confidence interval.

Economic Analysis From these figures, it is evident that the random policy underperforms in power production compared to the zero-yaw baseline, with a relative difference in the order of a few per cent. To explain the relatively low loss in power production, it is essential to consider the conditions under which the surrogate models were developed. In section 3.3, the turbine-level model was developed based on OpenFAST simulations that inherently included an automatic pitch controller. This pitch controller aims to control blade pitch to ensure the turbine operates as closely to ideal conditions as possible, thereby extracting the most power out of the wind as possible. Under yawed conditions, this controller can increase pitch angles in an attempt to counteract the decreased wind velocity and swept area due to projection on a misaligned turbine. As such, it can mitigate a significant amount of the yaw losses that would have otherwise occurred without using a pitch controller. Despite what may seem like a disastrous policy - setting yaw angles at random - the pitch controller helps to minimise the losses, resulting in only a few per cent of power loss. These mitigating pitch actions combined with the yaw misaligned do, however, result in a significant increase in fatigue degradation, which is reflected in the nearly 60 per cent increase in maintenance costs. Consequently, both the total reward and cost of energy suffer from this.

The power optimisation agent performs excellently in power production, achieving a nearly 1.5 per cent increase in power on average. Furthermore, power production increased by as much as five per cent in some episodes. However, similar to the case with the random agent, a significant increase in component degradation is visible. Compared to the random agent, however, it is able to extract more energy out of the farm, at least counteracting the increased damage costs to a certain extent. Still, looking at the sum of profits and expenses, the power optimisation agent underperforms compared to the zero-yaw baseline. The net result is effectively an increase in the cost of energy and a decrease in revenue. The informed agent, however, aims to minimise this cost of energy directly or, more specifically, maximise the average turbine reward. Whilst doing so, it seems to lose out on some power production in favour of minimising damage. It seems like certain yaw angles, which are effectively not beneficial for wake steering and power optimisation, are instead very beneficial for damage minimisation. This is indeed reflected in the maintenance costs, which it is able to decrease by as much as fifty per cent. This, in turn, increases the average return and, likewise, decreases the cost of energy. Interestingly, the risk-averse agent achieves very similar results compared to the informed agent. Though the informed agent sees a slightly higher power production, which in both cases still lacks compared to the zero-yaw baseline, the risk-averse agent does marginally better in cost minimisation. The net result is a lower turbine reward but an almost identical cost of energy. Despite similar CoE, the informed agent simply produces more energy and, consequently, yields a higher revenue.

Lifetime Analysis Furthermore, the average lifetime of each component can be assessed with the Mean Time Between Failures (MTBF) statistic. This is shown in Figure 7.11. Effectively, this should reflect the changes in maintenance cost shown in Figure 7.8, as the cost is linearly related to damage. Indeed, an approximately 22-year lifespan for all components of the zero-yaw baseline agent can be observed. This is logical, as the fatigue degradation parameters were tuned in chapter 4 to match 22 years on average. The random agent sees very similar blade-area lifetimes but also significant differences in tower-area lifetimes. The yaw system seems to experience fewer loads due to a different distribution of aerodynamic loads on the blades. When it comes to tower loads, a large discrepancy can be seen in both fore-aft and side-to-side loads. Yawing the turbine causes the distribution of loads to change, as yawing essentially causes the direction of the loads on the tower top to change. Whereas under zero-yaw conditions, the tower experiences mainly fore-aft moments, many of these moments

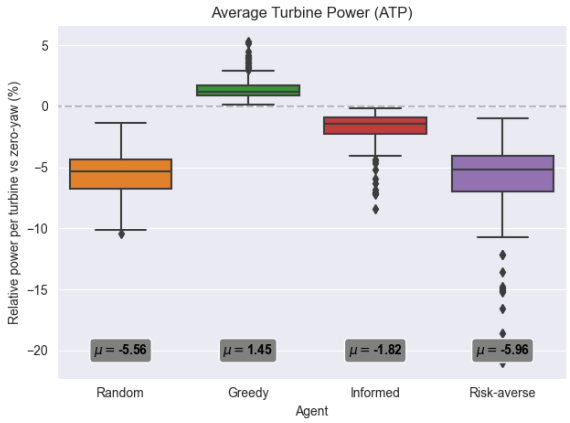

Figure 7.7: Per-turbine average power relative to baseline (zero-yaw).

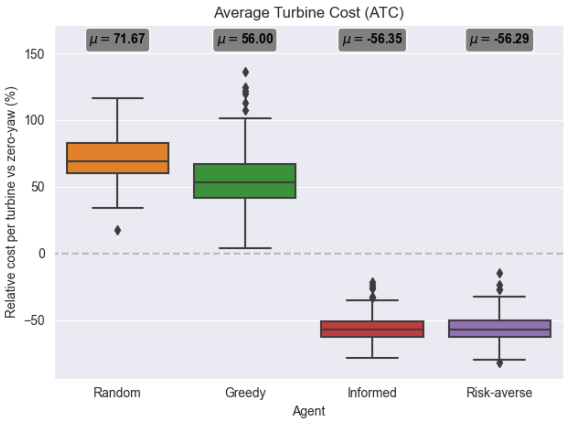

Figure 7.8: Per-turbine average total maintenance cost relative to baseline (zero-yaw).

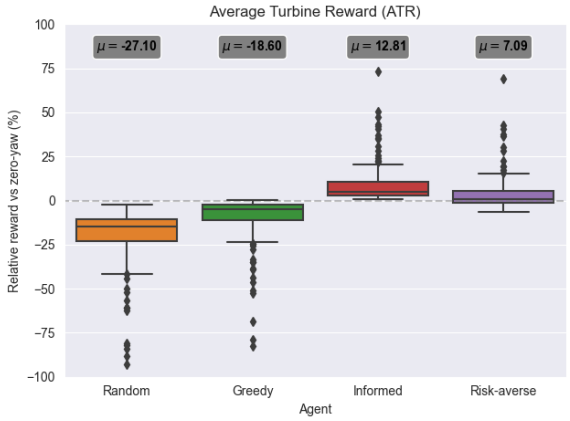

Figure 7.9: Per-turbine average total reward relative to baseline (zero-yaw).

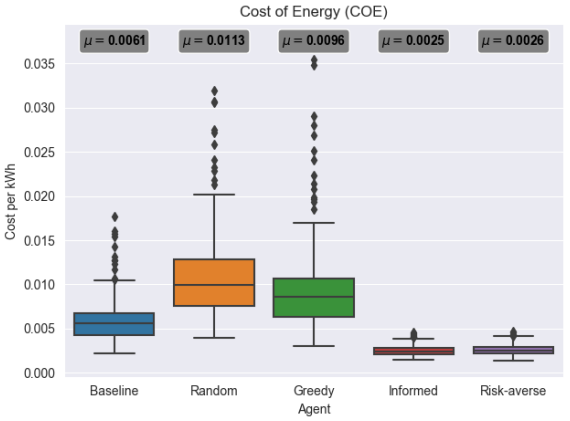

Figure 7.10: Average cost of energy.

are now offset with the yaw angles in yawed conditions. This causes a prolonged fore-aft lifetime and a shortened side-to-side lifetime due to their fatigue parameters, which were tuned on zero-yaw conditions.

The greedy agent, whilst optimising for power, seems to incur more flapwise and edgewise moments. Interestingly, the yaw system lifetime is prolonged, which contrasts with the observations from the random policy. The tower loads, however, show similar behaviour compared to the random policy: more yaw misalignments cause a shift in the distribution of fore-aft and side-to-side tower loads, leading to faster side-to-side failure and slower fore-aft failure. The informed agent incurs more flapwise loads but seems to manage to keep loads about equal edgewise. The 'sacrifice' of the relatively cheap flapwise components might be a trade-off to significantly prolong the fore-aft lifetime, as tower components are substantially more expensive to replace. Different behaviour, however, can be seen for the risk-averse agent: due to it not considering power production in any way, it does not feel the need to minimise power loss. This can result in being more conservative with the blades, thereby prolonging their lifetime and reducing cost. Still, the side-to-side lifetime has decreased, as has the yaw system lifetime. It is worth noting, also, that both the informed agent and risk-averse agent showed very similar performance in reducing costs despite having very different MTBF distributions, as is evident from Figure 7.11.

Policy Analysis - Yaw Angles versus Wind Direction There are several ways to inspect the agents' policy for their wake steering control. In Figure 7.12, the yaw angles for three turbines in the 16-turbine farm under varying wind directions are shown. Interestingly, the risk-averse agent generally chooses much higher yaw angles than the greedy and informed agents. To explain this behaviour, the load-yaw relations in Figure 7.13 can be studied. Here, it becomes evident that several loads have a tendency to

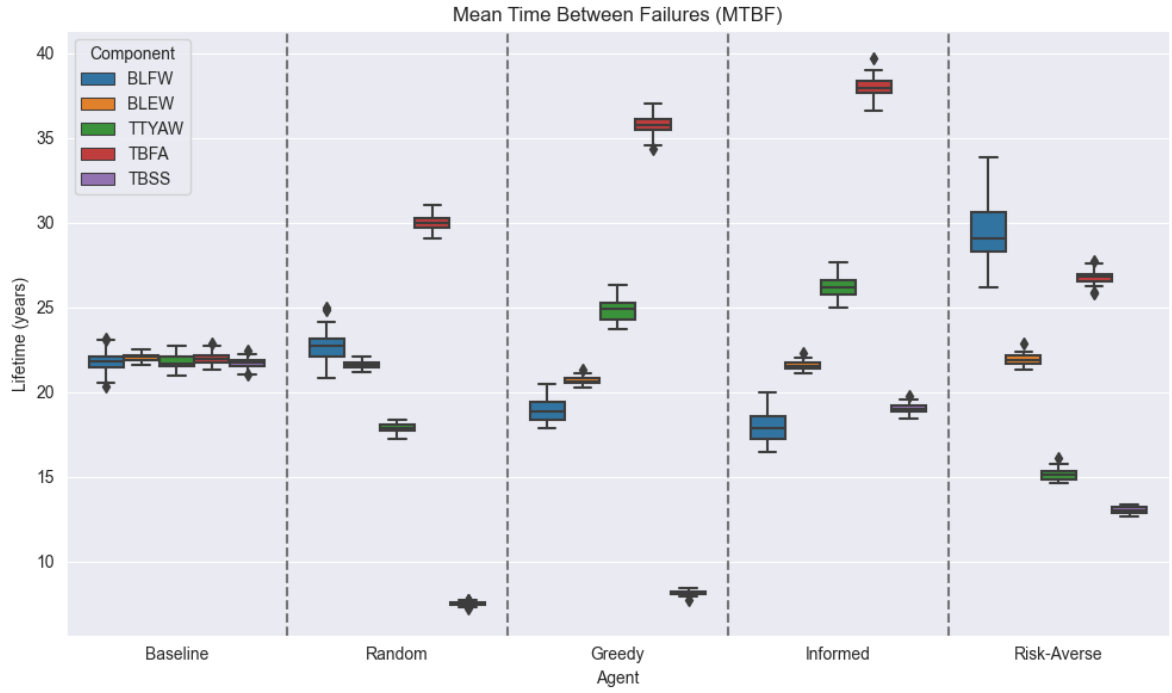

Figure 7.11: MTBF as a result of different control policies.

decrease as the yaw angle gets more positive. Given the damage-minimising nature of the risk-averse agent, it is only logical it tends to look for these global minima and yaw the turbines such that they operate in this region. The power-yaw curve, which degrades fast as yaw angles increase, plays no role here as profit (and therefore power) optimisation is not part of the risk-averse agent's objective function. Yawing the turbines within this low-load region effectively minimises both loads and significantly steers wakes, further helping reduce downstream turbulence and, therefore, downstream loads.

Both the greedy and informed agents use smaller yaw angles. Since in both their objective functions, power plays a role, high yaw angles would quickly lead to a large decrease in profit and therefore in observed reward. In other words, they are dealing with a more balanced trade-off between yawing for farm-level optimality and not yawing for turbine-level optimality. Here, the informed agent tends to take slightly smaller yaw angles in most cases, which matches the idea of certain loads increasing as yaw angles increase, such as the tower and yaw loads. Interestingly, both agents never choose a yaw angle of zero, which should seem optimal given that yaw misalignments generally lead to power decreases if the effects of wake steering are otherwise minimal. To explain this, the power-yaw curve in Figure 7.13 can once again be observed. From this curve, it is evident that the curve flattens out significantly around a zero-yaw angle. In other words, relatively small yaw angles can be applied without causing a significant change in power production. This flattening of the curve around the zero-yaw angle results from the pitch controller's attempt to minimise the losses of yaw misalignment by adjusting the pitch angles of the blades. This way, it can effectively, but to a certain extent, minimise the adverse effects of yaw misalignment at the cost of slightly higher loads. Having a slight yaw angle at all times thus means two things: an always-present wake steering effect and no significant reduction in produced power.

As for the yaw behaviour relative to the wind direction, there seems to be a discontinuity at zero degrees. This likely follows from the fact that the angles are represented on a linear scale from 0 to 360, whereas in reality, they would loop around at the end. This cyclic nature of angles is not captured on a linear scale, and the optimisation problem likely does not provide enough feedback to force the yaw angles to meet up at either end. There are no clear peaks in yaw angle for each cardinal direction in which turbines would typically line up relative to the wind, and wakes would be worst. Likely, this is because the turbines, as mentioned before, already have a few degrees of misalignment on average, which already ensures a certain degree of wake steering.

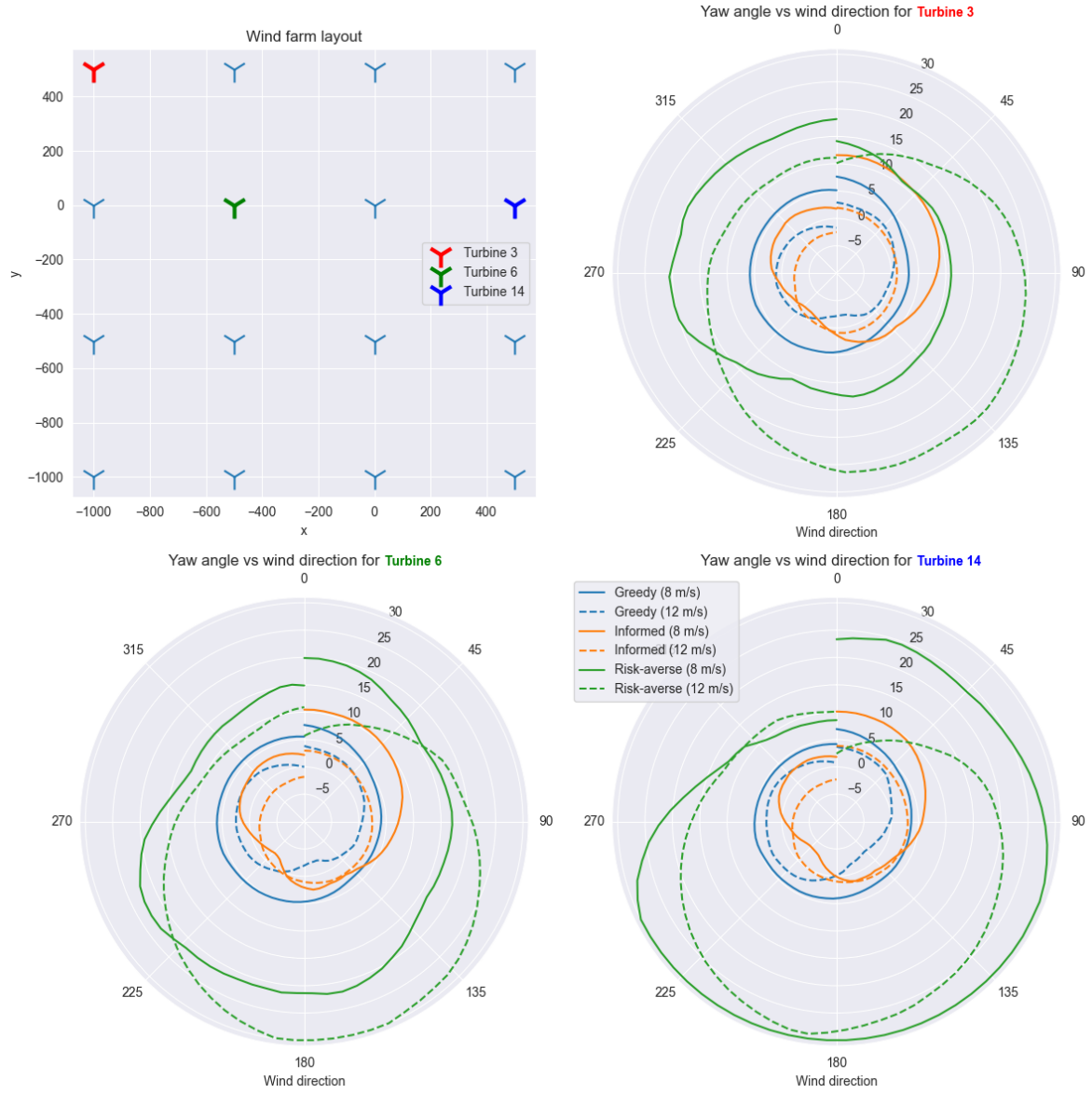

Figure 7.12: Yaw policies versus wind direction at $I = 0.1$ and $\alpha = 0.1$ with $V_m = 8.0$ m/s (solid) and $V_m = 12.0$ m/s (dashed).

Greedy Policy Analysis - Yaw Angles Farm-Wide The yaw angles and power improvements relative to baseline throughout the wind farm can also be inspected. The yaw angles for the 16-turbine farm subject to Eastern and Western wind can be found in Figure 7.14 and Figure 7.15, respectively. Note that the wind directions are slightly offset as the cardinal directions themselves are local minima for power production, as will be discussed in the paragraph on power improvements versus wind direction. Generally, a gradient of yaw angles can be discovered that goes from high (upstream) to low (downstream). This is logical, as the upstream turbines typically have more downstream turbines where their wakes will end up. As such, yawing the upstream turbines can have a tremendous impact on total farm performance. The more downstream the turbine is, the less positive effect a yaw misalignment will have and, thus, the lower the chosen yaw angles are. These gradients are what would typically make sense in a wake steering policy, and these observations align with the conclusions drawn by Zong and Porté-Agel (2021). Furthermore, it is evident that the front turbines suffer a power loss compared to the baseline, which is then compensated by the improvement of downstream turbines' performance to obtain a net farm-wide improvement. The lack of zero-yaw angles, similar to what was discussed before, is likely due to the power curve flattening out significantly around zero due to the pitch controller

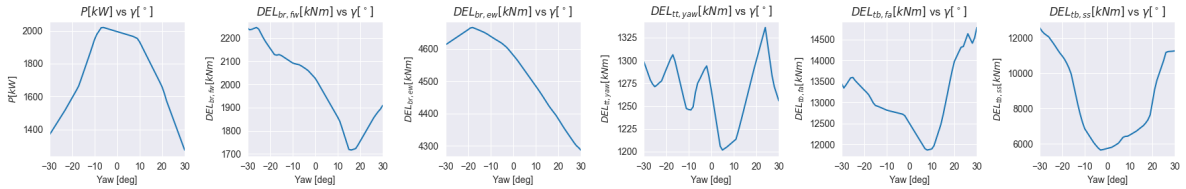

Figure 7.13: Power and loads versus yaw angle at $V_m = 8.0$ m/s, $I = 0.1$ and $\alpha = 0.1$.

adjusting for the effects of yaw misalignment.

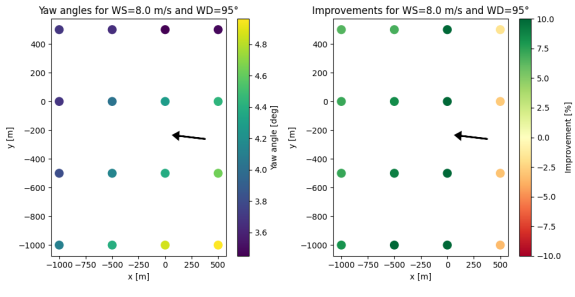

Figure 7.14: Yaw angles under Eastern, with $V_m = 8.0$ m/s, $I = 0.1$ and $\alpha = 0.1$.

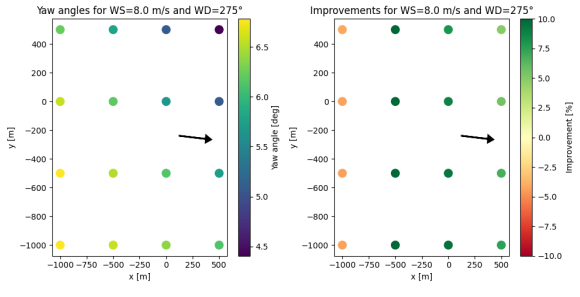

Figure 7.15: Yaw angles under Western wind, with $V_m = 8.0$ m/s, $I = 0.1$ and $\alpha = 0.1$.

Greedy Policy Analysis - Power Increase versus Wind Direction Furthermore, the power improvement relative to baseline (zero-yaw) can be plotted as a function of the wind direction. This is done for $V_m = 7$, $V_m = 10$ and $V_m = 14$ m/s in Figure 7.17, Figure 7.18 and Figure 7.19 respectively. It is evident that the biggest increases in power production happen when the wind comes from any of the four cardinal directions or any of the diagonals. This can be explained by looking at the farm layout, where most turbines will sit behind each other in each of those directions. The baseline policy thus suffers most in these directions, and the biggest power increases are achievable. Interestingly, the major improvements are shifted a few degrees relative to the aforementioned directions. Instead of the (expected) improvement at 90 degrees, it rather happens at around 95. This is because if the turbines line up perfectly, a rather large deflection of the produced wake is necessary to move a portion of the wake away from downstream turbines. In other words, it takes a large sacrifice (yaw misalignment) of the upstream turbine to deflect the wake sufficiently out of the way of downstream turbines. Under 'perfectly aligned' conditions, this relatively large yaw angle might cause a performance decrease that is too big for the upstream turbine to be balanced out by farm-level gains. Zong and Porté-Agel (2021), in their research, came to the same conclusion, arguing that full-wake conditions (e.g. direct alignment of turbines) cause ineffectiveness in wake steering and that the best performance increases are found in partial-wake conditions.

Another interesting behaviour to note is the decrease in power improvements as wind speed increases. In Figure 7.18 the relative improvements have already begun to decrease significantly, and in Figure 7.19 they are even gone completely. To explain this behaviour, the wake steering effect under high wind speeds must be investigated. This is shown for a relatively low wind speed ($V_m = 8$ m/s) in Figure 7.20 and high wind speed ($V_m = 16$ m/s) in Figure 7.21. From these figures, it becomes clear that at higher wind speeds, the deflection due to yaw misalignments, which form the basis of the wake steering concept, decreases. In other words, as wind speed decreases, wake steering becomes less effective and requires higher yaw angles. This is reflected in the results, where wake steering becomes less and less effective as wind speed increases. However, there is one more reason why wake steering becomes less effective at higher wind speeds. By inspecting the power-versus-windspeed curve in Figure 7.16, it becomes evident that above $V_m = 12$ m/s the power does not change as wind speed changes. This is the effect of the turbine controller adjusting pitch angles to ensure the turbine operates at rated power. This effectively means that even if upstream turbines cause a wind speed deficit in the produced wake, it might still be enough to 'saturate' the turbine, e.g. the change in wind speed does

not affect produced power. As such, wake steering will provide no benefit, as there is plenty of wind speed throughout the whole farm despite the wake effects.

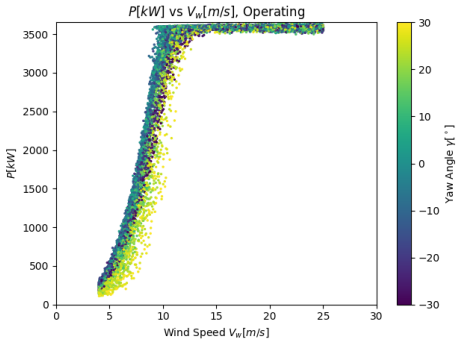

Figure 7.16: Power versus wind speed; colouring is based on yaw angle.

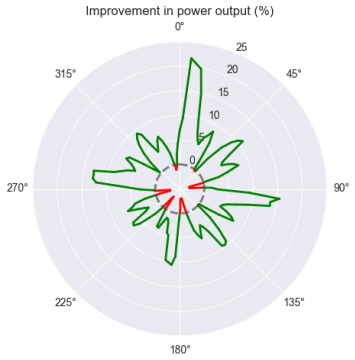

Figure 7.17: Power improvement vs wind direction, at $V_w = 7$ m/s, $I = 0.1$ and $\alpha = 0.1$.

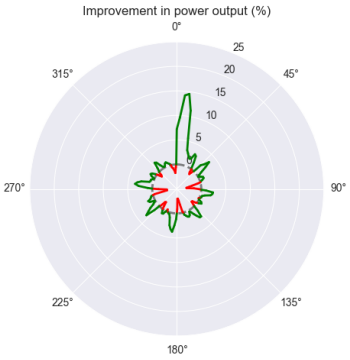

Figure 7.18: Power improvement vs wind direction, at $V_w = 10$ m/s, $I = 0.1$ and $\alpha = 0.1$.

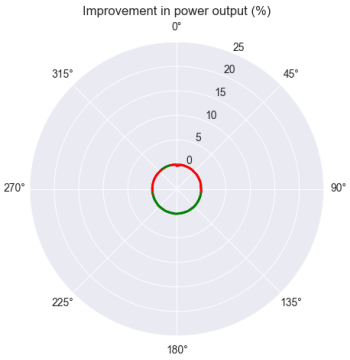

Figure 7.19: Power improvement vs wind direction, at $V_w = 14$ m/s, $I = 0.1$ and $\alpha = 0.1$.

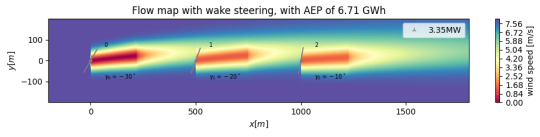

Figure 7.20: Flow map at $V_m = 8$ m/s, $I = 0.1$ and $\alpha = 0.1$.

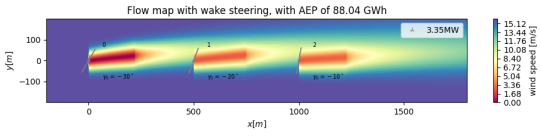

Figure 7.21: Flow map at $V_m = 16$ m/s, $I = 0.1$ and $\alpha = 0.1$.

Informed Policy Analysis - Yaw Angles versus Electricity Price The electricity price is part of the agent's observation and might be especially relevant for the informed agent. Since the informed agent acts to optimise the balance between profit through energy production and cost through component degradation, the price of electricity is explicitly part of the trade-off. To investigate to which extent the informed agent considers this price, the average yaw angle versus the electricity price can be plotted, as is done in Figure 7.22. Two interesting behaviours can be derived from these results. On the one hand, all yaw angles tend to zero as wind speed increases; this is consistent with the conclusion that was drawn before, where wake steering becomes ineffective at high wind speeds. Thus, having yaw angles provides no benefit with regard to power production whilst also increasing the loads due to the yaw misalignment. On the other hand, when wake steering is effective (e.g. at wind speeds around 8 m/s), an increase in electricity price yields an increase in average yaw angles used throughout the farm. This makes sense, as higher prices mean more benefit from wake steering whilst costs remain as usual, making it more attractive to use wake steering. As such, higher yaw angles that exploit the

bigger benefits of using yaw misalignments can be observed. Another interesting behaviour to notice is the tendency of the average yaw versus price curves at 7 and 8 m/s to drop below zero at low prices. This can have two causes. One option is that at low electricity prices, the policy is to go into damage minimisation mode as electricity is worth very little compared to the damage the generation causes. As such, the policy looks for a local minimum for the turbine loads, which might, under certain waked wind conditions, be at a slight negative yaw angle. The second option is that, since electricity prices are sampled from a categorical distribution based on real-life data, low prices are very rare, and the agent has seen very few of them during training. This is further supported by the analysis of prices shown in Figure 5.12, where electricity prices as low as 10 or 20 rarely occur on average. This effectively means that during evaluation, the policy could be tested out of the distribution it has seen during training. Behaviour at such low rates might thus not be entirely realistic and would, in general, rarely occur.

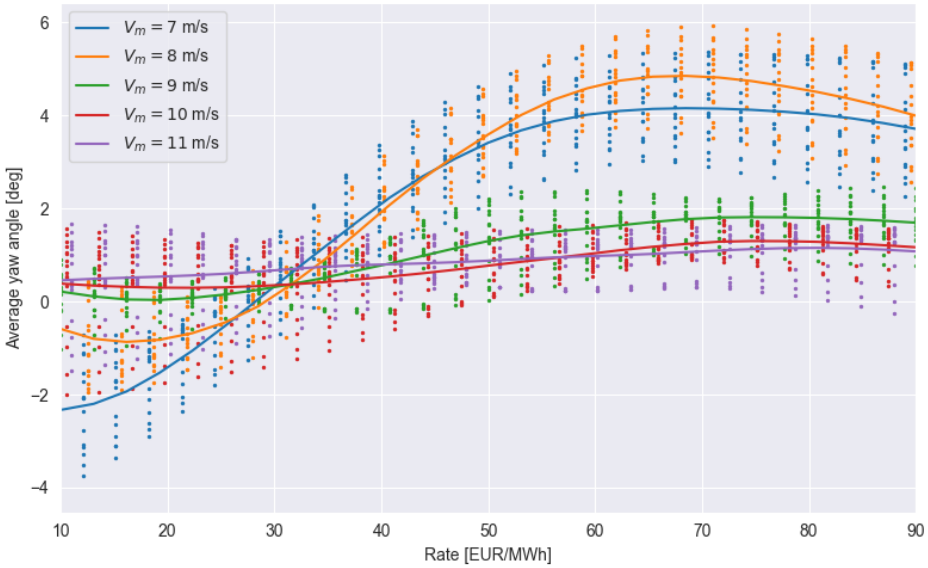

Figure 7.22: Average yaw angle versus the price of electricity.

Informed Policy Analysis - Yaw Angles versus Damage State The damage state D of all components of each turbine is part of the observation. However, the damage state itself in the infinite-horizon case should not affect the policy as it does not directly influence the reward. The cost at each timestep is determined using ΔD, invariant to the current instantaneous state D. The agent should thus ignore the damage state during the decision-making process. To investigate this, the yaw angles of each of the turbines under consideration in Figure 7.12 can be plotted as a function of both wind direction and damage state in Figure 7.23. Indeed, the policy remains unchanged under changing damage states, indicating the policy is invariant to the damage state of the turbines in the environment. This is logical, as there is no reason to change behaviour if the turbine is near failure - the component will be replaced 'free of cost'. All the agent observes is a component which is paid for based on usage, regardless of the number of replacements. Still, the replacement will ensure a certain downtime of the turbine; the agent could still attempt to use wake steering strategically to minimise the number of downtimes. However, the influence these downtimes have on the policy is so minimal that they are not considered, as is evident from its damage-invariant behaviour. This damage-invariant property is also present in greedy and risk-averse policies.

7.3.2.2. Lillgrund Case

To investigate training and execution performance on a larger wind farm, the above steps and some analyses are repeated on the Lillgrund wind farm with 48 turbines. Due to computational memory constraints, the `minibatch size` was set to 500. Furthermore, to discourage divergence of the policy due to it preferring to increase entropy instead, the `entropy coefficient` was set to 0. All runs were stopped after 24 hours of runtime, applying the same constraint as was the case for the 16-turbine

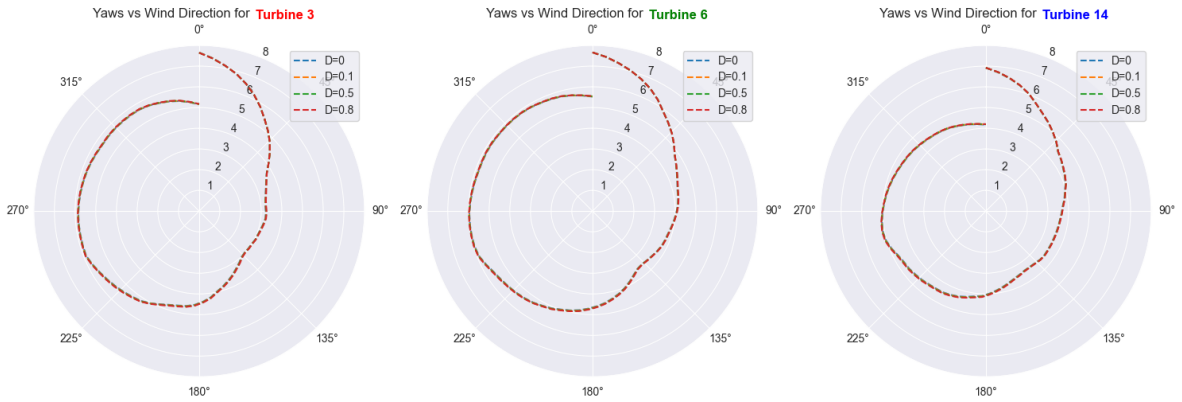

Figure 7.23: Yaw angle versus wind direction at $V_m = 8.0$ m/s, $I = 0.1$ and $\alpha = 0.1$, at various damage states. The damage state is equal for all components in all turbines.

runs. The training curves can be found in Figure 7.24. Note that the training episode reward does not accurately reflect the true performance of the agent during evaluation, as it might still include exploration and has not yet converged to a low-entropy (low-variance) policy within the training time.

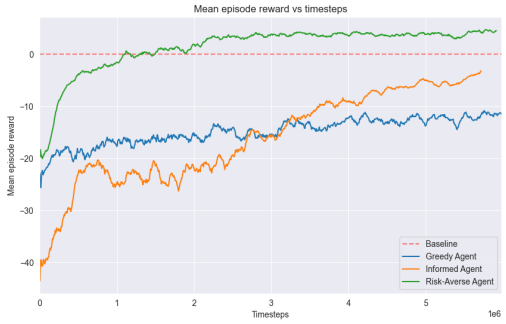

Figure 7.24: Training curves for the Lillgrund agents.

A very similar trend in results, compared to the 16-turbine case, can be seen in the evaluation statistics. Figure 7.25 still indicates the greedy agent optimises power by about 0.7%, but causes significantly more damage and thereby cost in Figure 7.26. The informed and risk-averse agents manage to bring down costs, as expected, and reduce the cost of energy. However, what becomes evident by inspecting the results is that the relative performance increases for each of the agents are not as high as they were for the 16-turbine case. Despite what one might expect, there is, for example, a lower power increase. Similarly, the cost of energy in Figure 7.28 is relatively high compared to the performance of the informed and risk-averse agents in Figure 7.10. One might expect, however, that these relative performance increases could be of higher magnitude due to there being more wake effects and a larger number of downstream turbines at all times.

To explain why this is the case, multiple characteristics and statistics must be inspected. For one, there seems not to have been a complete and proper convergence of the informed agent during training, as is evident from the corresponding training curve in Figure 7.24. This observation is the result of a bigger underlying challenge, which is the difficulty of fully centralised multi-agent controllers to deal with systems including many agents. As the number of agents grows, the combined observation and action spaces grow exponentially. This issue is commonly referred to as the 'curse of dimensionality'. In the 16-turbine case, the agent was able to explore this action space quite reasonably and could eventually converge to a single policy within a reasonable time. In the Lillgrund case, with 48 turbines and, therefore, an equal amount of 'agents', the fully centralised learning paradigm has trouble finding the optimal policy. This behaviour is commonly seen in fully centralised multi-agent controllers and proves to be an issue as the number of agents scales. Yet, despite the training issues, each of the agents is able to improve compared to the baseline in each of their respective objective functions.

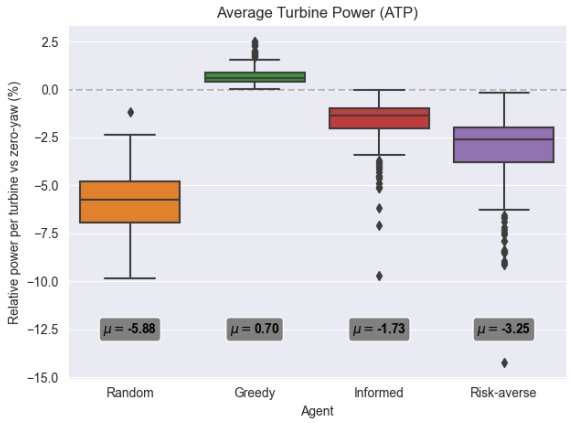

Figure 7.25: Lillgrund per-turbine average power relative to baseline (zero-yaw).

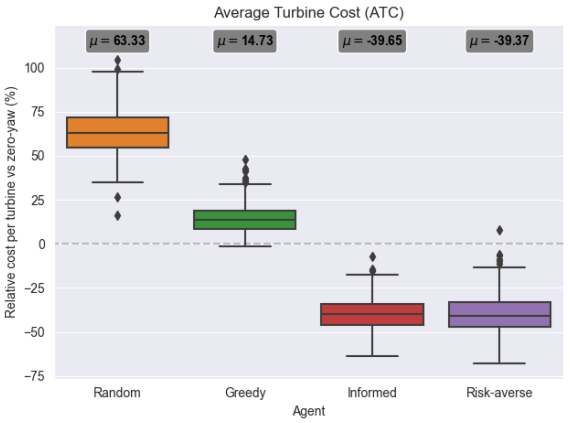

Figure 7.26: Lillgrund per-turbine average total maintenance cost relative to baseline (zero-yaw).

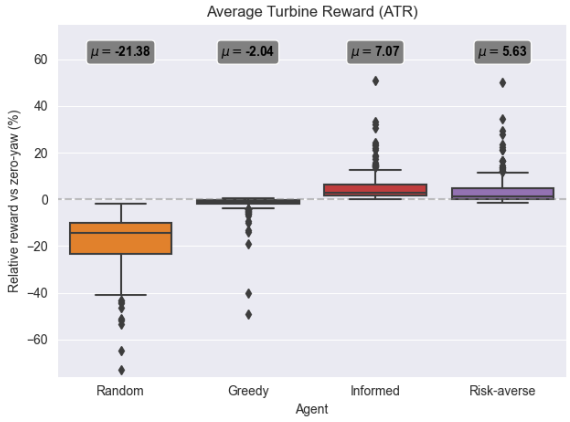

Figure 7.27: Lillgrund per-turbine average total reward relative to baseline (zero-yaw).

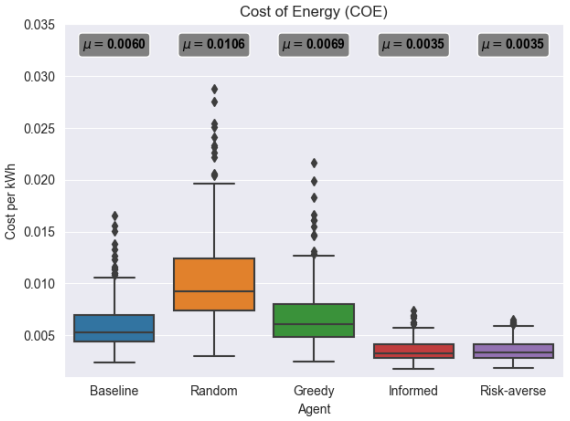

Figure 7.28: Lillgrund average cost of energy.

7.3.2.3. Horns Rev Case

Similarly to the Lillgrund case, the agents can be trained on the Horns Rev farm with as many as 80 turbines. Here, too, due to computational memory constraints, the `minibatch size` had to be set to 250. Furthermore, the `entropy coefficient` was set to 0 to encourage convergence during training. All runs were stopped after 24 hours of runtime, identical to the training runs for the 16-turbine and Lillgrund cases. The training curves can be found in Figure 7.29.

From the training curves, it becomes evident that the algorithm has stopped, to some degree, with learning near the end of the training progress. Reward increases seem to stagnate, and the curves flatten out. All three curves seem to stagnate at a training reward below that of baseline, though the training reward does not accurately represent the true agent performance. Looking at the evaluation statistics for each of the agents, again, very similar trends can be seen compared to the 16-turbine and Lillgrund cases. The greedy agent is able to optimise power in Figure 7.30 and the informed and risk-averse agents are able to minimise costs and therefore the cost of energy in Figure 7.31 and Figure 7.33 respectively. However, it becomes evident that these performances are even worse than that of the 16-turbine and Lillgrund cases. In fact, there is a downward trend in all these results as the number of turbines increases. Despite the agent being able to optimise the rewards to some extent, the more turbines the system has, the harder it gets to explore the joint action space effectively and find the best policy. There is effectively a growing scalability challenge with the centralised learning paradigm, where the larger the system becomes, the lower the relative performance increases get.

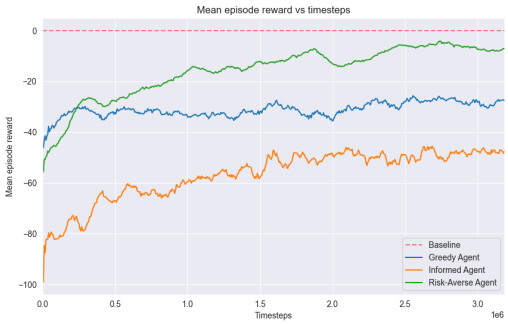

Figure 7.29: Training curves for the Horns Rev agents.

7.3.2.4. Generalisability

During the construction of the RL agent's deep learning architecture, no assumptions were made on the number of turbines and, therefore, on the graph topology. In fact, in theory, the architecture would function just fine on *any* number of connected turbines in *any* topology and connectivity. The question is, however, whether in the context of wind farm control, using such trained graph-based agents enjoys the same topology-agnostic properties as the architecture it is built on. In other words, whether the trained agent *generalises* to wind farm topologies it has not yet encountered during training. This would effectively mean that the problems with centralised learning can be circumvented by leveraging the convergence property in small farms whilst still enjoying performance improvements when transferred to larger farms. To investigate this, an agent trained on one layout can be transferred to control another layout, and its performance can be assessed.

In the first experiment, the 16-turbine case controllers are applied to the Lillgrund farm layout. Its performance under the various objective functions can be seen relative to the zero-yaw policy's performance. Furthermore, the performance of the controllers trained on the Lillgrund layout directly can also be found in the same figure. Interestingly, these statistics indicate that the 16-turbine controller seems to generalise quite nicely to the Lillgrund wind farm, confidently outperforming the zero-yaw policy. In fact, it is even able to outperform the controller specifically trained on the Lillgrund farm. This likely stems from the fact that the fully centralised multi-agent training paradigm has trouble exploring and converging to a policy under the larger amount of turbines, as discussed in subsubsection 7.3.2.2. The 16-turbine case did not suffer from this issue and was thus able to converge to a policy just fine. All in all, the 16-turbine agents are able to achieve nearly the same performance increase as they did on the 16-turbine farm.

Similarly, the 16-turbine controller can be applied to the Horns Rev wind farm layout. Both zero-yaw and 'Horns Rev' policies are shown alongside the performance of the 16-turbine policy. Again, the 16-turbine controller seems to outperform the zero-yaw policy confidently. As was the case in the generalisability experiment for the Lillgrund farm, the 16-turbine again outperforms the controller trained specifically on the Horns Rev farm for the same reason as before. The performance increases are *nearly identical* for all three wind farms, despite their topologies and number of turbines varying wildly.

This property of the trained agent to generalise to other farms comes from its graph-based structure. In the architecture, there is no notion of *absolute* turbine locations in the farm; instead, all turbine positions are treated as *relative* to others. This property is embedded in the edge features, which indicate relative positions and headings between each other and the wind. As such, the network learns to determine which relative positions require which actions to optimise the environment reward. When presented with new and unseen farm layouts, it can apply these rules it has learned no matter the number of turbines or their positions. As long as *relative* positions are known, the wake effects are implicitly considered in the network, and the right actions can be inferred. To investigate the behaviour of the agents on the new farm topologies, the yaw angles and power improvements of the Greedy agent under various wind directions can be plotted for both the Lillgrund and Horns Rev farms. In Figure 7.42 and Figure 7.43, the actions taken under the two extreme cases of full-wake conditions are shown. In both cases, the front row suffers a power loss compared to the baseline, but the turbines downstream show a power improvement due to the deflected wakes. Furthermore, a gradient of yaw angles is

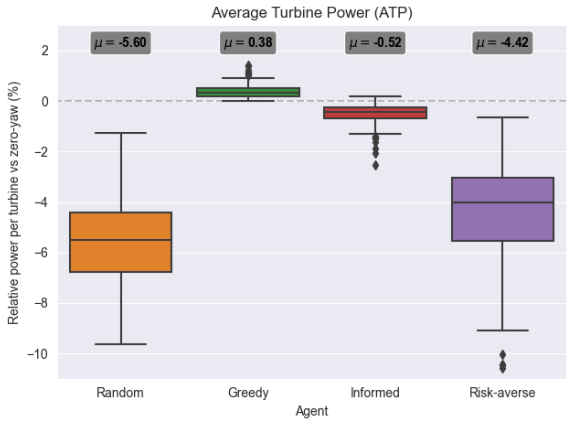

Figure 7.30: Horns Rev per-turbine average power relative to baseline (zero-yaw).

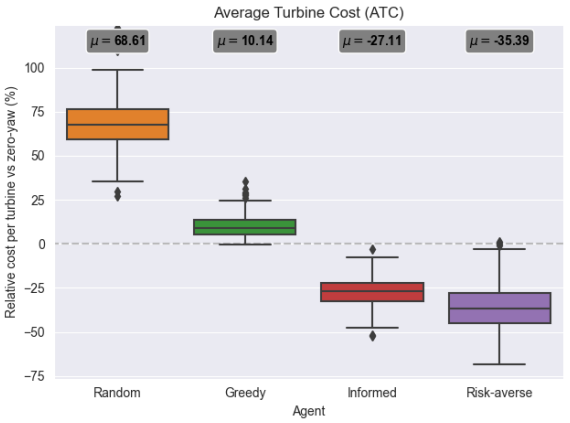

Figure 7.31: Horns Rev per-turbine average total maintenance cost relative to baseline (zero-yaw).

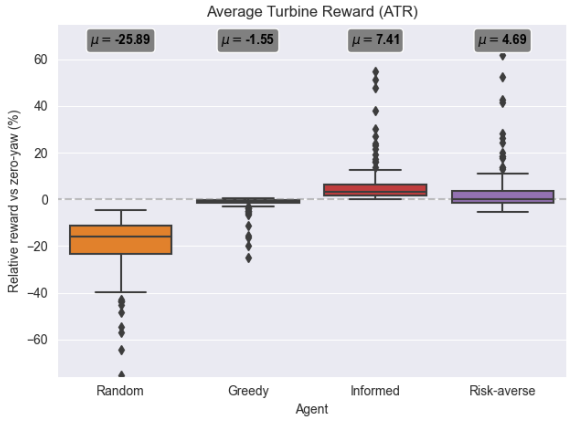

Figure 7.32: Horns Rev per-turbine average total reward relative to baseline (zero-yaw).

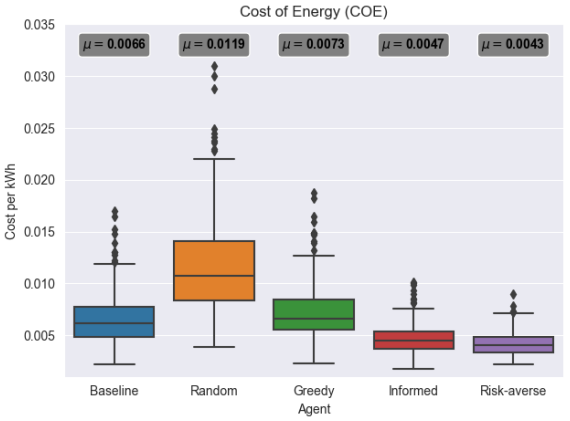

Figure 7.33: Horns Rev average cost of energy.

visible in the farm with upstream turbines typically showing higher yaw angles. The same behaviour can be seen in Figure 7.45 and Figure 7.44 for the Horns Rev wind farm. There is clearly a trend of upstream turbines yawing more to cause more wake deflection, thereby sacrificing their own power production in favour of the production downstream. All cases showed a farm-wide power improvement of several per cent compared to the zero-yaw baseline.

The question now becomes whether training on smaller farms, which ensures better converging during training, can be adjusted to better generalise to larger farms. In other words, what can be changed about the training setup to allow it to generalise better to unseen farms, e.g. make it more layout-agnostic? How can the *generalisability* property be leveraged to enable convergence during training but high performance during inference? One problem that could emerge in training on smaller farms is the overfitting on the edge features that are present in the farm. The 16-turbine farm presents very geometrically perfect angles and distances, ones which are unlikely to be seen in more complex farm layouts. Furthermore, it trains on a farm with nice grid-like layouts, which were also present at the evaluation. During training, all the agent sees is this single set of edges, and there is no guarantee that it will extrapolate its behaviour nicely to more diverse graph structures. To tackle this, inspiration can be taken from training the farm-level surrogate in section 3.4; here, layouts were randomised in the training dataset. To apply this technique to RL training, the layouts can be randomised after each episode. To ensure consistency in observation and action spaces, the number of turbines should be fixed, but the way in which they are arranged can be arbitrary. Note that this is only possible because the graph-based architecture does not learn policies for absolute turbine locations but rather for relative locations. This ultimately allows the layout to shift in whatever way, as long as relative positions are known the right actions can be inferred. Furthermore, the number of turbines is kept at 16 to ensure

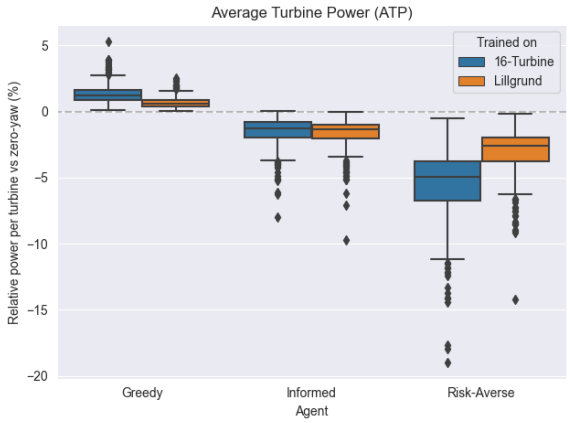

Figure 7.34: Per-turbine average power: the transferred 16-turbine agent vs the Lillgrund agent.

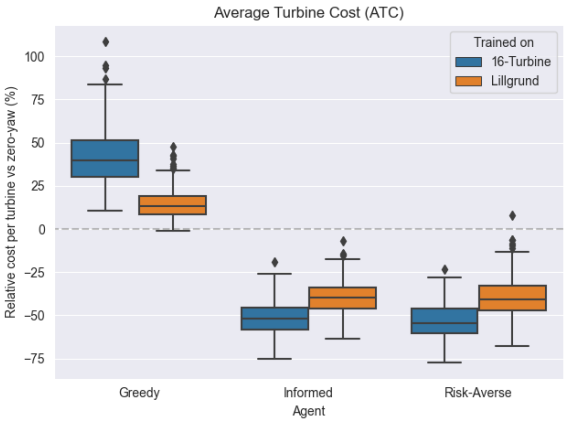

Figure 7.35: Per-turbine average cost: the transferred 16-turbine agent vs the Lillgrund agent.

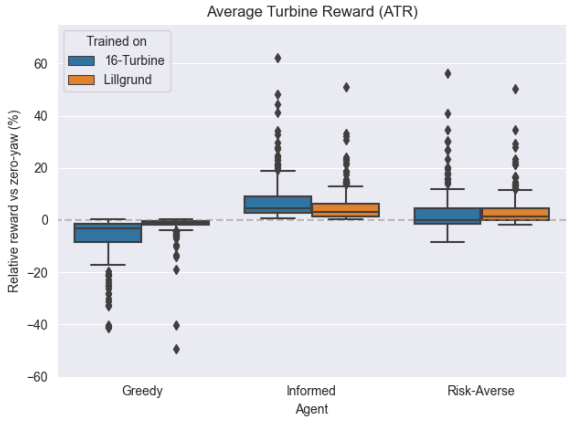

Figure 7.36: Per-turbine average reward: the transferred 16-turbine agent vs the Lillgrund agent.

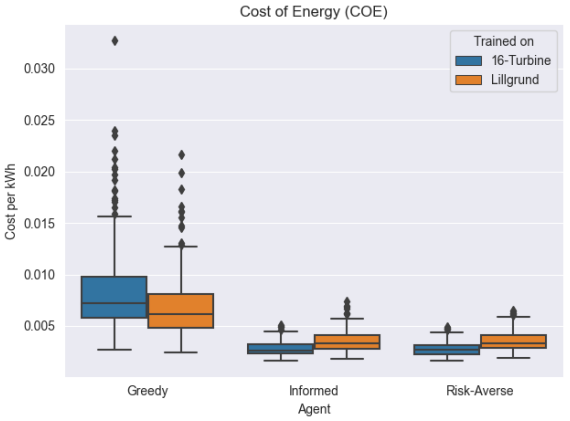

Figure 7.37: Average cost of energy: the transferred 16-turbine agent vs the Lillgrund agent.

that the convergence of the policies still happens and that the limitations of fully centralised learning do not constrain the agent. Eight samples of possible wind farms the agent might encounter in its episodes of training are shown in Figure 7.46.

As can be derived from the training curves in Figure 7.47, all agents still seem to converge to a final solution. This indicates that the introduction of the random layouts has not destabilised training and that generalizability, even during training, is very much present. Next, both the agents trained on the fixed 16-turbine farm, which is denoted `16T`, can be compared with the agents trained on the random 16-turbine farms, `16R`. This will be done on four new layouts, each of which is purposely chosen to not explicitly have a perfect grid-like layout and to include some features which might be hard for the 16T agent to generalise to. These layouts are illustrated in Figure 7.48. Evaluation is done in the same way as was the case before, except each agent is only evaluated on the objective function it optimised for during training. Furthermore, the COE KPI is shown for all agents. In all cases, the agent trained on the random 16-turbine layouts outperforms the previous 16-turbine agent when it comes to power production. It generalises better to wind farm layouts with different turbine spacings and relative angles. Regarding informed and risk-averse policies, all agents seem to operate very similarly with negligible differences. In fact, the COE of all these agents is nearly identical. The inability of the 16R agents to improve on these metrics likely stems from the fact that these policies' performance increases are primarily dominated by a reduction in costs, which mostly happens on a local turbine level. The cost reduction for both the risk-averse and informed 16R policies is slightly worse than that of the 16T ones. For power optimisation, wind farm topology and relative locations become especially important, hence the noticeable power increase of the 16R agent which seems to generalise better. The informed and risk-averse agents, however, benefit mostly from damage-mitigating behaviour at the turbine level,

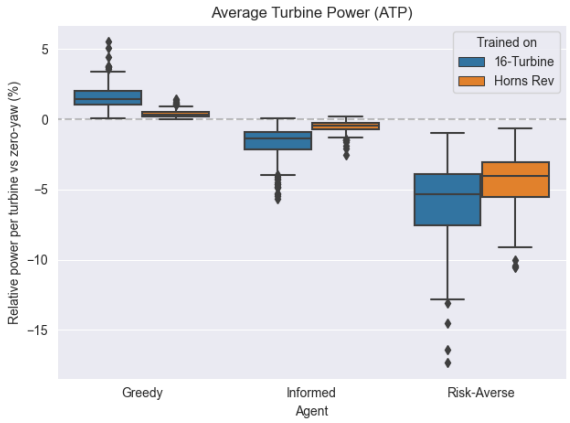

Figure 7.38: Per-turbine average power: the transferred 16-turbine agent vs the Horns Rev agent.

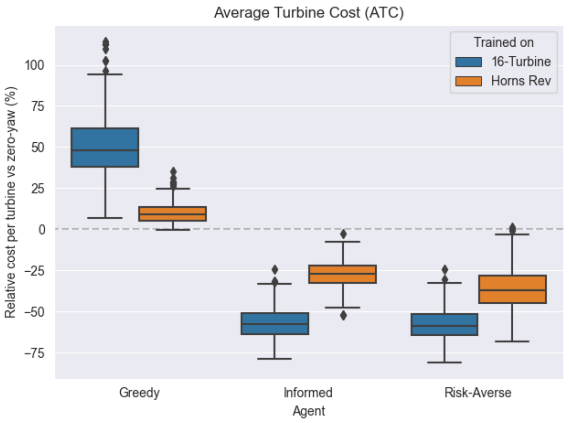

Figure 7.39: Per-turbine average cost: the transferred 16-turbine agent vs the Horns Rev agent.

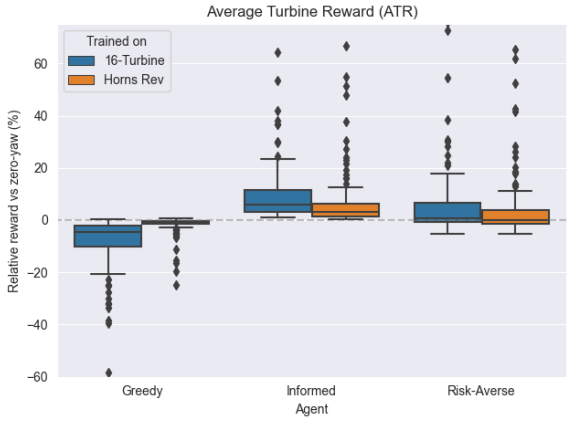

Figure 7.40: Per-turbine average reward: the transferred 16-turbine agent vs the Horns Rev agent.

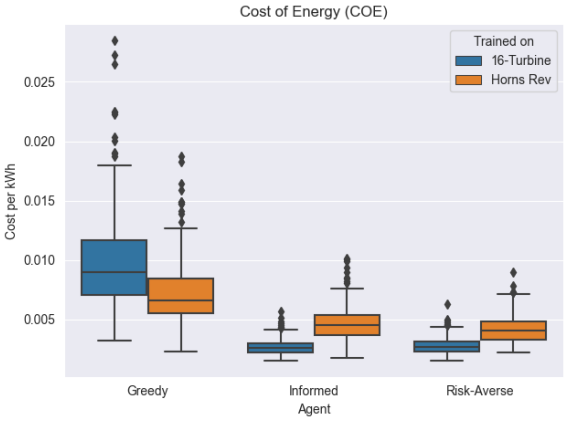

Figure 7.41: Average cost of energy: the transferred 16-turbine agent vs the Horns Rev agent.

which is not as dependent on farm topology. It is thus evident that training on random layouts mainly helps the power optimisation agent generalise better. However, as evident from the data, due to the slightly worse cost minimisation, the overall revenue of the informed policy suffers.

7.3.2.5. Comparison with literature

In this subsection, the performance of the agents constructed and trained in this thesis will be compared with the available literature on wake steering and wind farms. This should provide an overview of whether the produced results align with earlier work and whether they agree on the gains that wake steering can bring. Since the majority of wake steering literature, of which many are covered in section 2.4, solely considers power optimisation, the Greedy agent will first be compared and contrasted with previous results.

Firstly, it is essential to note that many of the covered papers in the literature review only considered a very select few wind directions, often only one, which presents the worst-case wake scenario(s) for the farm. The performance increases quoted in these works are thus only valid for the specific condition under consideration, which might only occur during a fraction of the year. Anyhow, all works that explicitly presented statistics on wake steering performance are shown in Table 7.2. Looking at the statistics valid for *selected wind directions* (which can be assumed to be the peak performances for wake steering), the results seem to align well with results found in Figure 7.17. The greedy agent trained in this thesis was, under conditions that saw aligned turbines and therefore presented the worst conditions wake-wise, able to see power optimisations in the range of 20%. This aligns well with work by Dong et al. (2021) and Zong and Porté-Agel (2021). Furthermore, looking at works that quote annual performance increases like Howland et al. (2019), Gebraad et al. (2017) and Zong and Porté-Agel

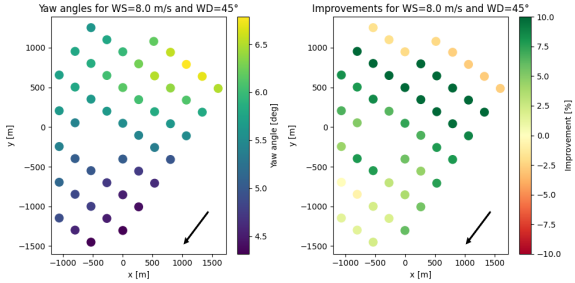

Figure 7.42: Yaw angles and improvements relative to baseline for the transferred greedy agent on the Lillgrund farm with a wind direction of 45 degrees (aligned with gridlines).

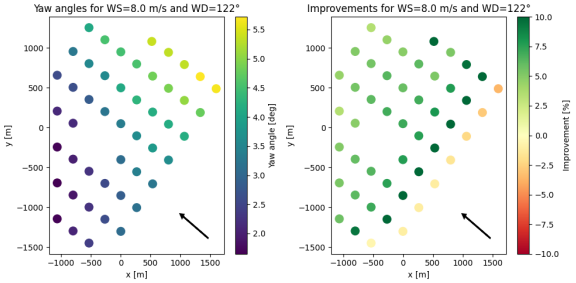

Figure 7.43: Yaw angles and improvements relative to baseline for the transferred greedy agent on the Lillgrund farm with a wind direction of 122 degrees (aligned with gridlines).

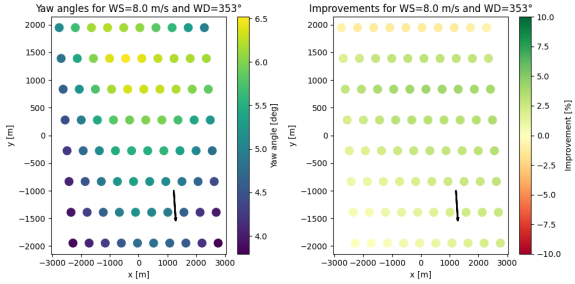

Figure 7.44: Yaw angles and improvements relative to baseline for the transferred greedy agent on the Horns Rev farm with a wind direction of 353 degrees (aligned with gridlines).

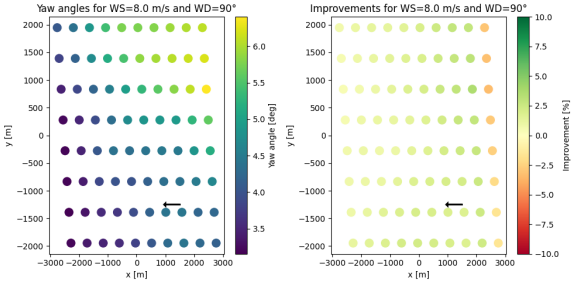

Figure 7.45: Yaw angles and improvements relative to baseline for the transferred greedy agent on the Horns Rev farm with a wind direction of 90 degrees (aligned with gridlines).

(2021), this thesis' results of 1.5% AEP improvement on average agree very well.

To investigate how realistic the Cost Of Energy (COE) metrics in Figure 7.10, Figure 7.28 and Figure 7.33 are, they can be compared with some real-life numbers. In research by NREL regarding the cost of energy of wind energy in 2022 (Stehly et al., 2023), they found that typical commercial-scale wind turbines in large-scale distributed projects have a COE of around 78 dollars per MWh, of which 11.8 can be attributed to operational expenditure (OpEx). Converted into EUR per kWh, this is a COE of 0.0108 EUR/kWh. This is very close to the values that followed from the evaluation runs performed in this chapter. The calculated values here are slightly lower, but that can be explained by 1) the fact that these numbers only reflect the pure component cost for replacements, and 2) the fact that they are optimised by the Informed and Risk-Averse agents. Any other operation or maintenance costs have not yet been considered, which also contributes to the discrepency. All in all, the values are in the same order of magnitude and are, therefore, seemingly realistic.

7.3.3. Finite Horizon

As mentioned before, the finite horizon case significantly differs from the infinite horizon case. Optimising for, say, a 20-year wind farm lifetime under realistic costs brings a completely different optimisation problem compared to finding the best trade-off between component utilisation and power production. This can be explained by the fact that moving from one component to a new replacement will immediately incur the complete replacement cost. In contrast, the dense cost model will only penalise based on usage. In the realistic case, components are paid for in advance rather than based on usage, and thus, there is a sizeable difference between both cost models. It might, for example, be beneficial to ensure components fail just outside the considered time horizon, optimising power as much as possible without causing an additional replacement due to the yawing behaviour. Cost can thus only be modelled using the sparse cost model, ensuring the step-wise costs allow for strategic influencing of component lifetime.

It is thus quite interesting to investigate how certain policies perform compared to each other in these more realistic conditions. The agents trained in the previous section, e.g. the 'infinite horizon'-agents, can be evaluated on a sparse-cost finite-horizon environment. Each agent is evaluated on the

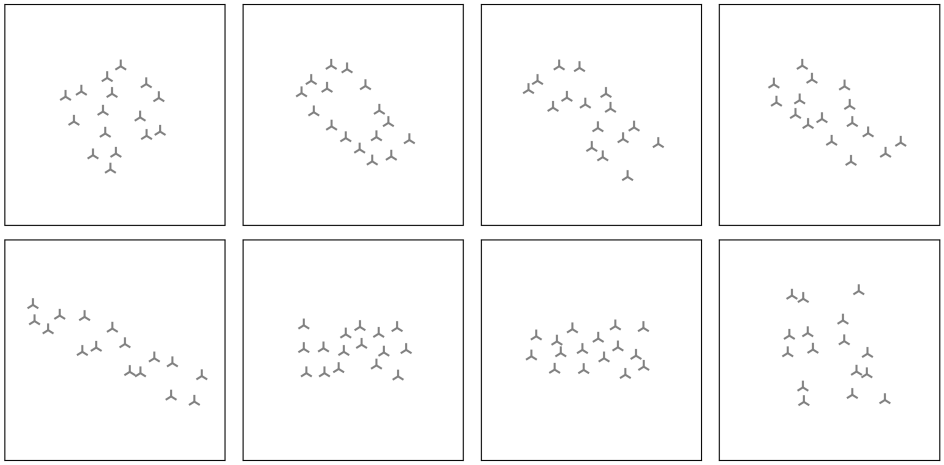

Figure 7.46: Samples of random 16-turbine layouts.

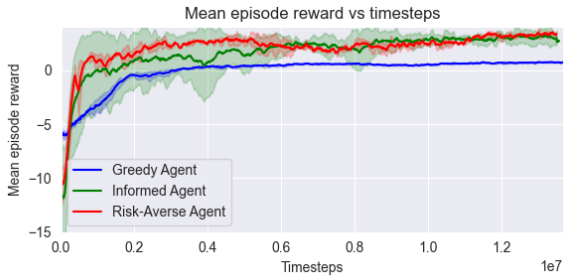

Figure 7.47: Training curves of the agents trained on random 16-turbine layouts

16-turbine farm for a total period of 40 years, plotting their accumulated total balance of profits and costs as a function of time. Each agent is evaluated 20 times on the same 20 seeds, and the mean of the 20 runs is plotted in Figure 7.53. Below the figures is a timeline indicating which of the agents would obtain the highest cumulative profit *if* the wind farm were only to be considered up until that year. Furthermore, the right plot subtracts the cumulative profit of the baseline 'zero-yaw' agent from all curves.

As would be expected, the greedy agent quickly surpasses all other agents due to the immediate effect of power optimisation which is reflected in the resulting profit. Since the impact of fatigue degradation is not yet felt, the greedy agent quickly finds a sizeable lead compared to the others. However, at around year 10, the components start to fail one by one due to the increased loads caused by the wake steering actions. Here, it quickly loses its lead and eventually drops below all others. The informed agent, though initially losing out on profit due to the conservative strategy, enjoys longer component lifetimes and thus lower costs compared to the greedy agent and baseline. Especially around the time the baseline's components start reaching their 22-year design lifetime, the informed agent sees a large lead over the baseline policy. Interestingly, once the baseline has caught up with the greedy agent in the first 'cycle' of breakdowns, the greedy agent sees another lead over the baseline agent. This is due to both policies having experienced a full breakdown of all components, but the greedy agent having produced more energy in the meantime. This lead then quickly disappears again as another cycle of components starts to break down.

The risk-averse agent loses out on profit from day one due to a significantly lower power production. It is never the most optimal policy to choose as it never surpasses the other policies, at least not within the 40 years under analysis. However, should the wind farm be in operation for long enough, the risk-averse policy will eventually lag behind in enough breakdowns to catch up to all the other policies except for the informed policy. This is due to the fact that the risk-averse policy is optimal for *infinite horizon* cases, where the difference between dense and sparse costs can be considered negligible. The same goes for the informed policy: eventually, it will lag behind in failures, which will end up making enough

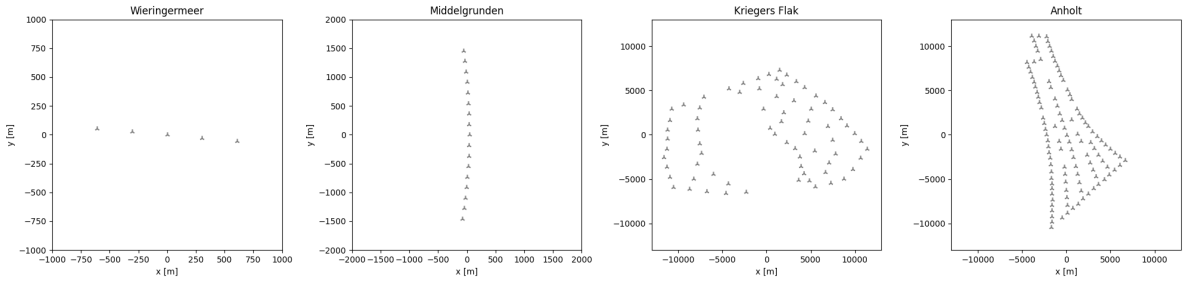

Figure 7.48: Farm layouts to test with the 16T and 16R agents.

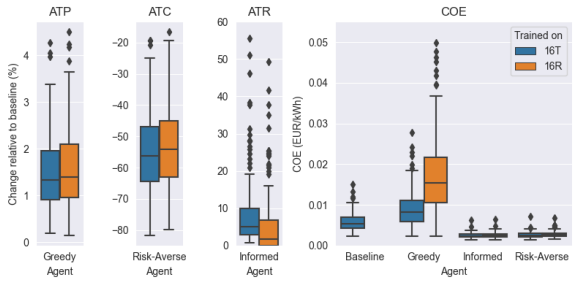

Figure 7.49: 16T vs 16R agents on Wieringermeer.

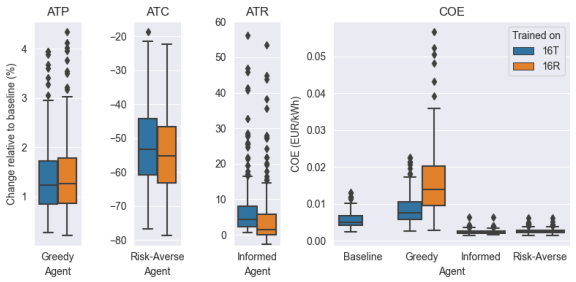

Figure 7.50: 16T vs 16R agents on Middelgrunden.

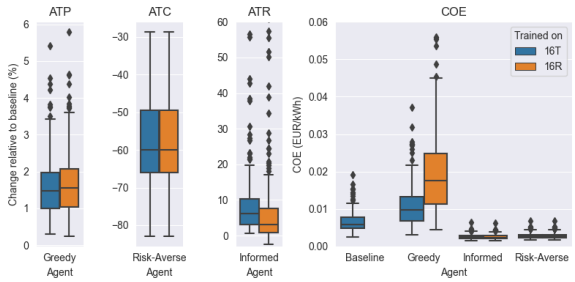

Figure 7.51: 16T vs 16R agents on Kriegers Flak.

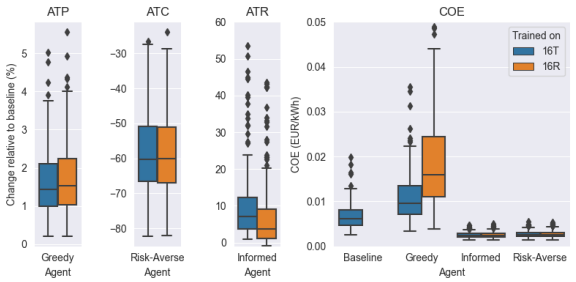

Figure 7.52: 16T vs 16R agents on Anholt.

of an impact to counteract the power loss due to sub-optimal yaw misalignments.

Looking at the timeline of which policies are optimal given what time horizon, it is evident that there is quite some diversity. As expected, initially, the greedy policy is optimal as all effects of fatigue and failure can essentially be ignored. However, the baseline policy quickly becomes the best policy for choosing between lifetimes of 9 to 22 years. At the 22-year mark, its own components start to break down, whereas the informed policy manages to yet prevent this from happening. In fact, it can go all the way to approximately 34 years without having accumulated enough failures to once again drop below the baseline performance. Essentially, in the region just beyond the design lifetime of the turbine components, a policy that preserves components can manage to postpone failure costs as much as possible and thereby maximise profit. This will not always be optimal, though; an example is the risk-averse policy, which also minimises damage but is never the optimal policy. This is because it does not consider power production and, therefore, loses out on too much energy production. However, in *very long term*, both risk-averse and informed policies will once again surpass all other policies once they lag behind in enough failures. Ultimately, the informed policy will find the best balance between profits and costs and thereby maximise wind farm net profit.

Paper	Power Increase (approx.)	Random Wind	Farm Size
Howland et al. (2019)	7% (0.3%[1])	≈	6
Fleming et al. (2019)	4%[2]	√	5
Zong and Porté-Agel (2021)	15.7%[2] (1.8%[1])	√	3-80
Gebraad et al. (2016)	13%[2]	×	6
Gebraad et al. (2017)	3.7%[1]	√	60
Bui et al. (2020)	2-4%[2,3]	×	15
Dong et al. (2021)	15%[2]	×	6
Kadoche et al. (2023)	1-6%[2,3]	≈	4-151

[1]annual average, [2]on selected wind directions, [3]depending on layout

Table 7.2: Power increases of 'greedy' agents trained in literature.

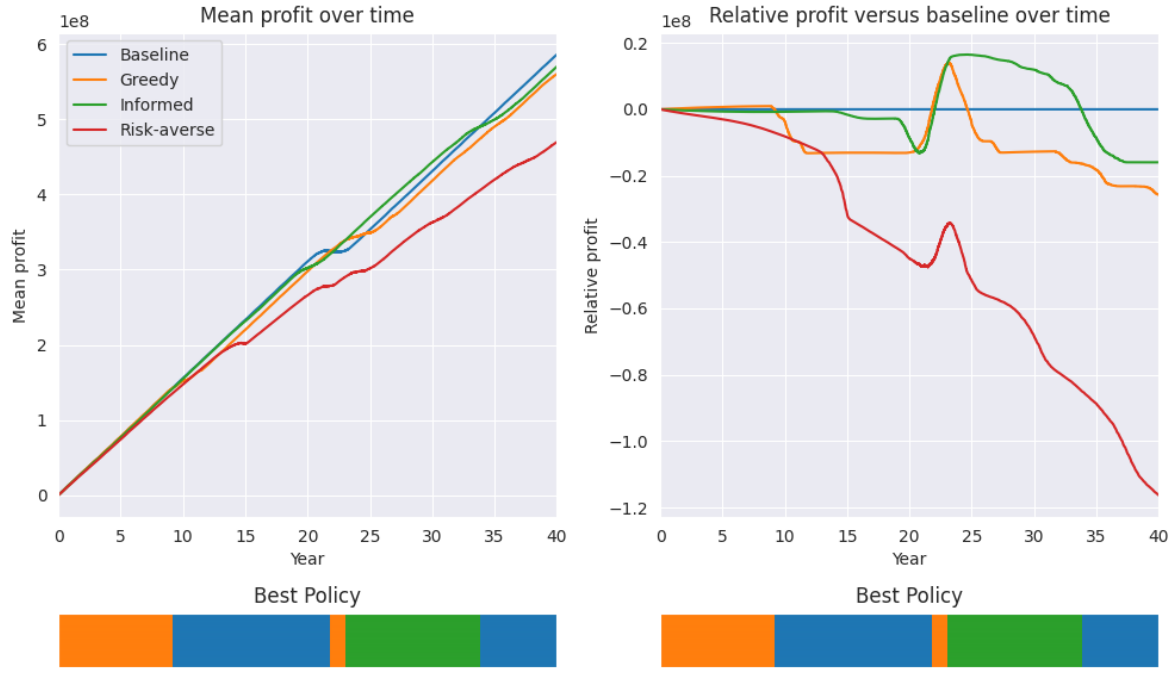

Figure 7.53: 40-year analysis of the cumulative profits generated by each policy.

8

Discussion

Following the work in previous chapters, this chapter presents a reflection and discussion on the obtained results in section 8.1. Furthermore, several future research opportunities are presented in section 8.2 that can extend the research conducted in this thesis.

8.1. Reflection & Limitations

Optimality Judging from the agents' behaviour, the yaw angles as a function of wind direction seem to be relatively smooth lines. Furthermore, the farm-wide behaviour that they show regarding yaw angles throughout the farm does not perfectly align with results from papers like that from Zong and Porté-Agel (2021). There, they find that, typically, a perfect gradient of yaw angles in the stream-wise direction going downstream exists. The results in this work, however, show that this generally is the case to a certain degree, but not always. In fact, some wind directions show that certain areas of the farm generally exhibit higher yaw angles. The question thus becomes whether the learned policy is, in fact, optimal or whether it is some local optimum. What likely seems to be the case is that, due to there being so many input parameters and layouts it has to generalise, it has to average out in some areas due to limits in the expressivity of the deep learning architecture. With the number of changing wind conditions, electricity prices, relative turbine positions and wake effects depending on the wind direction, it seems very hard to generalise to every condition while maintaining optimality. Generalising over such a large domain inevitably means compromising on optimality under specific environmental conditions.

Though each of the agents does see significant improvements in their relative expertise, even when averaging out over all the input parameters, it is impossible to judge whether they have truly achieved optimal behaviour. In the previous chapter, the results obtained were compared to those in the literature, which showed comparable results despite using a different technique and using more randomised and realistic environmental conditions. The results are promising regardless of this optimality, but knowing whether a better policy can be learnt can help to determine their relative performance better. One way of doing this could be to use a gradient descent (or any other optimisation algorithm) to perform one-step optimisation for the same conditions as the agents and compare their improvements relative to the baseline. Given enough time, the optimisation algorithm should be able to find the most optimal action given the current state, which can be assumed to be an upper limit of performance. This is only possible due to the fact that the infinite horizon problem at hand can be further simplified towards a single-step optimisation problem. Making this comparison should help shed some light on the true performance of the trained agents and allow assessment of RL performance.

Loads & Wear The yawing action itself (e.g. the physical movement of the yaw system) can carry a form of degradation that is not yet modelled in the current environment. Active yaw control for wake steering requires constant yaw angle adjustment given the changing flow conditions, meaning constant wear on parts of the yaw system itself. Right now only the fatigue loads on the yaw system are modelled as a result of loads on the tower, but the yawing movement itself might introduce additional fatigue or even simply wearing down of bearings and/or gears. Traditional wind farms might maintain some

yaw angle set-point based on averaged wind conditions to minimise yaw movement, and as such, this thesis' yaw control policies might cause premature failure of certain components. Furthermore, the environment assumes that any yaw angle can be reached within the timestep, e.g. there is no constraint on the speed at which the yaw angle changes between timesteps. Typically, there might be some maximum rated rate or rotation which must be respected and can, therefore, not be exceeded. Introducing both of these factors (yaw wear and yaw constraints) to the environment can cause the agent to take drastically different actions and exhibit much more conservative behaviour. Minimising yaw wear can encourage the agent to choose slightly suboptimal yaw angles if that ensures less movement is required in total, and constraints on yaw speed might cause the agent to anticipate changes in wind direction. This would play well with the temporally coherent wind conditions already present in the environment.

Continuing on the topic of non-modelled sources of wear, the blades do not account for typical types of erosion, such as that caused by wind or rain. Fatigue failure is not the only cause of premature blade failures in wind turbines; leading edge erosion also sees a large portion of failure cases (Mishnaevsky, 2022) and cannot be ignored in a complete simulation. The combined effects of erosion and fatigue cracking might result in an even faster degradation of the blades during operation. It is effectively reasonably unrealistic to assume the component lifetimes are only dependent on fatigue loads due to yaw control. In contrast, wake steering might also have an effect on the rate of leading edge erosion. Adding this to the environment should help make it more realistic with regard to the impact caused by wake steering.

Furthermore, in chapter 4, the assumption was made that the tower loads (e.g. fore-aft and side-to-side) could be treated as separate components with individual damage accumulations. Of course, in real life, these two loads are only truly defined in the rotating frame of reference of the nacelle. The full tower itself does not rotate as the nacelle does; thus, the fore-aft and side-to-side loads will act in a rotating frame compared to the tower base. This can be made more realistic by either 1) projecting the determined loads back into two separate tower-base fixed load directions or 2) combining the two loads into a single load acting on a single component 'tower base'. In the first case, both directional tower-base fixed components would have identical fatigue parameters, whereas in the second case, only a single 'component' needs tuning. Both options for combining tower base loads are shown in Figure 8.1. It is important to note that this does not apply to the blade root, as the frame of reference for those loads is always the same regardless of the nacelle's yaw angle. Making this change to the tower-base loads could potentially change the agent's behaviour with regard to minimising tower damage, possibly allowing it to act more aggressively towards power optimisation.

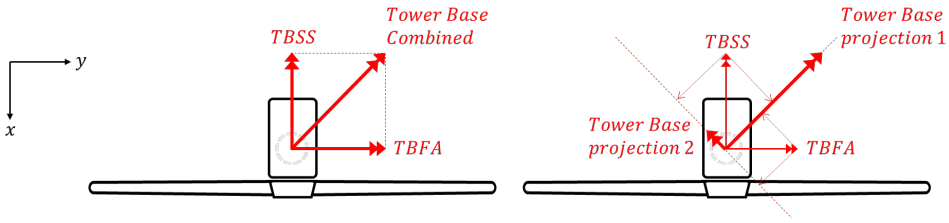

Figure 8.1: Methods of adjusting tower loads: combining (left) and projecting into a world-fixed frame of reference (right).

Discontinuity in Wind Direction In the observation given to the agent, the current wind direction is presented directly as an angle. The same applies to the relative angles encoded in the edges between the nodes of the graph. Consequently, there exists a discontinuity in the behaviour exhibited by the agents between 0 and 359 degrees, despite them being close in a polar sense. This behaviour seems unlogical and somewhat strange, as neighbouring wind directions should cause similar behaviour from the agents. In order to tackle this, the wind directions and angles provided to the agent should inherently already include the cyclic characteristic of directions. This can be done by encoding the wind direction (and any other direction sense) as its corresponding sine and cosine values, which are already cyclic in nature. Any policy the agent learns will, therefore, also become smooth and cyclic with respect to the wind direction input. Consequently, any of the observed discontinuities will likely disappear, and the behaviour at zero degrees will be (nearly) identical to the behaviour at 359 degrees.

8.2. Future Work

Finite-Horizon Optimisation In subsection 7.3.3, the implications of using each trained policy on a finite-horizon problem were investigated. It became evident that certain policies would be optimal for certain fixed horizons, meaning there was no one best policy that would be, economically speaking, the best choice. This was a result of the combination of dense profits (e.g. power production delivering immediate profits) and sparse costs (e.g. replacement costs needing payment solely when a failure occurs). The sparse cost aspect allows the agent to employ a strategic policy that accounts for the long-term sparse effects that result from yawing behaviour throughout its lifetime. For example, using a slightly more conservative yawing policy might ensure components will not end up breaking before the finite horizon is reached. However, the optimal policy could also be using the most aggressive wake steering behaviour for power optimisation if that means the increased profit exceeds the additional replacement costs that result from it. It is thus safe to assume that an optimal policy exists that maximises the total net profit for a given time horizon, finding the perfect trade-off between power maximisation and component breakdowns. This is effectively different from the dense-cost, infinite-horizon case as costs are no longer pay-per-use but pay-ahead.

Future work could thus include the optimisation for a given time horizon, potentially given some initial state. This could be applicable in cases where the lifetime of a wind farm is known beforehand, say 25 years, and the operator seeks to maximise the net profit over those 25 years. This effectively comes down to finding the control policy that considers both short-term effects (immediate power production optimisation) and long-term effects (postponing breakdowns or increasing them if the resulting power benefit exceeds the costs). What's more is that an operator that has been using a wind farm for, say, 25 years can choose to find a policy that optimises for the final lifetime extension of 5 years, given an estimate of the current fatigue damage states of all turbines. In either case, it is unlikely a 'default' policy such as Greedy or Informed will yield the best net profit; instead, an agent must be trained to optimise for the given time horizon and initial condition specifically. Furthermore, it is no longer solvable with single-step optimisation methods, as optimising for each timestep will ignore the long-term effects if sparse costs are employed. Power maximisation might no longer be the most optimal policy.

However, one problem that will arise with long time horizons of, say, 20 years is the vast time difference between cause and effect. Fatigue degradation happens throughout the entire turbine lifetime and is a result of all actions taken up until the point of failure. It can be tough for an agent to learn that actions taken at timestep one can have an effect on the costs suddenly incurred by 1 million 10-minute timesteps later in year 20. This introduces a large *credit assignment* problem. One way learning can possibly be made slightly easier is to make use of *curriculum learning*, making the problem harder as training progresses. For this, the environment can start with any of the two exponential cost functions with a parameter c that causes the damage distribution to be evenly distributed. As training progresses, the environment can then shift the costs towards the end of the component lifetimes by increasing c. Eventually, the agent should learn that higher damage states cause higher costs, thereby guiding the agent from the simpler problem (dense costs) to the final problem (sparse costs). Another solution might come in the form of potential-based reward shaping Ng et al. (1999) or reward redistribution of sparse and delayed rewards using RUDDER (Arjona-Medina et al., 2019). Furthermore, there is a clear clash of interests between what is, in effect, immediate rewards of profit through power maximisation and delayed penalties due to component degradation. On the one hand, the 'tactical' part of the agent will seek to optimise immediate rewards, whereas the 'strategic' part of the agent will seek to explicitly consider when the relatively large replacement costs will occur. Both of these aspects, being the extremely long time horizon and the clash between tactical and strategic decision-making, make the problem very hard to solve.

One solution to this problem might come in the form of Hierarchical Reinforcement Learning (HRL). HRL decomposes a problem into smaller sub-problems which are then composed together using a composition strategy. More specifically, in this case, the *temporal abstraction* techniques of HRL can be leveraged to decompose the complete problem into a *tactical* agent and a *strategic* level. The tactical level has short-term goals in maximising power output, as that provides immediate positive reward. The strategic level, however, has long-term goals and looks at the long-term effects of component degradation. Using HRL, both hierarchical levels can then be merged together to optimise for the combined goals. In the field of multi-agent reinforcement learning, HRL has already seen applications in cooperative tasks (Ghavamzadeh et al., 2006). Furthermore, recently, Tang et al. (2019) investigated the use of HRL with temporal abstractions to tackle multi-agent cooperative systems with sparse and

delayed rewards. They decompose the problem into a hierarchy of two time scales, where the higher level considers long-term goals and the lower level attempts to reach these goals. In the context of the wind farm environment, the high-level metacontroller outputs sub-goals to the lower-level controller, e.g. to use conservative, aggressive or balanced wake steering strategies. It makes these decisions based on accumulated damage or projected costs. The lower-level controller then acts based on the provided goals, applying yaw actions accordingly.

There remain some more difficulties in finite-horizon optimisation. One of them is the baseline removal, for which a different technique must be invented. In the case of the infinite horizon problem with dense costs, the cost (and therefore reward) of the baseline policy is immediately known and can be inferred directly. However, in the case of a finite horizon with sparse costs, this cost cannot be directly inferred. There is no way to infer from a single timestep what cost should be associated with its behaviour. This problem is not present for the power production part, as nothing changes with regard to the baseline removal in the finite horizon case. Still, baseline removal needs a technique to infer the cost the baseline policy would incur at each timestep in order to work effectively. It cannot be omitted either, as performing baseline removal on power but not on damage introduces a bias to the reward, as power becomes *relative* and damage *absolute*. Neither can a separate damage state D be maintained for the baseline policy as the episode progresses, as it then no longer becomes a *state-dependent* baseline because it also depends on all the states preceding it. The difficulty is, therefore, in defining what baseline cost to subtract from the reward.

Observability In the definition of the environment and its observation space in section 5.8, the assumption was made that the environment is fully observable. This means that all variables that are provided to the agent for its decision-making process are perfectly known and can be measured as such. For most of the variables, this is a reasonable assumption to make, for example, for the wind speed or wind direction. However, it is unrealistic to assume these can be observed for other variables like the damage states D. In fact, they are not even a measurable physical variable but rather a virtual approximation of the fatigue state. Yet, the agent observes this variable in its inference process. The environment should thus not provide the damage states D directly and explicitly as part of the observation. For the infinite-horizon agents, this will not directly have any impact as they are invariant to it. Still, finite-horizon optimisation, as discussed in section 8.2, will need some observation of the current damage state to operate strategically. This can be solved by casting the Fully Observable Markov Decision Process into a Partially Observable Markov Decision Process (POMDP).

In a POMDP, not all environment states can be directly observed by the agent. Instead, the agent makes other observations from which the true system state can potentially be inferred. In fact, the agent constructs a belief state $b(s_t)$ of the environment based on its best guess of the true system state. Since it is reasonable to assume that all system states except for the damage state D can be observed with high accuracy, only D needs belief state inference. For this, the agent requires a variable which can be measured to some extent and from which the damage state can be inferred over time. It is reasonable to assume that the fatigue damages (DELs) can be approximated, calculated or measured to some extent. For example, strain sensors can be placed on the blades, or a numerical model can be used to calculate structural loads based on the local flow conditions. In the environment, this can be emulated by constructing a probability distribution around the 'true' calculated DELs, sampling from that and providing that as an observation for the agent to construct its belief state. In the environment, the calculated DELs are assumed to be the 'ground truth' values as they would exactly be in a real-life system. The sampling from the distributions would be the measurement or approximation uncertainty of the tools used in practice. The agent can then, for example using an LSTM model, determine the accumulated damage state D over time. If the fatigue parameters are unknown, the LSTM model can be trained on the load history of other turbines which have operated till failure before. It can then learn to approximate the damage accumulation based on the provided measurements of the DELs.

Using this, the agent obtains a combination of state variables (e.g. wind conditions, electricity price, etc.), and the belief states $b(s_t)$. Since the belief state will include some uncertainty as it remains an approximation, the agent will then learn to deal with this uncertainty whilst maximising the wind farm profit. The problem, now a POMDP, is summarised in the diagram shown in Figure 8.2. Observations z_t from the environment are used to infer a belief state b_t, which is kept over time; b_t is used to decide on an action a_t, which is fed back to the environment. Rather than using the state s_t directly (as was done before), only the observations z_t are available to the agent. Furthermore, z_t can include the fatigue

loads that are required to infer the belief state b_t and the other variables which were fully observable to begin with.

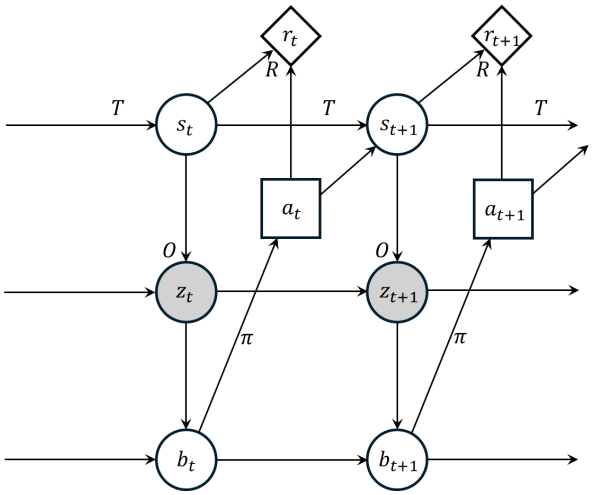

Figure 8.2: Diagram of the Partially Observable Markov Decision Process

Gradient-Based optimisation An interesting property of the farm-level surrogate used in the environment is that it provides a fully differentiable function for the wind farm simulation. Using PyTorch's Autograd capabilities, the gradient of the output with respect to the inputs is known directly. What this effectively means is that the gradient of power production with respect to the input yaw angles is known, and gradient descent can be used directly without a point-based approximation of the local gradient. Gradient descent can be used on the model directly to find the instantaneous optimal yaw angle distribution throughout the wind farm with relatively little computational effort compared to gradient descent methods that have to approximate the gradient at each step. Furthermore, both risk-averse and informed optimisation can too be solved in a similar way by using PyTorch to calculate the derivative of the full objective function with respect to the input parameters. It can, therefore, be interesting to investigate the performance of gradient-based optimisation and the performance of the trained RL agents to examine whether they are able to achieve similar results. Furthermore, it allows to investigate whether the RL agents that are transferred from smaller farms to bigger farms through the generalizability property are anywhere close to optimal behaviour in the larger farm or not.

Furthermore, the GNN-based surrogates themselves can be used outside of the environment for set point optimisation in real-life wind farms. In fact, the pre-trained surrogates can be fine-tuned using real-life measured data from wind farms to make their outputs match reality as closely as possible. Next, gradient-based optimisation can be used to directly optimise the yaw angles in the wind farm. Quite possibly the observed measurements as a result of its behaviour can then be used to update the surrogate itself, making it more accurate in the progress. It is then possible to create a wake steering controller that can make itself more accurate as time progresses. Important to note, however, that this is indeed single-step optimisation and does not include the limited-horizon optimality that would be found using the limited-horizon controller.

8.2.1. Decentralised Learning

As became evident when moving from the 16-turbine farm to the 48-turbine Lillgrund and 80-turbine Horns Rev farm, the fully centralised learning paradigm runs into scalability issues. The exploding joint action space of all agents combined means the agent has trouble exploring and converging to an optimal policy. This could be partially overcome by training on small farms, where proper convergence is known to happen, and transferring the obtained agents to larger farms. Despite their impressive generalizability, the achieved performance on the larger farms, which was almost as good as on the 16-turbine farm but not quite, indicated that more performance could still be extracted. Effectively, this means that training on the large farms is likely the only way of truly extracting optimal performance from each of the agents. Given the scalability constraints of centralised learning, a different training

paradigm is necessary.

This is where decentralised multi-agent learning could offer a solution. By training using the CTDE scheme, information sharing and collaborative training are still achieved whilst keeping action spaces low for each agent. To still benefit from the graph-based architecture and explicitly encoded topological relations, a shared state encoder can be shared across agents. More specifically, the state-encoder section of the agent, including the encode and process steps, performs initial processing, after which each agent branches off individually. Information sharing can be achieved in two ways: on the one hand, through the shared encoder, and on the other hand, through a shared critic network. As such, only a single critic network head branches off the state encoder, whereas each agent branches off with their own policy head. The architecture would look like what is depicted in Figure 8.3. Since each agent is no longer working with a large joint action space, exploration should become simpler and the algorithm could converge to a solution (faster).

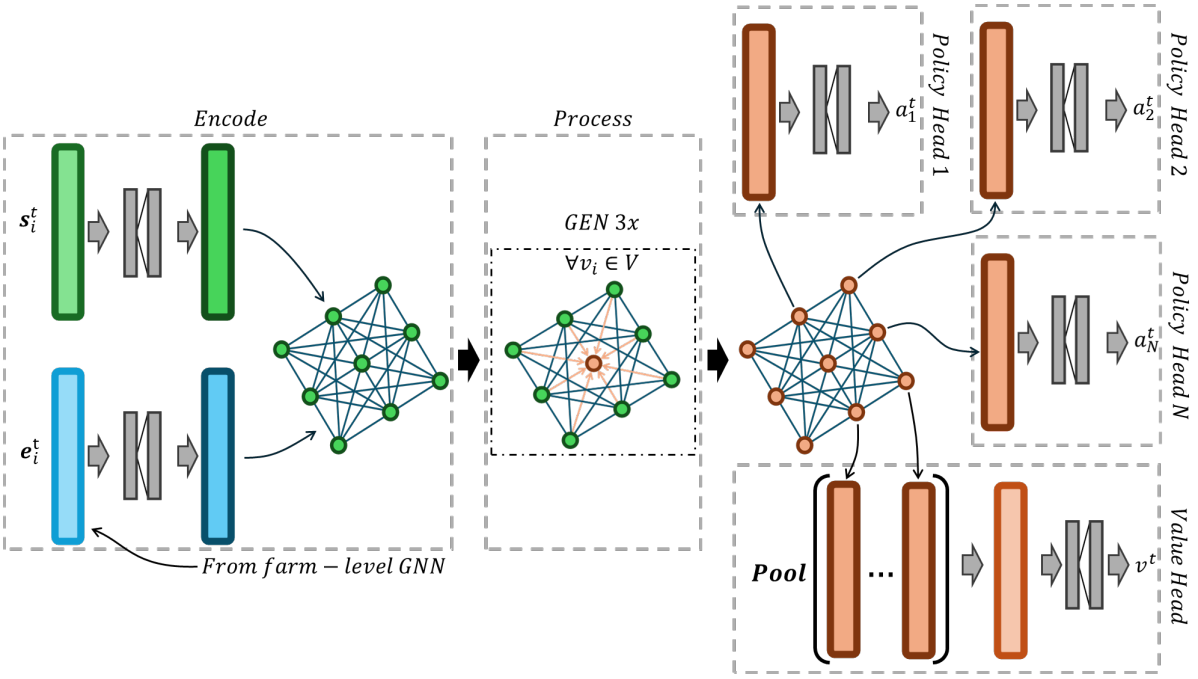

Figure 8.3: CTDE architecture suggestion for training on large farms.

8.2.2. Combined Pitch & Yaw Control

Yaw control for wake steering, as covered in this thesis, is not the only way of mitigating the effects of wakes in wind farms. Other methods include the use of Pitch Control (Lee et al., 2013; Dilip and Porté-Agel, 2017; Frederik et al., 2020) and Tilt Control (Fleming et al., 2014; Weipao et al., 2016). Pitch control, which is easily implemented due to the pitching actuators already present in most turbines, could be combined with yaw control to achieve an even higher degree of wake-mitigating action. It is thus worthwhile to study whether a wind farm control agent can make use of both pitch and yaw control to obtain even better farm-wide performance increases. For this, the turbine-level surrogates need adjusting such that they support pitch angle as input, and consequently the OpenFAST simulations need to be run without an active controller. After adjusting the farm-level surrogate to include pitch angles for all turbines and adding pitch angles in the action space, the RL agent can learn to jointly optimise yaw and pitch angles throughout the farm. This would effectively mean that the turbine pitch control works in harmony with the yaw control to achieve both the best local and global performance. Certain combinations of pitch and yaw angles may allow for both a load decrease, as well as wake deflection. Consequently, the trade-off between power and loads changes and entirely different behaviour might emerge.

9

Conclusion

This thesis investigated the applicability of graph-based deep reinforcement learning to train maintenance-conditioned wake steering controllers. In the first part of the thesis, a *fast*, *realistic* and *accurate* wind farm simulation environment was constructed. At the heart of this environment is a fast GNN-based wind farm simulation surrogate model, building on the work by Duthé et al. (2023), that infers both wake-affected farm-level and turbine-level effects as a result of the control policy and free-stream wind conditions. The environment was then augmented with a synthetic wind condition generator based on Discrete Markov Chains, an electricity price model, a fatigue degradation model based on Palmgren-Miner's linear damage accumulation rule, and a maintenance model to determine the costs of component degradation. Together, all these components make up the wind farm simulation environment that models both wake-affected power production and turbine loads. A 10-minute timestep was chosen, and the assumption was made that all variables in the environment were kept constant within the timestep. The action space consists of the yaw angles for each turbine, and the graph-embedded observation space includes wind conditions, electricity price, calendar time and damages for each turbine.

Reinforcement learning is done using the Centralized Training with Centralized Execution (CTCE) framework, using Proximal Policy Optimisation (PPO) and a graph-based agent architecture. Additionally, state-dependent baseline removal was added to tackle the high degree of variance in the environment. Stable learning was further facilitated by action and observation normalisation as well as reward scaling. An ablation study pointed out that especially baseline removal was a necessary requirement for stable learning, but all other techniques did help in performance increases too. Next, five different agents were defined: the zero-yaw baseline (do-nothing) and random policies, and the Greedy, Risk-averse and Informed policies for power, damage and combined optimisation.

Training started on the 16-turbine wind farm using an infinite horizon assumption, showing good convergence of all three Reinforcement Learning agents. The results showed that the greedy agent can improve power production by 1.5% on average compared to the zero-yaw policy, with peak improvements of up to 20%. However, due to increased maintenance costs, it experiences a large decrease in net revenue. Both informed and risk-averse agents decreased costs by as much as 50%. The informed agent, whilst doing so, also managed to keep power losses low compared to the zero-yaw policy. Both informed and risk-averse policies manage to reduce the cost of energy significantly from 0.0061 to 0.0025 EUR/kWh. However, the informed agent produces more energy overall, ultimately yielding the highest total revenue at an increase of up to 20% compared to zero-yaw. Additionally, it is evident that each policy sees wildly different lifetimes for each component, indicating different behaviours are used to achieve different objectives. When analysing the greedy policies' performance under different wind directions, clear increases in the cardinal and diagonal directions are visible. This is logical, given the largest adverse effects of wakes under these conditions. Interestingly, the directions of the largest improvement are offset from the cardinal and diagonal directions by a few degrees. This is due to full-wake conditions being most infeasible and therefore challenging to steer wakes in. Furthermore, higher wind speeds result in a decrease in wake steering power improvements due to 1) the diminishing effects of wake steering and 2) the high wind speeds available throughout the farm despite velocity deficits. When analysing the informed policy, it becomes evident that higher electricity prices cause the agent to choose higher yaws for better wake steering, as electricity gets of higher value.

When training on larger farms like Lillgrund (48 turbines) and Horns Rev (80 turbines), the fully centralised training technique starts to run into scalability issues. While the agents are still able to improve in their respective objectives compared to baseline, performance decreases as wind farm size increases. This is due to the difficulty of a centralised agent to explore the joint action space effectively and converge on an optimal solution. Instead, the 16-turbine controller was transferred to the Lillgrund and Horns Rev farms to inspect the generalizability of the trained controllers. Indeed, they seem to generalise very well to unseen farms, indicating the agent learns policies based on *relative positions* of turbines in a farm. The transferred 16-turbine agent, despite being trained on a smaller farm, surpasses the performance of the agents specifically trained on the larger farms. Finally, the 16-turbine agent was trained on randomised 16-turbine layouts to investigate whether this improved generalizability. As it turns out, by evaluating both 16-turbine agents on four new farms, the randomised-layout training scheme improves power production on unseen farms. The other agents - informed and risk-averse - however, see similar performance but due to decreased cost minimisation also a decrease in total revenue. Their cost of energy remains nearly identical.

A comparison of the agents' performance with results published in the literature shows that they align very well. Indeed, improvements quoted in the literature on Annual Energy Production (AEP) of 1.5 to 2.5% align very well with the average improvements that were found in this thesis. Additionally, the peak performance of the greedy agent (up to 20% power improvement) aligns very well with previous work on power optimisation studies in worst-case wake conditions. Finally, all policies were run for 40 simulation years to investigate which policy is optimal at which point. For the first ten years, the greedy agent achieves the highest wind farm profit by optimising power production. However, components quickly start to fail and the baseline policy comes out on top. After the baseline policies' components start failing as well, the informed agent, having preserved components, becomes the optimal policy. When considering the true infinite horizon, the informed agent will start to lag behind in component failures and permanently become the best policy of all. However, as most wind farms generally operate with shorter lifetimes, the infinite horizon assumption made for the environment does not hold as dense costs deviate from sparse costs significantly, requiring explicit sparse cost finite-horizon optimisation.

Some points of discussion are left to be addressed. For one, the optimality of the trained agents is unknown. Given the high degree of generalizability to various wind conditions, electricity prices and farm layouts would probably mean the agent has to compromise on some optimality. As for loads and wear in the environment, the yawing movement itself is not considered in the degradation model despite being an important source of component wear. This applies to other types of blade erosion too, which could change as part of the yawing policy. Furthermore, the tower base loads likely need adjustment to account for the mismatch between nacelle-fixed and world-fixed frames of reference. When it comes to observed policy behaviour, there seems to be a discontinuity in yaw angles between 0 and 359 degrees despite them being close together in a polar sense. This is unlogical but likely caused by the linear angle representation in observations and can be addressed by providing sine and cosine angle representations. Finally, some future work was proposed. This includes the aforementioned finite-horizon optimisation with which policies specific to certain horizon lengths can be trained to find the truly best policy that maximises profit. This type of optimisation has many difficulties, for which some potential solutions were provided. Next, the observability of the environment was discussed. Not all variables can realistically be assumed to be observable, such as the damage states D. Instead, casting the problem into a Partially Observable Markov Decision Process (POMDP) was proposed as a solution. Furthermore, gradient-based yaw angle optimisation is proposed as the derivative of the simulation model can be computed directly. Additionally, to tackle the scalability issue of fully centralized training, a decentralised training approach with a shared state encoder is proposed as well. Finally, the idea of combined pitch and yaw control for more wake steering authority is discussed.

To conclude, graph-based deep reinforcement learning can be used to make a wind condition-agnostic, layout-agnostic wake steering controller. This can be done for power optimisation, damage minimisation and combined optimisation as an objective. The informed policy is the ideal revenue-maximising policy in the long run but does not surpass the baseline policy for horizons shorter than the turbine component design lifetime. The generalisability of the greedy agent is improved by training on random 16-turbine layouts. Ultimately, the informed agent is best at maximising long-term wind farm net profit, considering both power production and maintenance costs. Such a controller can be used by wind farm operators to maximise revenue and provide lower energy prices to the customer. However, optimising for fixed time horizons with realistic sparse costs remains an open area of research.

Bibliography

Allaix, D.L., Gijsbers, F.B.J., 2016. Bayesian estimation of characteristic S-N curves for reinforcement bars and proposal for the national annex of NEN-EN1992-1-1. Technical Report.

Arjona-Medina, J.A., Gillhofer, M., Widrich, M., Unterthiner, T., Brandstetter, J., Hochreiter, S., 2019. RUDDER: Return Decomposition for Delayed Rewards. URL: http://arxiv.org/abs/1806.07857, http://dx.doi.org/10.48550/arXiv.1806.07857.

Barthelmie, R.J., Frandsen, S.T., Nielsen, M.N., Pryor, S.C., Rethore, P.E., Jørgensen, H.E., 2007. Modelling and measurements of power losses and turbulence intensity in wind turbine wakes at Middelgrunden offshore wind farm. Wind Energy 10, 517–528. http://dx.doi.org/10.1002/we.238.

Barthelmie, R.J., Hansen, K., Frandsen, S.T., Rathmann, O., Schepers, J.G., Schlez, W., Phillips, J., Rados, K., Zervos, A., Politis, E.S., Chaviaropoulos, P.K., 2009. Modelling and measuring flow and wind turbine wakes in large wind farms offshore. Wind Energy 12, 431–444. http://dx.doi.org/10.1002/we.348.

Bastankhah, M., Porté-Agel, F., 2014. A new analytical model for wind-turbine wakes. Renewable Energy 70, 116–123. http://dx.doi.org/10.1016/j.renene.2014.01.002.

Bossanyi, E., 2018. Combining induction control and wake steering for wind farm energy and fatigue loads optimisation. Journal of Physics: Conference Series 1037, 032011. http://dx.doi.org/10.1088/1742-6596/1037/3/032011.

Bui, V.H., Nguyen, T.T., Kim, H.M., 2020. Distributed Operation of Wind Farm for Maximizing Output Power: A Multi-Agent Deep Reinforcement Learning Approach. IEEE Access 8, 173136–173146. http://dx.doi.org/10.1109/ACCESS.2020.3022890.

Campagnolo, F., Imširović, L., Braunbehrens, R., Bottasso, C.L., 2022. Further calibration and validation of FLORIS with wind tunnel data. Journal of Physics: Conference Series 2265, 022019. http://dx.doi.org/10.1088/1742-6596/2265/2/022019.

Carroll, J., McDonald, A., McMillan, D., 2016. Failure rate, repair time and unscheduled O&M cost analysis of offshore wind turbines. Wind Energy 19, 1107–1119. http://dx.doi.org/10.1002/we.1887.

Churchfield, M.J., Lee, S., Moriarty, P.J., Hao, Y., Lackner, M.A., Barthelmie, R., Lundquist, J.K., Oxley, G., 2015. A Comparison of the Dynamic Wake Meandering Model, Large-Eddy Simulation, and Field Data at the Egmond aan Zee Offshore Wind Plant, in: 33rd Wind Energy Symposium, American Institute of Aeronautics and Astronautics. http://dx.doi.org/10.2514/6.2015-0724.

Crespo, A., Herna´ndez, J., 1996. Turbulence characteristics in wind-turbine wakes. Journal of Wind Engineering and Industrial Aerodynamics 61, 71–85. http://dx.doi.org/10.1016/0167-6105(95)00033-X.

Damiani, R., Dana, S., Annoni, J., Fleming, P., Roadman, J., van Dam, J., Dykes, K., 2018. Assessment of wind turbine component loads under yaw-offset conditions. Wind Energy Science 3, 173–189. http://dx.doi.org/10.5194/wes-3-173-2018.

Dao, C., Kazemtabrizi, B., Crabtree, C., 2019. Wind turbine reliability data review and impacts on levelised cost of energy. Wind Energy 22, 1848–1871. http://dx.doi.org/10.1002/we.2404.

De La Fuente, N., Guerra, D.A.V., 2024. A Comparative Study of Deep Reinforcement Learning Models: DQN vs PPO vs A2C. URL: http://arxiv.org/abs/2407.14151, http://dx.doi.org/10.48550/arXiv.2407.14151.

van Dijk, M.T., van Wingerden, J.W., Ashuri, T., Li, Y., 2017. Wind farm multi-objective wake redirection for optimizing power production and loads. Energy 121, 561–569. http://dx.doi.org/10.1016/j.energy.2017.01.051.

Dilip, D., Porté-Agel, F., 2017. Wind Turbine Wake Mitigation through Blade Pitch Offset. Energies 10, 757. http://dx.doi.org/10.3390/en10060757.

Dimitrov, N., 2019. Surrogate models for parameterized representation of wake-induced loads in wind farms. Wind Energy 22, 1371–1389. http://dx.doi.org/10.1002/we.2362.

Dimitrov, N., Kelly, M.C., Vignaroli, A., Berg, J., 2018. From wind to loads: wind turbine site-specific load estimation with surrogate models trained on high-fidelity load databases. Wind Energy Science 3, 767–790. http://dx.doi.org/10.5194/wes-3-767-2018.

Dong, H., Zhang, J., Zhao, X., 2021. Intelligent wind farm control via deep reinforcement learning and high-fidelity simulations. Applied Energy 292, 116928. http://dx.doi.org/10.1016/j.apenergy.2021.116928.

DTU Wind, Technical University of Denmark, . HAWC2. URL: https://www.hawc2.dk/.

Duthé, G., Santos, F.d.N., Abdallah, I., Réthore, P.□., Weijtjens, W., Chatzi, E., Devriendt, C., 2023. Local flow and loads estimation on wake-affected wind turbines using graph neural networks and PyWake. Journal of Physics: Conference Series 2505, 012014. http://dx.doi.org/10.1088/1742-6596/2505/1/012014.

El-Naggar, M.F., Abdelhamid, A.S., Elshahed, M.A., El-Shimy Mahmoud Bekhet, M., 2021. Ranking Subassemblies of Wind Energy Conversion Systems Concerning Their Impact on the Overall Reliability. IEEE Access 9, 53754–53768. http://dx.doi.org/10.1109/ACCESS.2021.3070533.

Energinet, . Home. URL: https://www.energidataservice.dk/tso-electricity/Elspotprices.

Ennis, B.L., White, J.R., Paquette, J.A., 2018. Wind turbine blade load characterization under yaw offset at the SWiFT facility. Journal of Physics: Conference Series 1037, 052001. http://dx.doi.org/10.1088/1742-6596/1037/5/052001.

Fan, W., Ma, Y., Li, Q., He, Y., Zhao, E., Tang, J., Yin, D., 2019. Graph Neural Networks for Social Recommendation, in: The World Wide Web Conference, Association for Computing Machinery, New York, NY, USA. pp. 417–426. http://dx.doi.org/10.1145/3308558.3313488.

Fischereit, J., Schaldemose Hansen, K., Larsén, X.G., van der Laan, M.P., Réthoré, P.E., Murcia Leon, J.P., 2022. Comparing and validating intra-farm and farm-to-farm wakes across different mesoscale and high-resolution wake models. Wind Energy Science 7, 1069–1091. http://dx.doi.org/10.5194/wes-7-1069-2022.

Fleming, P., Annoni, J., Shah, J.J., Wang, L., Ananthan, S., Zhang, Z., Hutchings, K., Wang, P., Chen, W., Chen, L., 2017. Field test of wake steering at an offshore wind farm. Wind Energy Science 2, 229–239. http://dx.doi.org/10.5194/wes-2-229-2017.

Fleming, P., Gebraad, P.M., Lee, S., van Wingerden, J.W., Johnson, K., Churchfield, M., Michalakes, J., Spalart, P., Moriarty, P., 2015. Simulation comparison of wake mitigation control strategies for a two-turbine case. Wind Energy 18, 2135–2143. http://dx.doi.org/10.1002/we.1810.

Fleming, P., King, J., Dykes, K., Simley, E., Roadman, J., Scholbrock, A., Murphy, P., Lundquist, J.K., Moriarty, P., Fleming, K., van Dam, J., Bay, C., Mudafort, R., Lopez, H., Skopek, J., Scott, M., Ryan, B., Guernsey, C., Brake, D., 2019. Initial results from a field campaign of wake steering applied at a commercial wind farm – Part 1. Wind Energy Science 4, 273–285. http://dx.doi.org/10.5194/wes-4-273-2019.

Fleming, P., Sinner, M., Young, T., Lannic, M., King, J., Simley, E., Doekemeijer, B., 2021. Experimental results of wake steering using fixed angles. Wind Energy Science 6, 1521–1531. http://dx.doi.org/10.5194/wes-6-1521-2021.

Fleming, P.A., Gebraad, P.M.O., Lee, S., van Wingerden, J.W., Johnson, K., Churchfield, M., Michalakes, J., Spalart, P., Moriarty, P., 2014. Evaluating techniques for redirecting turbine wakes using SOWFA. Renewable Energy 70, 211–218. http://dx.doi.org/10.1016/j.renene.2014.02.015.

Frederik, J.A., Doekemeijer, B.M., Mulders, S.P., van Wingerden, J.W., 2020. The helix approach: Using dynamic individual pitch control to enhance wake mixing in wind farms. Wind Energy 23, 1739–1751. http://dx.doi.org/10.1002/we.2513.

Freebury, G., Musial, W., 2000. Determining equivalent damage loading for full-scale wind turbine blade fatigue tests, in: 2000 ASME Wind Energy Symposium, American Institute of Aeronautics and Astronautics. http://dx.doi.org/10.2514/6.2000-50.

Gebraad, P., Thomas, J.J., Ning, A., Fleming, P., Dykes, K., 2017. Maximization of the annual energy production of wind power plants by optimization of layout and yaw-based wake control. Wind Energy 20, 97–107. http://dx.doi.org/10.1002/we.1993.

Gebraad, P.M.O., Teeuwisse, F.W., van Wingerden, J.W., Fleming, P.A., Ruben, S.D., Marden, J.R., Pao, L.Y., 2016. Wind plant power optimization through yaw control using a parametric model for wake effects—a CFD simulation study. Wind Energy 19, 95–114. http://dx.doi.org/10.1002/we.1822.

Ghavamzadeh, M., Mahadevan, S., Makar, R., 2006. Hierarchical multi-agent reinforcement learning. Autonomous Agents and Multi-Agent Systems 13, 197–229. http://dx.doi.org/10.1007/s10458-006-7035-4.

Gori, F., Laizet, S., Wynn, A., 2023. Sensitivity analysis of wake steering optimisation for wind farm power maximisation. Wind Energy Science 8, 1425–1451. http://dx.doi.org/10.5194/wes-8-1425-2023.

Greensmith, E., Bartlett, P., Baxter, J., 2001. Variance Reduction Techniques for Gradient Estimates in Reinforcement Learning, in: Advances in Neural Information Processing Systems, MIT Press.

Haghi, R., Crawford, C., 2023. Data-driven surrogate model for wind turbine damage equivalent load. Wind Energy Science Discussions , 1–34http://dx.doi.org/10.5194/wes-2023-157.

He, R., Yang, H., Lu, L., 2023. Optimal yaw strategy and fatigue analysis of wind turbines under the combined effects of wake and yaw control. Applied Energy 337, 120878. http://dx.doi.org/10.1016/j.apenergy.2023.120878.

Howland, M.F., Lele, S.K., Dabiri, J.O., 2019. Wind farm power optimization through wake steering. Proceedings of the National Academy of Sciences 116, 14495–14500. http://dx.doi.org/10.1073/pnas.1903680116.

IEA, 2024. Renewables 2023 – Analysis. URL: https://www.iea.org/reports/renewables-2023.

IEA Wind Task 37, . IEAWindTask37/IEA-3.4-130-RWT. URL: https://github.com/IEAWindTask37/IEA-3.4-130-RWT.

International Energy Agency (IEA), 2016. IEA Wind TCP Task 37. URL: https://iea-wind.org/task37/.

International Energy Agency (IEA), 2019. Wind energy generation systems - Part 3-1: Design requirements for fixed offshore wind turbines.

Jeong, M.S., Kim, S.W., Lee, I., Yoo, S.J., Park, K.C., 2013. The impact of yaw error on aeroelastic characteristics of a horizontal axis wind turbine blade. Renewable Energy 60, 256–268. http://dx.doi.org/10.1016/j.renene.2013.05.014.

Jiménez, □., Crespo, A., Migoya, E., 2010. Application of a LES technique to characterize the wake deflection of a wind turbine in yaw. Wind Energy 13, 559–572. http://dx.doi.org/10.1002/we.380.

Jonkman, J., 2019. Designing for yaw errors using FAST - Wind & Water / Structural Analysis. URL: https://forums.nrel.gov/t/designing-for-yaw-errors-using-fast/1051/7.

Jonkman, J., Doubrawa, P., Hamilton, N., Annoni, J., Fleming, P., 2018. Validation of FAST.Farm Against Large-Eddy Simulations. Journal of Physics: Conference Series 1037, 062005. http://dx.doi.org/10.1088/1742-6596/1037/6/062005.

Kadoche, E., Gourvénec, S., Pallud, M., Levent, T., 2023. MARLYC: Multi-Agent Reinforcement Learning Yaw Control. Renewable Energy 217, 119129. http://dx.doi.org/10.1016/j.renene.2023.119129.

Ke, S., Wang, T., Ge, Y., Wang, H., 2018. Wind-induced fatigue of large HAWT coupled tower–blade structures considering aeroelastic and yaw effects. The Structural Design of Tall and Special Buildings 27, e1467. http://dx.doi.org/10.1002/tal.1467.

Kragh, K.A., Hansen, M.H., 2014. Load alleviation of wind turbines by yaw misalignment. Wind Energy 17, 971–982. http://dx.doi.org/10.1002/we.1612.

Kretschmer, M., Jonkman, J., Pettas, V., Cheng, P.W., 2021. FAST.Farm load validation for single wake situations at alpha ventus. Wind Energy Science 6, 1247–1262. http://dx.doi.org/10.5194/wes-6-1247-2021.

Larsen, G.C., Ott, S., Liew, J., Laan, M.P.v.d., Simon, E., Thorsen, G.R., Jacobs, P., 2020. Yaw induced wake deflection-a full-scale validation study. Journal of Physics: Conference Series 1618, 062047. http://dx.doi.org/10.1088/1742-6596/1618/6/062047.

Lee, J., Son, E., Hwang, B., Lee, S., 2013. Blade pitch angle control for aerodynamic performance optimization of a wind farm. Renewable Energy 54, 124–130. http://dx.doi.org/10.1016/j.renene.2012.08.048.

Lee, S., Churchfield, M., Driscoll, F., Sirnivas, S., Jonkman, J., Moriarty, P., Skaare, B., Nielsen, F.G., Byklum, E., 2018. Load Estimation of Offshore Wind Turbines. Energies 11, 1895. http://dx.doi.org/10.3390/en11071895.

Leonetti, D., Maljaars, J., Snijder, H.H.B., 2017. Fitting fatigue test data with a novel S-N curve using frequentist and Bayesian inference. International Journal of Fatigue 105, 128–143. http://dx.doi.org/10.1016/j.ijfatigue.2017.08.024.

Li, G., Xiong, C., Thabet, A., Ghanem, B., 2020. DeeperGCN: All You Need to Train Deeper GCNs. URL: http://arxiv.org/abs/2006.07739, http://dx.doi.org/10.48550/arXiv.2006.07739.

Lin, M., Porté-Agel, F., 2020. Power Maximization and Fatigue-Load Mitigation in a Wind-turbine Array by Active Yaw Control: an LES Study. Journal of Physics: Conference Series 1618, 042036. http://dx.doi.org/10.1088/1742-6596/1618/4/042036.

Mahmood, A.R., Korenkevych, D., Vasan, G., Ma, W., Bergstra, J., 2018. Benchmarking Reinforcement Learning Algorithms on Real-World Robots, in: Proceedings of The 2nd Conference on Robot Learning, PMLR. pp. 561–591.

Mao, H., Venkatakrishnan, S.B., Schwarzkopf, M., Alizadeh, M., 2019. Variance Reduction for Reinforcement Learning in Input-Driven Environments. URL: http://arxiv.org/abs/1807.02264, http://dx.doi.org/10.48550/arXiv.1807.02264.

Medici, D., 2005. Experimental studies of wind turbine wakes : power optimisation and meandering. Ph.D. thesis. KTH Royal Institute of Technology.

Meyer Forsting, A.R., Navarro Diaz, G.P., Segalini, A., Andersen, S.J., Ivanell, S., 2023. On the accuracy of predicting wind-farm blockage. Renewable Energy 214, 114–129. http://dx.doi.org/10.1016/j.renene.2023.05.129.

Miner, M.A., 1945. Cumulative Damage in Fatigue. Journal of Applied Mechanics 12, A159–A164. http://dx.doi.org/10.1115/1.4009458.

Mishnaevsky, L., 2022. Root Causes and Mechanisms of Failure of Wind Turbine Blades: Overview. Materials 15, 2959. http://dx.doi.org/10.3390/ma15092959.

National Renewable Energy Laboratory (NREL), a. FLORIS: FLOw Redirection and Induction in Steady State. URL: https://www.nrel.gov/wind/floris.html.

National Renewable Energy Laboratory (NREL), b. SOWFA: Simulator fOr Wind Farm Applications. URL: https://www.nrel.gov/wind/nwtc/sowfa.html.

National Renewable Energy Laboratory (NREL), 2024a. NREL/WindSE. URL: https://github.com/NREL/WindSE. original-date: 2019-04-26T21:00:20Z.

National Renewable Energy Laboratory (NREL), 2024b. OpenFAST/openfast. URL: https://github.com/OpenFAST/openfast. original-date: 2016-08-31T20:07:10Z.

Ng, A.Y., Harada, D., Russell, S.J., 1999. Policy Invariance Under Reward Transformations: Theory and Application to Reward Shaping, in: Proceedings of the Sixteenth International Conference on Machine Learning, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA. pp. 278–287.

Nord Pool Group, . Nord Pool | Day-ahead prices. URL: https://data.nordpoolgroup.com/auction/day-ahead/prices?deliveryDate=latest&currency=EUR&aggregation=Hourly&deliveryAreas=AT.

NREL, . MLife. URL: https://www.nrel.gov/wind/nwtc/mlife.html.

NREL, 2024a. ROSCO. URL: https://github.com/NREL/ROSCO. original-date: 2019-11-08T15:47:14Z.

NREL, 2024b. WISDEM/CCBlade. URL: https://github.com/WISDEM/CCBlade. original-date: 2013-09-12T13:15:11Z.

Padullaparthi, V.R., Nagarathinam, S., Vasan, A., Menon, V., Sudarsanam, D., 2022. FALCON- FArm Level CONtrol for wind turbines using multi-agent deep reinforcement learning. Renewable Energy 181, 445–456. http://dx.doi.org/10.1016/j.renene.2021.09.023.

Panja, M., Chakraborty, T., Kumar, U., Hadid, A., 2024. Probabilistic AutoRegressive Neural Networks for Accurate Long-range Forecasting, in: arXiv. volume 1967, pp. 457–477.

Pedersen, M.M., Forsting, A.M., Laan, P.v.d., Riva, R., Romàn, L.A.A., Risco, J.C., Friis-Møller, M., Quick, J., Christiansen, J.P.S., Rodrigues, R.V., Olsen, B.T., Réthoré, P.E., 2023. PyWake 2.5.0: An open-source wind farm simulation tool. DTU Wind, Technical University of Denmark .

PyWake Development Team, 2024. Automated validation report for PyWake. Technical Report. DTU Wind, Technical University of Denmark.

Qian, Q., Ma, P., Wang, N., Zhang, H., Wang, C., Li, X., 2023. Research on Industrial Process Fault Diagnosis Based on Deep Spatio-Temporal Fusion Graph Convolutional Network. URL: https://papers.ssrn.com/abstract=4651621, http://dx.doi.org/10.2139/ssrn.4651621.

Quaeghebeur, E., Bos, R., Zaaijer, M.B., 2021. Wind farm layout optimization using pseudo-gradients. Wind Energy Science 6, 815–839. http://dx.doi.org/10.5194/wes-6-815-2021.

Rivera-Arreba, I., Li, Z., Yang, X., Bachynski-Polić, E.E., 2023. Validation of the dynamic wake meandering model against large eddy simulation for horizontal and vertical steering of wind turbine wakes. URL: http://arxiv.org/abs/2308.01004, http://dx.doi.org/10.48550/arXiv.2308.01004.

Sahin, A.D., Sen, Z., 2001. First-order Markov chain approach to wind speed modelling. Journal of Wind Engineering and Industrial Aerodynamics 89, 263–269. http://dx.doi.org/10.1016/S0167-6105(00)00081-7.

Scarselli, F., Gori, M., Tsoi, A.C., Hagenbuchner, M., Monfardini, G., 2009. The Graph Neural Network Model. IEEE Transactions on Neural Networks 20, 61–80. http://dx.doi.org/10.1109/TNN.2008.2005605.

Scholz, T., Lopes, V.V., Estanqueiro, A., 2013. A cyclic time-dependent Markov process to model daily patterns in wind turbine power production. URL: http://arxiv.org/abs/1310.3073, http://dx.doi.org/10.48550/arXiv.1310.3073.

Schulman, J., Moritz, P., Levine, S., Jordan, M., Abbeel, P., 2018. High-Dimensional Continuous Control Using Generalized Advantage Estimation. URL: http://arxiv.org/abs/1506.02438, http://dx.doi.org/10.48550/arXiv.1506.02438.

Seita, D., 2017. Going Deeper Into Reinforcement Learning: Fundamentals of Policy Gradients. URL: https://danieltakeshi.github.io/2017/03/28/going-deeper-into-reinforcement-learning-fundamentals-of-policy-gradients/.

Shaler, K., Quon, E., Ivanov, H., Jonkman, J., 2023. Wind Farm Structural Response and Wake Dynamics for an Evolving Stable Boundary Layer: Computational and Experimental Comparisons. Wind Energy Science Discussions , 1–18http://dx.doi.org/10.5194/wes-2023-138.

Shamshad, A., Bawadi, M.A., Wan Hussin, W.M.A., Majid, T.A., Sanusi, S.A.M., 2005. First and second order Markov chain models for synthetic generation of wind speed time series. Energy 30, 693–708. http://dx.doi.org/10.1016/j.energy.2004.05.026.

Sheehan, H., Poole, D., Filho, T.S., Bossanyi, E., Landberg, L., 2024. Graph-based Deep Reinforcement Learning for Wind Farm Set-Point Optimisation. Journal of Physics: Conference Series 2767, 092028. http://dx.doi.org/10.1088/1742-6596/2767/9/092028.

Shen, X., Zhu, X., Du, Z., 2011. Wind turbine aerodynamics and loads control in wind shear flow. Energy 36, 1424–1434. http://dx.doi.org/10.1016/j.energy.2011.01.028.

Stanfel, P., Johnson, K., Bay, C., King, J., 2020. A Distributed Reinforcement Learning Yaw Control Approach for Wind Farm Energy Capture Maximization: Preprint.

Stehly, T., Duffy, P., Hernando, D.M., 2023. 2022 Cost of Wind Energy Review http://dx.doi.org/10.2172/2278805.

T, E., K, M., K, T., K, K., M, M., 1974. Damage evaluation of metals for random or varying loading - Three aspects of rain flow method. Symp Mech Behav Mater , 371–380.

Tang, H., Hao, J., Lv, T., Chen, Y., Zhang, Z., Jia, H., Ren, C., Zheng, Y., Meng, Z., Fan, C., Wang, L., 2019. Hierarchical Deep Multiagent Reinforcement Learning with Temporal Abstraction. URL: http://arxiv.org/abs/1809.09332, http://dx.doi.org/10.48550/arXiv.1809.09332.

Towers, M., Terry, J.K., Kwiatkowski, A., Balis, J.U., Cola, G.d., Deleu, T., Goulão, M., Kallinteris, A., KG, A., Krimmel, M., Perez-Vicente, R., Pierré, A., Schulhoff, S., Tai, J.J., Shen, A.T.J., Younis, O.G., 2023. Gymnasium. URL: https://zenodo.org/record/8127025.

Weipao, M., Chun, L., Jun, Y., Yang, Y., Xiaoyun, X., 2016. Numerical Investigation of Wake Control Strategies for Maximizing the Power Generation of Wind Farm. Journal of Solar Energy Engineering 138. http://dx.doi.org/10.1115/1.4033110.

Wu, Y.T., Porté-Agel, F., 2015. Modeling turbine wakes and power losses within a wind farm using LES: An application to the Horns Rev offshore wind farm. Renewable Energy 75, 945–955. http://dx.doi.org/10.1016/j.renene.2014.06.019.

Yoon, J., Jarrett, D., van der Schaar, M., 2019. Time-series Generative Adversarial Networks, in: Advances in Neural Information Processing Systems, Curran Associates, Inc.

Zhou, J., Cui, G., Hu, S., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C., Sun, M., 2021. Graph Neural Networks: A Review of Methods and Applications. URL: http://arxiv.org/abs/1812.08434, http://dx.doi.org/10.48550/arXiv.1812.08434.

Zong, H., Porté-Agel, F., 2021. Experimental investigation and analytical modelling of active yaw control for wind farm power optimization. Renewable Energy 170, 1228–1244. http://dx.doi.org/10.1016/j.renene.2021.02.059.

Ørsted, . Offshore wind data. URL: https://orsted.com/en/what-we-do/renewable-energy-solutions/offshore-wind/offshore-wind-data.

A

Scientific Paper

Graph-based Deep Reinforcement Learning for Maintenance-Conditioned Wind Farm Wake Steering Control

Bas van Berkel
Delft University of Technology
B.D.vanBerkel@student.tudelft.nl

Gregory Duthé
ETH Zürich
duthe@ibk.baug.ethz.ch

Giacomo Arcieri
ETH Zürich
giacomo.arcieri@ibk.baug.ethz.ch

Pablo G. Morato
Delft University of Technology
p.g.moratodominguez@tudelft.nl

Xiaoli Jiang
Delft University of Technology
X.Jiang@tudelft.nl

Eleni Chatzi
ETH Zürich
chatzi@ibk.baug.ethz.ch

Abstract

We investigate how long-term revenue in wind farms can be maximised, considering both profits through power optimisation and maintenance costs through fatigue-induced component failures due to wake steering control. First, we construct a realistic wind farm simulation environment, based on a Graph Neural Network (GNN) surrogate wind farm simulation model, to facilitate efficient reinforcement learning training. Next, we use that to train reinforcement learning agents based on a GNN architecture, resulting in agents that can generalise across all wind conditions and unseen wind farm layouts. We find that an 'informed' agent that considers all profits and costs involved manages to significantly reduce the cost of energy compared to 'greedy' (power optimisation only), 'risk-averse' (damage minimisation only) and 'baseline' (zero-yaw) policies, furthermore considerably maximising long-term wind farm profit by as much as 20%.

1 Introduction

As the world is transitioning to the usage of sustainable energy sources in an attempt to tackle climate change, wind energy is growing to be an essential area of energy production. In a recent report by the International Energy Agency (IEA), they forecast that by 2028, as much as 12.1% of all energy will come from wind energy alone, making up more than a quarter of all sustainable energy produced [1]. This indicates a growth of nearly 9% per year on average. Additionally, the IEA found that the majority of newly installed wind energy sources can provide competitive Levelized Cost Of Energy (LCOE) prices compared to fossil fuel sources, often even outperforming them entirely. It is thus evident that wind energy will play a vital role in the energy transition and that effective and cost-efficient operations are essential to replace the current fossil fuel energy system.

Wind farms often deal with the effects of wakes, e.g. the disturbed air that flows downstream of a wind turbine. In the process of generating energy out of the atmospheric boundary layer, wind turbines leave behind more turbulent and lower-velocity wind. This tends to be a significant problem, as these velocity deficits cause decreased performance of downstream turbines and, thereby, reduced power production for the wind farm as a whole [2, 3]. This problem often worsens as wind farm size increases due to an increasing number of turbines being aligned relative to the wind direction. Despite wind farm operators' best efforts to ensure the most optimal wake-effect-mitigating wind farm layouts under dominant wind directions, many wind conditions will still yield wake problems and, consequently, sub-optimal wind farm performance. Wake effects in wind farms can cause as much as a 40% efficiency loss in cases where turbines are aligned relative to the wind [4]. Additionally, the

increased flow field turbulence in wakes can cause significant increases in aeroelastic loads in the downstream turbines [5].

In recent years, one proposed solution to this problem came in the form of wake steering. Using the turbines' yawing capability, e.g. rotating the nacelle relative to the tower, the propagation direction of the produced wakes can be adjusted. Under default circumstances, where turbines are yawed to face the incoming wind directly, the wakes propagate parallel to the wind direction. However, assigning a yaw misalignment to upstream turbines facilitates a degree of control over the direction in which the wakes propagate through the farm. Using this wake steering technique, problematic wakes can be steered away from downstream turbines, thereby mitigating the velocity deficit and increased turbulence effects, which would have otherwise resulted in significant adverse effects downstream. Despite the yawing action causing sub-optimal power production for the upstream turbines, the improved performance of downstream turbines collectively ensures a net benefit for the wind farm. Wake steering techniques facilitate several per cent performance increases compared to the naive wake-affected 'do-nothing' zero-yaw policy typically used in wind farms [6]. Figure 1 and Figure 2 show the flow field and Annual Energy Production (AEP) effects of the do-nothing and optimised strategy, respectively.

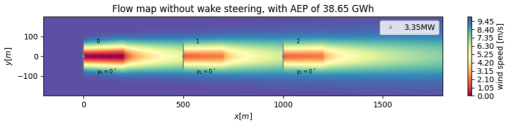

Figure 1: Flow map of a three-turbine wind farm without wake steering.

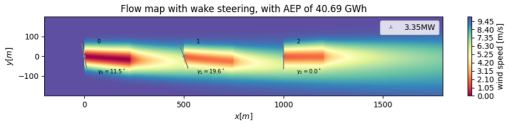

Figure 2: Flow map of a three-turbine wind farm with wake steering.

However, the yaw misalignment in wake steering leads to undesired secondary effects. Generally designed for zero-yaw operating conditions, turbines can experience increased loads due to the yawing action [7]. What initially seems beneficial performance through power optimisation with wake steering is more nuanced. The resulting increased loads, cyclic in nature, lead to accelerated fatigue degradation of wind turbine components. Fatigue failure, typically the dominating factor in turbine lifetime design, comes with significant maintenance costs, often requiring the complete replacement of components. What is more, the acceleration of fatigue accumulation only becomes visible in the long term when components start to fail, which can be as much as 20 years after the commissioning of the wind farm. The positive consequences of wake steering are instantaneous, whereas the adverse effects can linger in the background unnoticed for many years, after which they can cause significant monetary losses.

The problem of revenue optimisation thus becomes twofold. On the one hand, instantaneous power optimisation through wake steering allows immediate increases in energy production and, thereby, profit through energy sales. On the other hand, long-term fatigue effects coupled with wake steering for power optimisation cannot be ignored as they can potentially eventually nullify any revenue gains obtained. To keep the cost of energy low and to optimise the yield of sustainable energy sources, the typical wake steering controllers must be adjusted to account for all effects involved. In this paper, we show how a centralised reinforcement learning controller can be used to solve the optimisation problem at hand. We develop a simulation environment that can model the power and damage effects of wake steering in a wind farm to allow fast and efficient reinforcement learning. Next, we develop graph-based reinforcement learning agents using PPO and train them to find the optimal yaw angle strategy under varying wind conditions. We show the influence of the objective function on several performance indicators and show the agents' performance on three different farm layouts. Finally, we investigate the ability of the agents to generalise to unseen farm layouts.

The following contributions are made in this paper:

- **IEA37 3.4MW Turbine-Level Load surrogate** - Development of power, thrust coefficient and load surrogates for the NREL-developed 3.4MW reference wind turbine, considering operating regions, operating modes, wind conditions and yaw angles.
- **Graph Neural Network Farm-Level Surrogates** - Development of an all-in-one combined layout-agnostic wind farm simulation surrogate, considering operating modes, wake effects and local power & loads.

- **WakeWISE Environment** - Development of a fast, efficient and realistic simulation environment for training wind farm wake steering controllers using reinforcement learning.
- **Wind Farm Wake Steering Controllers** - Development of generalisable graph-based wind farm wake steering controllers to optimise for power, maintenance or a combination of both.

This paper is structured as follows. section 3 shows how we model damage accumulation in the form of fatigue. Next, section 4 describes how we developed a fast and efficient simulation environment to accurately and realistically mimic a real wind farm. section 5 describes how we constructed the reinforcement learning agent and how the training was done. Finally, section 6 provides an analysis of the performance of the trained agents. The results of this paper are discussed in section 7 and concluded in section 8.

2 Related Work

Early work in wake steering can be traced back to 2005, in research by Medici [8] investigating the effects of Active Yaw Control (AYC). More recently, Howland et al. [9] conducted experiments on a six-turbine in-line wind farm, using analytic gradient ascent methods to optimise yaw setpoints. For selected wind directions, they showed that performance increases in power yield of the range $7-12\%$ are feasible. These results align well with earlier work highlighting experimental results from field tests using controllers trained in simulations [10, 11], showing performance gains of several per cent. In both cases, they developed wake-steering optimised yaw angle lookup tables using gradient descent and FLORIS [12] as a simulation model. Zong and Porté-Agel [13] aimed to reproduce these aforementioned results in a controlled wind tunnel environment, measuring stream-wise velocity deficits and turbine power outputs. Their results show agreement with prior research and experiments. Other examples of wake steering optimisation include the usage of Game Theory [14] and SNOPT, an optimiser for highly nonlinear, constrained problems [15].

In contrast, Stanfel et al. [16] employed decentralised multi-agent Q-learning for power optimisation instead in a three-turbine wind farm under a fixed wind condition. Under the single wind condition they considered, they achieved a clear increase in wind farm power production. Similarly, Bui et al. [17] grouped a 15-turbine wind farm in clusters of size three for information sharing and solved the optimisation problem using decentralised Double-Deep Q Learning. Using their MA-DRL approach under specific wind conditions, they achieved a power increase ranging from 1.99% to 4.11% depending on the layout of the three-turbine cluster. Dong et al. [18], on the other hand, chose to make use of high-fidelity Large Eddy Simulations (LES) using SOWFA [19] to optimise power production in a six-turbine wind farm. Using Deep Deterministic Policy Gradient (DDPG), they found a 15% increase in power compared to a zero-yaw baseline under a fixed flow field. Kadoche et al. [6] and Padullaparthi et al. [20] extend this to work with as many as 151 turbines. Both works use multi-agent RL, using either PyWake or FLORIS as wind farm simulators, and find power increases as high as several per cent depending on the farm size. In contrast to other work, Kadoche et al. [6] train their algorithm on an environment with (limited) dynamic wind conditions rather than the fixed conditions the majority of other papers in this section made use of. Additionally, Padullaparthi et al. [20] extend their simulation to include a measure of fatigue to effectively find a trade-off between power optimisation and component longevity. They include this in the reward function employing an arbitrarily chosen weighted sum between power production and fatigue loads.

This trade-off between power optimisation and fatigue loads is an important factor, as yawing action can significantly impact fatigue loads on components [7]. Interestingly, yaw control can minimise fatigue loads on the turbine [21–23]. However, Ke et al. [24] and Jeong et al. [25] also show that applying yaw offsets to wind turbines can lead to detrimental effects, such as significantly decreasing fatigue lifetime. These effects can thus not simply be neglected in favour of immediate power optimisation. Lin and Porté-Agel [26] investigated this trade-off through wake steering and found that in some cases, the benefits of power production increase are neglected by the increased loads' negative effects. In the same way that Padullaparthi et al. [20] modelled the trade-off between loads and power in their reward function, work by Ennis et al. [27], Bossanyi [28] and van Dijk et al. [29] attempted to solve the problem as a multi-objective optimisation problem by giving relative importance to power optimisation and load minimisation. Many of these works consider only a single DEL (fatigue load) in the objective function, such as Ennis et al. [27] (flapwise loads) and Bossanyi [28] (tower base loads). To address this issue, He et al. [30] consider five different fatigue loads to characterise the wake-fatigue effects as well as possible. However, all of these works addressing

the trade-off between power and load treat the problem as a multi-objective optimisation problem, neglecting the true monetary losses resulting from maintenance due to fatigue degradation.

Our work focuses on maintenance-conditioned wake steering for wind farm revenue optimisation without making any explicit assumptions on the relative importance of damage minimisation versus power maximisation. Additionally, we train and evaluate a wide variety of wind directions to ensure the controllers remain wind-condition-agnostic. Finally, we develop our agents in such a way that they are layout-agnostic and able to generalise to larger, more realistic wind farm topologies.

3 Fatigue Modelling

To ensure the agent receives feedback concerning the loads incurred by the turbines as a result of the yaw angles, some type of damage penalty must be designed. Fortunately, the IEC 61400-1 standard on wind turbine design [31] provides some guidance on this topic. Wind turbines are generally limited by their fatigue lifetime resulting from cyclic loads during operation. This fatigue degradation can be modelled as an accumulation of damages caused by individual load cycles. A typical measure for fatigue load is the Damage Equivalent Load (DEL); the DEL allows the fatigue damage contribution of various load cycles of different magnitudes to be summarised into a single cyclic load. This load, when applied over the same period, would result in an equivalent contribution to fatigue degradation. Individual load cycles i can be converted to damage contributions ΔD using Equation 1.

$$\Delta D = \sum_i \frac{n_i L_i}{N(L_i)} \tag{1}$$

Where L_i is the magnitude of the cyclic load, n_i is the number of cycles at that load and $N(L_i)$ is the number of cycles at load L_i, which would result in failure. According to Palmgren-Miner's rule, these individual contributions ΔD can then be summed linearly to obtain the total fatigue damage accumulation D. $N(L_i)$ is often modelled as an exponential curve of the form $\left(\frac{L_i}{L_u}\right)^{-m}$, where L_u is the ultimate load and m is a parameter called the Wöhler exponent. An arbitrary series of load cycles L_i with cycle counts n_i can be transformed into a DEL by equating their damage contributions. This requires the DEL to be defined at some reference frequency f_{eq} over the total time period T_s of the original to-be-transformed signal.

$$f_{eq} T_s \left(\frac{DEL}{L_u}\right)^m = \sum_i n_i \left(\frac{L_i}{L_u}\right)^m \tag{2}$$

$$f_{eq} T_s DEL^m = \sum_i n_i L_i^m \tag{3}$$

$$DEL = \sqrt[m]{\frac{\sum_i n_i L_i^m}{f_{eq} T_s}} \tag{4}$$

The damage contributions of multiple series of load cycles can thus be computed by using their respective DEL loads with Equation 1, replacing n_i by $f_{eq}T_s$ and L_i by the DEL-value. Palmgren-Miner's rule dictates that fatigue failure occurs upon reaching a damage-state D of 1. In this paper, we investigate the contributions of five different fatigue loads: the blade-root edgewise and flapwise moments, the tower base side-to-side and fore-aft moments and the tower top yaw moment. We choose these loads specifically as prior research by Carroll et al. [32] and Dao et al. [33] has shown that these are typically and often subject to fatigue failure. The conventions for each of these moments are shown in Figure 3.

4 Environment

4.1 Problem Formulation

Wake steering offers a feasible solution to tackle performance deterioration in wind farms due to wake effects. By introducing a yaw misalignment to various upstream turbines, the problem of

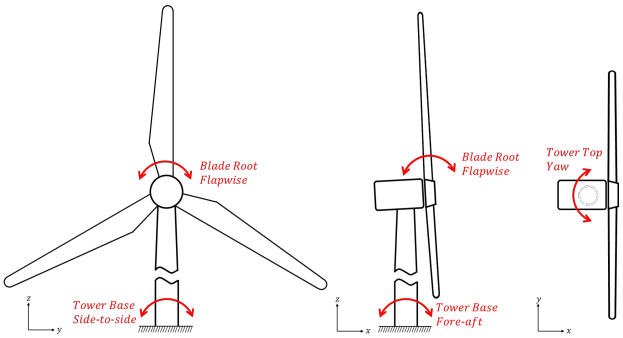

Figure 3: Conventions for the fatigue loads under investigation.

downstream-propagating velocity deficits and turbulence increases ending up in downstream turbines can be partially mitigated. This facilitates nearly instantaneous increases in power production and, thereby, in short-term profit. However, as a result of wake steering actions, the accumulation of fatigue damage in turbines can be accelerated, causing degradation effects that are unaccounted for in power-only optimisation. If these effects are not considered, the instantaneous revenue gains can eventually be nullified by the introduction of more component replacements due to fatigue failure. The problem at hand is thus to construct and train a wake steering controller that can explicitly account for all these effects and find the policy that maximises wind farm profit.

The reinforcement learning agent is concerned with the optimisation of the cumulative reward R over each episode. Based on the state s_t of the environment at timestep t, the agent takes an action a_t containing the yaw angles for all turbines in the environment, according to a policy π. It receives a reward r_t, based on a combination of costs c_t and profits p_t:

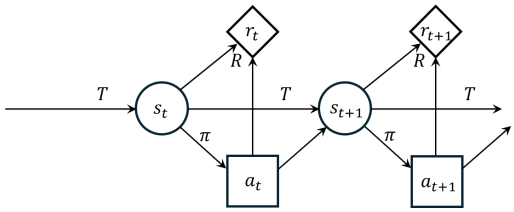

Figure 4: Diagram of the Markov Decision Process

The profits are dependent on power production, whereas the cost is dependent on maintenance costs; these will be introduced later. The agent must learn to find the (near) optimal set of parameters θ for π, such that the policy π^*_θ maximises the expected cumulative reward R over each episode of length T. The discount factor γ discounts future rewards.

$$\pi^* = \arg\max_{\pi} E\left[\sum_{t=0}^{T} \gamma^t r_t\right] \tag{5}$$

The RL problem is therefore defined as the tuple $\langle S, A, R, P, \gamma \rangle$ with the state-space S, action space A, reward function R, transition function P implemented by the environment and discount factor γ. We use 10-minute timesteps in the environment and further assume all conditions are fixed during the timestep.

4.2 Simulation

To model the power- and load effects of the turbines given input yaw actions, a farm-wide simulation model is required to determine wake interactions and local physical variables at each turbine. Typical numerical solvers that account for wake interactions in wind farms, such as PyWake [34], FAST.FARM [35] or SOWFA [19] require runtimes in the order of seconds up to minutes. Training an

RL agent in the environment will require simulating millions of timesteps to find a well-performing policy for which millisecond inference is desirable. However, the models mentioned above are order(s) of magnitude too slow. Furthermore, they might require additional models to determine the local turbine-level effects on top of the farm-level effects to obtain power outputs and aeroelastic loads. Instead, we use a surrogate model that can simultaneously simulate farm-level wake interactions and turbine-level physical variables with inference speeds in the order of milliseconds. This is inspired by previous work by Duthé et al. [36], which has shown that surrogate modelling of wake-affected turbines using Graph Neural Networks is feasible, efficient and reasonably accurate.

Training such a GNN-based surrogate requires a dataset of input-output pairs of simulation results. Here, the inputs are the free-stream wind flow conditions around the farm combined with yaw angles for each of the turbines; the outputs consist of the local flow conditions, thrust coefficient, power level and fatigue loads at each turbine. To generate this dataset, we use PyWake, as it offers an efficient model with a Python interface to facilitate fast dataset generation. PyWake can model wake interactions in the wind farm but cannot determine turbine-level physical variables under yaw misalignments and waked conditions. To tackle this, we train another model to supplement PyWake in the form of a turbine-level surrogate that infers local thrust coefficient, power and fatigue loads given the local flow conditions. In essence, PyWake and the surrogate will work together; PyWake computes the wake interactions and provides the local wake-affected flow conditions at each turbine, after which the turbine-level surrogates will determine the physical variables. Development of the GNN-based complete surrogate is thus a two-stage process: first, developing a turbine-level surrogate to complement PyWake, and then developing the GNN-based complete surrogate itself.

The turbine-level surrogate itself, too, needs a dataset of input-output sets. We use OpenFAST as a simulator for the turbine to compute numerically the desired physical variables. OpenFAST requires 1) a model of the turbine to be modelled and 2) a flow field to which the turbine is subject. For the turbine model, we make use of the 3.4MW full-scale reference turbines developed by the International Energy Agency (IEA) [37]. It offers a well-documented turbine with, most importantly, pre-configured model files and settings for simulating with OpenFAST. Additionally, they provide tuned settings for the ROSCO controller [38], which can be combined with the turbine. With the turbine files ready, what remains is the generation of the flow fields to be used in the simulation. These should cover a wide range of inflow conditions to ensure the trained surrogates generalise as well as possible. We do this using TurbSim [35], a turbulence field generation tool by NREL. Three variables are required to characterise the flow field: mean wind speed V_m, turbulence intensity I and shear exponent α. To cover a wide range of conditions extending beyond the operating region of the turbine, we choose $[-30, +30]$ as the boundaries of the sampled wind speeds. Following the IEC61400-1 standard on wind turbine design [31], we set the upper bound of turbulence intensity as $I = I_{ref}\left(0.75 + \frac{b}{V_{hub}}\right)$, where $I_{ref} = 0.16$, $b = 5.6$, $V_{hub} = V_m$ and $I = \frac{\sigma}{V_m}$ with σ as the standard deviation of wind speed fluctuations. Furthermore, following Haghi and Crawford [39], a fixed lower- and upper bound of 0.04 and 0.5 are enforced to prevent extreme values. For the shear exponent α, we follow the definition by Dimitrov [40] with added hard constraints of $[-0.3, 2.5]$ to tackle exploding values due to low wind speeds extending beyond their use-case. Finally, TurbSim uses seeds to prime the random flow field generation process; for this, we set the bounds as $[0, 2^{16}]$ to provide a wide range of seeds to sample from. Similarly, for the simulation process in OpenFAST later on, random yaw angles need to be sampled. Following work by Zong and Porté-Agel [13], we limit the yaw angles between $[-30, +30]$ as they found the most optimal yaw angles for wake steering to be within that region. All bounds of the distributions are shown in Table 1. To sample from the distribution, we make use of quasi-random SOBOL sampling to obtain an unbiased distribution of samples over the input space; the sampled variables are shown in Figure 5.

Since SOBOL sampling only maintains its distributional properties if the number of samples is a power of 2, furthermore taking inspiration from Haghi and Crawford [39], we produce 2^{15} turbulence files with TurbSim. We used a 24x24 grid resolution with the Kaiman turbulence model as outlined by the IEC61400-1 standard. Furthermore, each turbulence field allows 320 seconds of simulation time. Next, for the simulation in OpenFAST, nearly all default settings found in the model files of the 3.4MW reference turbine are kept. Adjustments we made were enabling all blade DOFs, setting initial pitch angles to 8 degrees and setting the initial rotor RPM to 11.5. Furthermore, in simulation cases outside the regular operating region (wind speed between 4 and 25), pitch control, generator control and generator DOF were turned off. Additionally, for parked (disabled) turbines and cases where the wind speed exceeds 25, the blades were feathered (e.g. pitch angle of 90 degrees). All simulations

Symbol	Name	Lower Bound	Upper Bound
V_m	Mean Wind Speed	0	30
I	Turbulence Intensity	0.04	$min\left(0.5, I_{ref}\left(0.75+\frac{b}{V_{hub}}\right)\right)$
α	Shear Exponent	$max\left(-0.3, \alpha_{ref,LB}-0.23\left(\frac{V_{max}}{V_{hub}}\right)\left(1-\left(0.4log\frac{R}{z}\right)^2\right)\right)$	$min\left(2.5, \alpha_{ref,UB}+0.4\left(\frac{R}{z}\right)\left(\frac{V_{max}}{V_{hub}}\right)\right)$
γ	Yaw Misalignment	-30	30
ζ	Seed	0	2^{16}

Notes:
$I_{ref} = 0.18$ and $b = 5.6$, according to IEA61400-1 standard [31]; $\alpha_{ref,LB}$ and $\alpha_{ref.UB}$ are 0.23 and 0.40 respectively, following [40]
V_{hub} and V_{max} are hub-height and maximum possible wind speed respectively
R and z are rotor diameter and hub height in meters, and 130 & 110 for the 3.4MW turbine respectively.
All samples are chosen uniformly between lower- and upper bounds

Table 1: Overview of distributions of input variables

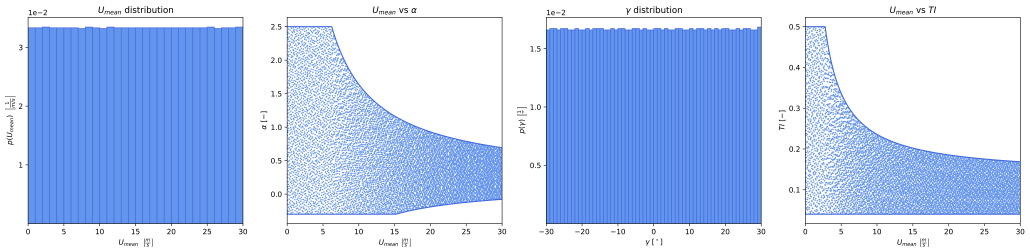

Figure 5: Input variables obtained using SOBOL-sampling

were then carried out using the `openfast-toolbox` Python package, which automatically adjusts the simulation settings to include the correct turbulence files.

Next, all outputs are postprocessed using the same `openfast-toolbox` library. The first 20 seconds of the simulation are discarded to remove transient start-up behaviour, leaving 300 seconds of time series data for analysis. Both thrust coefficient and power are averaged over the entire series. The DEL values are computed using `openfast-toolbox`'s `equivalent_load` function by feeding it the individual time series along with their respective Wöhler exponents found in Table 4. What results is a set of input variables indicating the wind speed V_m, turbulence intensity I, shear exponent α and yaw angle γ together with a set of output variables indicating the local thrust coefficient c_t, power P and the five DEL loads. We now use this dataset to train the turbine-level surrogate model. To do so, we train a separate model for each of the three operating regions: below cut-in wind speed, within the operating window and above cut-out wind speed. We do this as input-output relations can differ wildly depending on whether the turbine is active or not. We construct a three-layer MLP for each output variable separately, consisting of $[32, 64, 32]$ hidden nodes with `LeakyReLu` activation. The dataset is split into a training set (80%) and a validation set (20%). Furthermore, all input and output variables are scaled using min-max scaling. We use the ADAM optimiser with a learning rate of 0.0001, Mean Squared Error (MSE) loss function, 3000 learning epochs and early stopping based on validation loss. This yields 21 networks, three regions for each output variable. Note that the region above cut-off and the parked condition are modelled using the same network, as they exhibit the same turbine settings. Table 2 shows the evaluation metrics of the trained surrogates.

Model	r^2 score	RMSPE	RMSE	MAPE	MAE
P	0.998	0.067	50.388	0.020	25.787
C_t	0.999	0.030	0.008	0.022	0.005
$DEL_{br,ew}$	0.992	0.144	43.483	0.053	27.002
$DEL_{br,fw}$	0.981	0.230	256.816	0.092	167.69
$DEL_{tt,yaw}$	0.974	0.350	219.263	0.134	122.06
$DEL_{tb,ss}$	0.960	0.178	1631.687	0.117	924.066
$DEL_{tb,fa}$	0.943	0.139	2299.924	0.093	1093.924

Table 2: Performance metrics for all seven surrogates averaged over all operating regions.

Next, we combine these surrogates with PyWake to obtain the dataset for the farm-level surrogate training. Here, too, inflow conditions need to be generated. We use the same sampling process as before. Additionally, we introduce a binary 'operating mode' variable that indicates whether a turbine is in operation or parked, for example, due to maintenance work. With a 5% probability, we turn

turbines off, meaning they produce no power or wake and are subject to the turbine-level surrogate conditioned on parked conditions. Finally, we sample random wind farm layouts for the simulation process to ensure the surrogate generalises as well as possible over different topologies. We use a random layout generation process shown in Figure 6 to do this. First, an even grid is generated and randomly perturbed. Next, all coordinates are scaled such that the minimum distance between any two turbines is exactly D_{min}, which we set as $3D \approx 500$. Finally, after rotating the grid, we mask the farm with different shapes to obtain various sets of turbines, making up a variety of farm layouts. We generate three datasets: a training set with 5000 layouts, a validation set with 1500 layouts and a test set with 1500 layouts. Each layout is combined with ten inflow conditions, yielding 50000, 15000 and 15000 simulation cases, respectively.

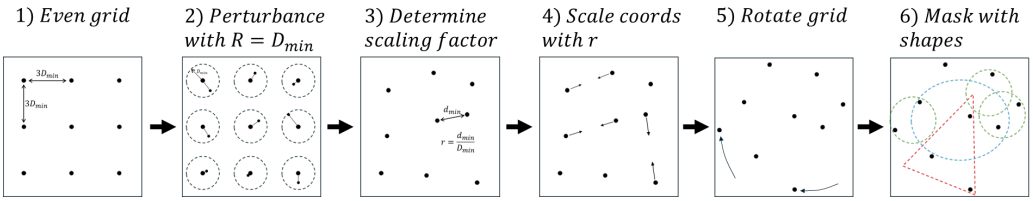

Figure 6: Random wind farm layout generation process.

We configure PyWake as follows. For the wake deficit modelling, we make use of the `BastankhahGaussianDeficit` module, implemented according to Bastankhah and Porté-Agel [41] and used by the authors of the IEA37 reference turbine in their wake studies [42]. Following the work this surrogate modelling is based on [36], we use the `CrespoHernandez` module for calculating turbulence in wakes. We use PyWake's `SquaredSum` module for superpositioning of multiple wakes. Next, for the wake deflection model, we make use of the `JiminezWakeDeflection` module, as it is the only available option that is valid and tested for yaw angles up to thirty degrees [43]. Finally, the custom turbine-level surrogate models developed earlier are injected into the PyWake logic to infer local physical variables. All 80000 cases are then simulated using PyWake to obtain the input-output dataset required for farm-level surrogate training.

Before training the GNN surrogate, we first transform the dataset into graphs that can be fed into the graph-based architecture. To do so, each turbine is represented by a node on a fully connected graph. Each node gets a feature vector for the input graph consisting of the respective turbine's yaw angle, operating mode and global wind conditions. Edge features are constructed by calculating the pairwise polar representation of their difference vector and are bidirectional. Furthermore, the edge features are supplemented with the relative angle between the wind direction and the polar angle of each edge. This is in accordance with work by Duthé et al. [36]. The output graphs are constructed by giving each node a feature vector containing the local wake-affected power, wind speed, turbulence intensity and each of the five fatigue loads. The graph construction process is shown in Figure 7.

The farm-level surrogate's GNN architecture is based on the encode-process-decode paradigm, where each input embedding is first encoded, then propagated on the graph and finally decoded again. Each node feature vector is fed through a node encoder with hidden layers of size $[256, 512, 256]$ with LeakyReLu activation; similarly, each edge feature vector is fed through an edge encoder with hidden layers of size $[256, 256]$ with ReLu activation. Both encoders transform the inputs into a latent space with 256 dimensions. Next, four layers of GENeralized Graph Convolution (GEN, Li et al. [44])

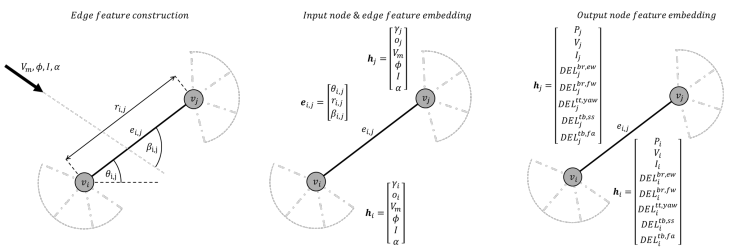

Figure 7: Graph embedding of the farm-level simulation dataset.

with softmax aggregation perform the message passing over the graph. Finally, a node decoder with hidden layers of size $[256, 256]$ with ReLu activation transforms the hidden output graph embedding back into the 8-dimensional output vector. The architecture is visualised in Figure 8. All input and output embeddings are standardised using mean-standard deviation standardisation. The learning rate is set at 0.0005 with a cosine annealing scheduler; 150 epochs of training with the ADAM optimiser and MSE loss are used with training. Furthermore, early stopping based on validation loss is used as well. The evaluation metrics on the test set are shown in Table 3. With this, the full combined wake, power and load simulation model is ready for inference in the environment.

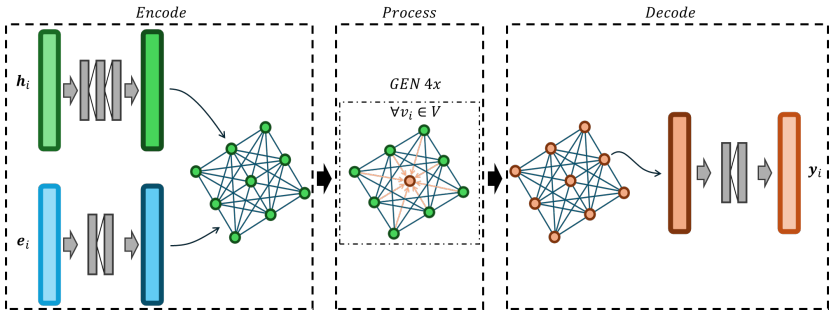

Figure 8: Overview of the architecture of the GNN-based farm-level surrogate.

Variable	r^2 score	RMSE	MAE
P	0.996	25.957	21.929
$V_{W,l}$	0.995	0.121	0.104
TI_l	0.980	0.006	0.005
$DEL_{br,ew}$	0.995	47.463	39.760
$DEL_{br,fw}$	0.986	191.774	157.324
$DEL_{tt,yaw}$	0.986	103.754	85.778
$DEL_{tb,ss}$	0.975	601.581	464.686
$DEL_{tb,fa}$	0.984	902.543	722.481

Table 3: Performance metrics for the farm-level GNN Surrogate model on the test set; subscripts $_l$ indicate that wind speed V_w and TI are the ones local to the turbines, e.g. with wake effects. Relative errors are omitted due to value explosions when ground truth values approach zero.

4.3 Wind Sampling

We model the wind in our environment as a Discrete Time Markov Chain (DTMC) to include temporal coherency in the generated wind signal. Both wind direction and wind speed are modelled as DTMCs. A DTMC consists of a finite set of states $S = x_1, x_2, ..., x_n$ and transition probabilities $P_{ij} = P\{X_{t+1} = x_j | X_t = x_i\}$ where $i, j \in n$ and X_t indicates the state of the system at time t. The Markov property of a DTMC dictates that the probability of moving from state X_t to X_{t+1} is only dependent on the state X_t and not on any of the states $X_0, X_1, ..., X_{t-1}$ preceding it. We thus require a matrix of transition probabilities of shape $n \times n$, where $\sum_{j \in S} P_{ij} = 1$. To construct the transition matrices for wind direction and wind speed, we first obtain a dataset of wind conditions from Ørsted [45], which includes two years of measured wind conditions with a ten-minute resolution. We discretise the wind direction into $N_{wd} = 18$ bins of 20 degrees width, and the wind speed into $N_{ws} = 40$ bins of width 1. We extend the DTMC with an extra dimension for indexing by the M-th month of the year to introduce seasonal effects, thus requiring the inference of a $M \times N_{wd} \times N_{wd}$ matrix of transition probabilities. Using frequency counting of transitions, conditioned on the month of the year, we populate the matrix before dividing it by the total number of transitions. Similarly, for the wind speed, we add an extra dimension to its matrix to account for the current wind direction; this allows the conditioning of wind speed on wind direction. We populate its $N_{wd} \times N_{ws} \times N_{ws}$ matrix the same way we populated the transition matrix for wind directions. Given an initial wind direction state, wind speed state and month of the year, the DTMCs can be used to sample a temporally coherent sequence of states with distributional properties similar to that of the input dataset. Finally, to remove the effects of discretisation, we add uniform noise between $[-10, +10]$ and $[-0.5, +0.5]$ to wind direction and wind speed, respectively. To complete the wind sampling, we add the turbulence intensity I and shear exponent α by uniform sampling from the same distributions as from Table 1.

4.4 Electricity Rate

We model the electricity prices using a lookup table conditioned on the month of the year and the hour of the day to account for seasonal and hourly effects. The time kept by the simulation is used to index a 12×24 matrix of electricity prices; this matrix is inferred from real-life historical electricity prices published by the NordPool energy exchange [46]. The data of electricity market spot prices - which is recorded hourly - is grouped by hour and month and then averaged. For our case, we make use of the 'SYSTEM' price category, which provides a stable reference baseline price. Figure 9 shows an analysis of the used dataset, showing clear signs of seasonal and hourly effects.

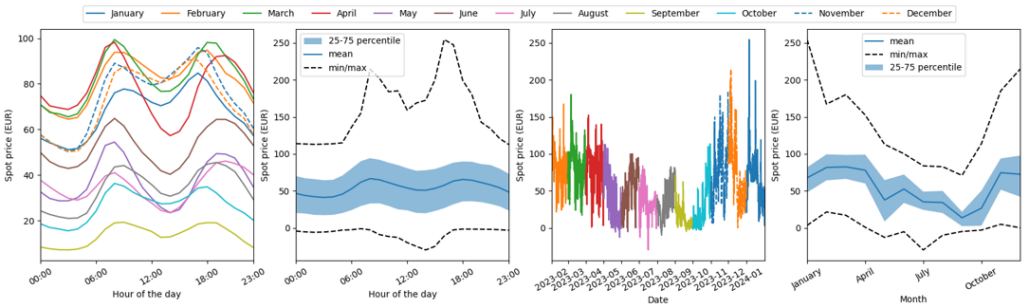

Figure 9: Electricity market spot prices of the NordPool market.

4.5 Damage Accumulation

We model the accumulation of fatigue damage using the methods outlined by the IEC61400-1 standard on wind turbine design, covered in section 3. The GNN-based simulation model outputs, for each turbine in the farm, the DEL values for each of the five locations under consideration. We use Palmgren-Miner's linear damage accumulation rule and maintain separate damage values D for each component for each turbine separately. For each timestep, given the DELs provided by the simulation model, we update the damage state D of each component by adding ΔD computed through Equation 1. We find the parameters required to characterise the exponential $N(L_i)$ curve by first setting 10, 7 and 4 as the Wöhler exponents m for the blade, yaw system, and tower base related DEL loads, respectively. Next, we tune DEL_u with the simulation model to ensure an average 22-year design lifetime under zero-yaw conditions for each component to mimic the process through which turbine components would generally be designed. The obtained fitted parameters, determined with gradient descent, are shown in Table 4.

DEL Location	Component	$m[-]$	$DEL_u[kNm]$	$Cost[EUR]$	$Downtime[Hours]$
Blade Root Edgewise	Blade	10	35634.785	120901.57	300
Blade Root Flapwise	Blade	10	55370.247	120901.57	300
Tower Top Yaw	Yaw System	7	63581.925	125013.75	150
Tower Base Side-to-Side	Tower	4	1962407.473	829755.28	600
Tower Base Fore-Aft	Tower	4	6163272.492	829755.28	600

Table 4: Fitted fatigue curve parameters, replacement costs and downtime duration.

4.6 Maintenance Cost

We introduce two methods of modelling maintenance costs: one 'dense' method and one 'sparse' method. We define the dense method as a model under which the usage of components is directly penalised through a per-per-usage paradigm, linearly relating fatigue damage fractions to fractions of replacement cost incurred. Conversely, the sparse model only incurs cost as soon as a component's damage state D equals or surpasses the damage limit of 1, thereby providing more sparse costs. More formally, given a ΔD of a component during a timestep, the dense cost is modelled as $\Delta D \cdot C$, where C is the total replacement cost. In the end, this sums up to the same value as the sparse case, which instantly returns the total cost C when D surpasses 1, essentially modelling a direct delta function $\delta(D-1) \cdot C$. Examples of how the dense and sparse cost models relate to the damage accumulation

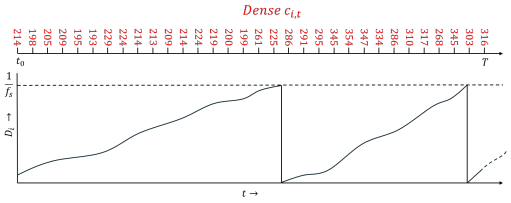

Figure 10: The dense cost model.

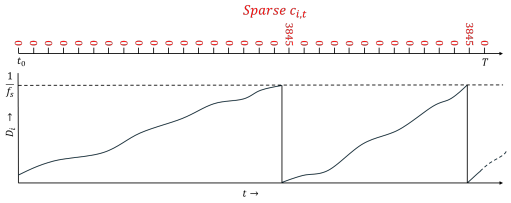

Figure 11: The sparse cost model.

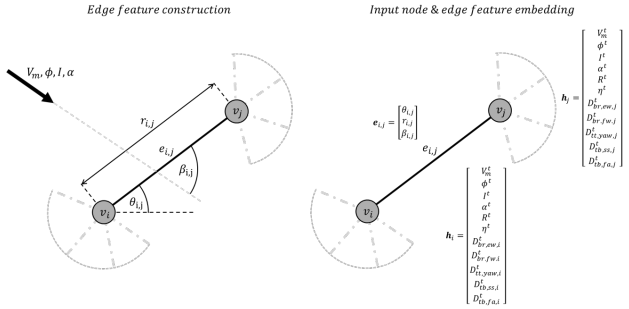

Figure 12: The graph observation of the environment.

are shown in Figure 10 and Figure 11, respectively. Replacement costs are taken from the IEA37 reference turbine design paper [47]. Furthermore, the downtime of turbines due to maintenance upon component failure is based on reviews by Dao et al. [33] and Carroll et al. [32]. All cost and maintenance parameters can be found in Table 4.

4.7 Action & Observation Space

We define the action space as the joint action space of all turbines, e.g. a vector of n dimensions for n turbines containing the yaw angles in the range $[-30, +30]$. Given the graph-like nature through which wind farm effects can be modelled, we believe using a graph-based agent architecture makes sense. As such, we define the observation space as a graph. Coincidentally, we can copy the nodes, graph connectivity and edge features directly from the GNN model embedded in the environment. Next, we supplement this with different node features containing global and local information for each turbine. More specifically, we include the wind conditions (wind speed V_m^t, wind direction ϕ^t, turbulence intensity I^t and shear exponent α^t), electricity rate R^t, the fraction of windfarm lifetime that has elapsed so far η^t and the damage states $D_{i,j}^t$ for each of the components. Figure 12.

4.8 Reward Function

We define the reward function of the environment as the balance between profit obtained through energy production minus cost incurred through maintenance, e.g. damage through usage. The profit of turbine i is determined using the electricity rate R_t and power P_i^t at timestep t, through the relation $p^t = R^t E_i^t$ where $E_i^t = \frac{P_i^t}{6.0e3}$ converts the power to energy production over the 10 minutes that passed during the timestep. The cost $c_{i,j}^t$ of component j of turbine i is modelled using either dense or sparse costs. In total, the reward function becomes:

$$r^t = \sum_i \left[p_i^t - \sum_j c_{i,j}^t \right] \tag{6}$$

4.9 Baseline Removal

The reward returned from the environment as-is will include a lot of variance due to the random processes (e.g. wind and electricity prices), adding a high degree of stochasticity. Similar policies

evaluated under random conditions might thus yield very dissimilar rewards. Actions that typically result in improvement might seem negative due to the high reward variance, and vice versa. In other words, the reward r_t does not always truly reflect the true performance of a policy π. Ideally, this variance would be removed from the environment to get a less noisy signal. This can be achieved through *variance reduction* techniques, such as baseline removal [48]. We can remove a *state-dependent baseline* from the reward at each timestep. By choosing the right action-independent baseline, we can decrease the reward variance without introducing a bias [49]. A logical choice as a baseline is the rewards obtained through a zero-yaw policy, meaning the baseline $b(s_t)$ reflects the reward the environment would have returned under the same conditions if all turbines were set with zero-yaw misalignment. Removing this baseline from the reward at each timestep provides a more stable reward signal, with the added benefit of directly seeing the agent's *relative performance* of the agent compared to the 'default' behaviour. We thus replace the reward r_t with $r_t - b_t$.

5 Reinforcement Learning

For the RL algorithm, we use Proximal Policy Optimisation (PPO), as it is a commonly used algorithm for optimisation with continuous action spaces, and it has historically shown high performance on many tasks. [50, 51]. Given the multi-agent nature of the problem combined with the high importance of farm topology in the decision-making process, we employ a Graph Neural Network based Deep Reinforcement Learning (DRL) architecture. It, therefore, makes sense to build the agent around a fully centralised single-actor framework, Centralised Training with Centralised Execution (CTCE), by representing each agent (turbine) as a node on a fully connected graph. This allows us to represent the RL agent as a single actor that simultaneously infers the actions for all turbines based on the full graph-like observation from Figure 12. Going forward, we use *agent* to refer to the individual fully centralised controller, rather than a turbine in the multi-agent system. We make use of the same encode-process-decode paradigm as was used for the farm-level surrogate model. All node and edge features are encoded onto the graph into a 64-dimension latent space using 2-layer Tanh-activated MLP encoders with $[64, 64]$ hidden nodes. Three layers of GEN graph propagation with SoftMax aggregation then perform the message passing step. We call the architecture up until this point the state encoder, yielding a hidden graph representation with hidden latent vectors for each node. Each hidden node embedding is then fed through a 2-layer Tanh-activated MLP with $[64, 64]$ hidden nodes; each node embedding hereby produces the mean and standard deviation of its action distribution. For the value network, node-wise graph `mean` pooling is used to obtain a single-vector representation of the graph state and consequently fed through a 2-layer ReLu-activated MLP with $[64, 64]$ hidden nodes to yield a single state value. An overview of the network is given in Figure 13. In our implementation, the policy and value networks do not share an encoder, the learning rate is set to 0.0001, each learning batch contains 10 episodes of 1000 timesteps, 20 gradient descent iterations are done, the PPO clip parameter is 0.1, the entropy coefficient is 0.001, the GAE lambda parameter is 0.1, and the discount factor is 1. The training uses the PPO implementation by RLLib. Furthermore, the seasonal in-simulation time is random at the start of each episode to promote diverse seasonal conditions. The agent architecture was constructed by choosing a similar structure as the farm-level surrogate and tuning it to increase average reward at the end of training. All other hyperparameters were tuned in the same way.

Aside from the baseline removal discussed in subsection 4.9, three more techniques are used to ensure stable learning. All observations are normalised to the range $[0, 1]$ based on their respective minimum and maximum attainable values; similarly, all actions are normalised to the range $[0, 1]$ but transformed back to the original range when fed to the environment. Finally, reward scaling is used to ensure the episode-wise accumulated reward is in the order of magnitude of 1. An ablation study regarding the use of a `MLP` versus GNN based architecture, `Baseline Removal` and `Normalisation` is available in Figure 14. It is evident that without baseline removal, the agents exhibit no learning and that the GNN-based architecture ensures faster convergence due to topological relations already being explicitly provided.

We define five agents for comparison, three of which are trained through the above reinforcement learning algorithm:

1. **The random agent** - The random agent has a policy of $\pi(s_t) = random([-30, +30])$ for all turbines.

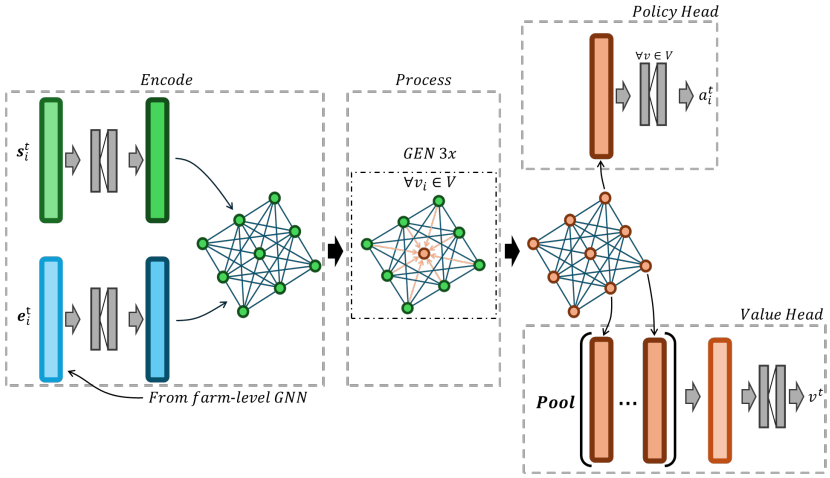

Figure 13: Overview of the architecture of the GNN-based CTCE agent.

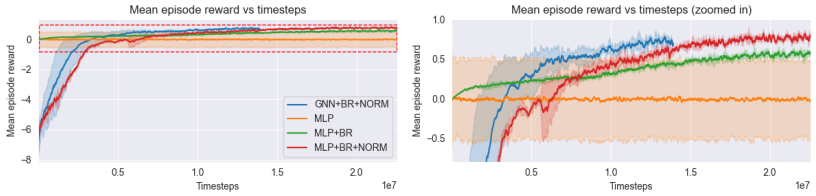

Figure 14: Ablation study for MLP/GNN architecture, baseline removal and normalisation.

2. **The zero-yaw (baseline) agent** - The zero-yaw, 'do-nothing' agent essentially has a policy of $\pi(s_t) = 0$, thereby always giving a yaw offset of zero to all turbines.
3. **The greedy agent** - The greedy agent optimises for power only; maintenance cost is removed from the reward function.
4. **The risk-averse agent** - The risk-averse agent optimises for maintenance cost only; electricity revenue is removed from the reward function.
5. **The informed agent** - The informed agent optimises for power, but also includes the cost of degradation in the reward. It uses the complete reward function.

We define four Key Performance Indicators (KPIs) to investigate the performance of all the agents:

- **Average Turbine Power (ATP)** - The average power produced by any turbine in the farm.
- **Average Turbine Cost (ATC)** - The average maintenance cost for any turbine in the farm.
- **Average Turbine Reward (ATR)** - The net sum of profit obtained through power production minus the maintenance costs incurred.
- **Cost Of Energy (COE)** - The total maintenance cost divided by the total amount of kWh energy produced over the same period.

Finally, we train the agents on three different wind farm layouts, varying in topology and number of turbines:

6 Results

The optimisation is done based on an *infinite horizon* assumption. This has everything to do with the usage of dense costs. Dense costs ensure stable and effective learning for the agent due to frequent feedback on degradation but lack the cost sparsity that would otherwise allow for strategic degradation control to manage failures within a *finite horizon*. In other words, in the case of sparse (realistic)

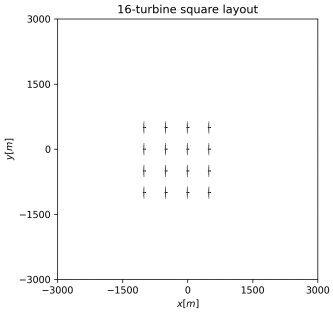

Figure 15: 16-turbine

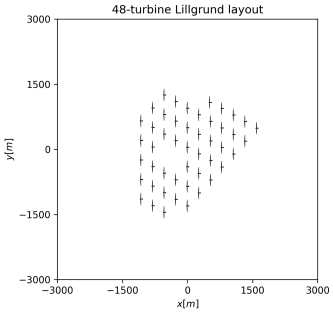

Figure 16: Lillgrund

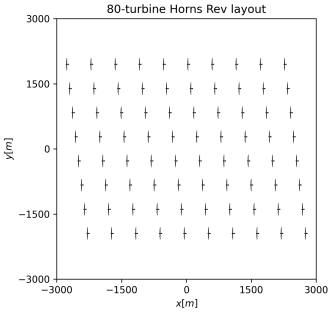

Figure 17: Horns Rev

costs, the agent can ensure subsequent component failures happen outside the finite horizon, should that consequently mean optimal cumulative rewards. In a finite horizon case, dense and sparse costs will present different total costs. However, given an infinite horizon, the *expected cost* of both the dense and sparse model approach the same value and relative differences are small. As such, we present our optimisation on an *infinite horizon* and later investigate the effects of using such policies on a *finite horizon*. Note that the infinite horizon assumption yields a static time-invariant policy, allowing training to happen on episodes with fixed lengths despite the optimisation horizon being infinite. The optimisation horizon does *not* refer to the length of episodes; episode-wise optimality equals infinite-horizon optimality under a static policy.

6.1 Infinite Horizon

Starting with the 16-turbine layout, its training curves for each of the three agents can be found in Figure 18. Each agent, including baseline and random, was evaluated on 200 1000-timestep randomly initialised episodes using the same 200 seeds. The four financially oriented KPIs are shown in Figure 19. ATP, ATC and ATR are expressed in per cent improvement compared to baseline; COE is expressed in absolute terms. ATP, ATC, and ATR/COE correspond to the objectives optimised by the Greedy, Informed and Risk-Averse agents.

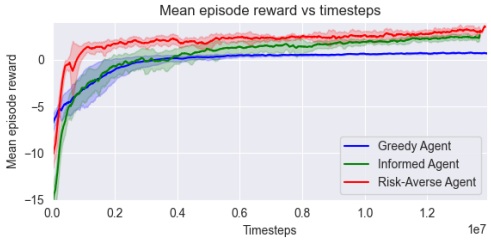

Figure 18: Training curves of the Greedy, Informed and Risk-Averse agents, based on 5 runs each.

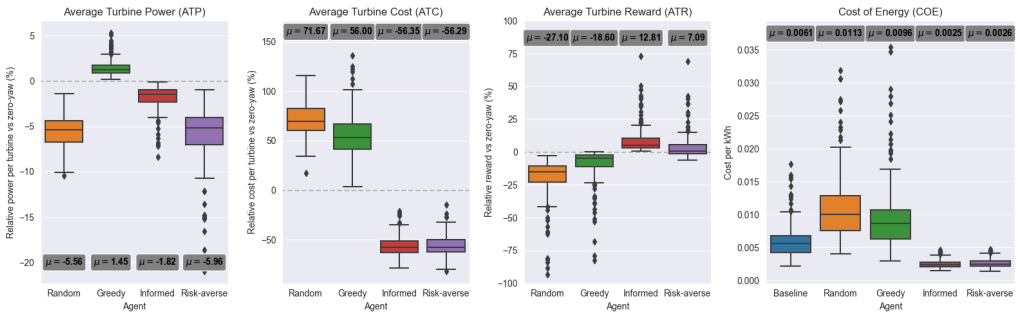

Figure 19: KPIs of the 16-turbine agent compared to baseline.

KPI Analysis. As is evident from the KPIs, each agent can improve their respective objective relative to the baseline. Interestingly, power loss under the random agent is only a few per cent. This results from mitigating actions by the turbine yaw controller, preventing excessive power losses. Pitch control, as was present in dataset generation for the simulation's surrogates, can significantly reduce yaw misalignment losses [52]. The greedy controller can increase power production by 1.5% on average at the cost of increased component fatigue degradation. Both the informed and risk-averse controllers significantly reduce maintenance costs, though the informed controller manages to minimise power losses, whereas the risk-averse controller suffers power losses similar to those of the random policy. Overall, the reduced maintenance costs cause the informed and risk-averse controllers to see a significant reduction in COE. The informed controller, however, produces more energy and thereby obtains a higher total revenue.

Peak Power Increase Analysis. The above statistics highlight the *average* performance of the agents over 200 100-timestep evaluation episodes. These episodes contain a variety of wind conditions, and as such, all metrics show the performance *on average*. To highlight the peak performance of the agent under high-wake conditions, we plot the power improvement versus wind direction for the greedy controller in Figure 20, Figure 21 and Figure 22. As is expected, the peak power improvements happen in the wind directions where turbines align relative to the wind, e.g., in the cardinal directions and their diagonals. These directions cause the highest power losses for the zero-yaw controller. Notable is the few-degree offset of the highest performance gains with the aforementioned high-wake wind directions. This has to do with the inherent difficulty of wake steering in full-wake conditions, where it becomes impossible to deflect a wake sufficiently without significantly compromising upstream power production [13]. Partial-wake conditions thus present the cases where the highest performance is achieved, with relative power increases up to twenty per cent. As wind speed increases, two effects come into play. On the one hand, wake deflection effects due to wake steering are reduced and become increasingly ineffective. On the other hand, wake velocity deficits at high wind speeds might still present decreased wind speeds that are high enough to satisfy downstream turbines without noticeable power loss. Indeed, at a wind speed of 14 m/s wake steering becomes entirely ineffective.

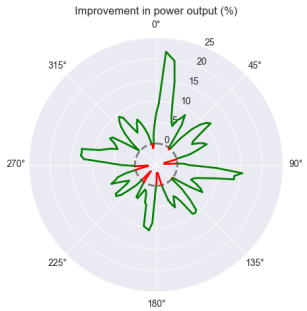

Figure 20: Power improvement vs wind direction, at $V_w = 7$ m/s, $I = 0.1$ and $\alpha = 0.1$.

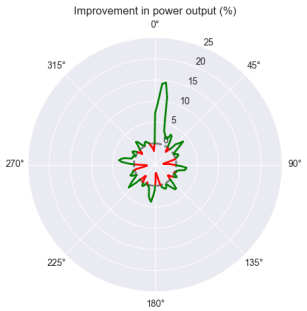

Figure 21: Power improvement vs wind direction, at $V_w = 10$ m/s, $I = 0.1$ and $\alpha = 0.1$.

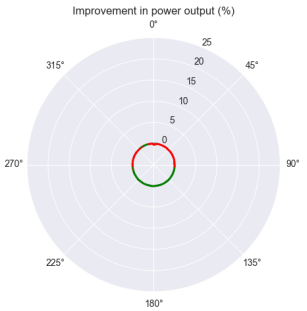

Figure 22: Power improvement vs wind direction, at $V_w = 14$ m/s, $I = 0.1$ and $\alpha = 0.1$.

Electricity Price Dependency. The informed agent balances power production with component degradation. This trade-off comes down to a financial balance between profit through energy sales and costs through maintenance. As such, the price of electricity plays a vital role in this trade-off. Figure 23 illustrates the average yaw angle of each of the 16 turbines throughout all wind directions at various wind speeds and electricity prices. At wind speeds where wake steering is most effective (7 and 8 m/s), there is a noticeable increase in average yaw angle as prices increase. In other words, as improving power production through wake steering becomes more valuable and higher yaw angles are used, the controller can also allow component degradation to increase as its adverse effects are offset by the profits from energy sales. At higher wind speeds, wake steering becomes ineffective, and the damage under yaw misalignments becomes sufficiently large that the controller refuses to employ any wake steering strategy. Interestingly, the yaw angle tends to drop below zero at low electricity prices at 7 and 8 m/s wind speed. The most likely explanation is that the controller goes into 'damage minimisation' mode as it finds a local minimum of turbine loads at a slight negative

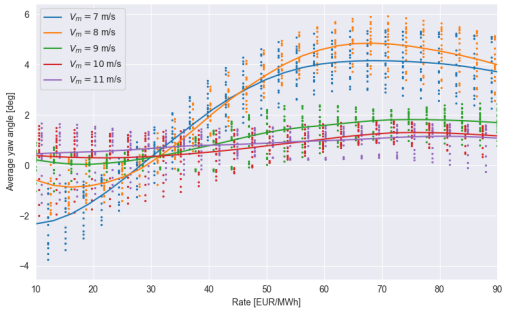

Figure 23: Average yaw angles versus electricity price at various wind speeds; $I = 0.1$ and $\alpha = 0.1$.

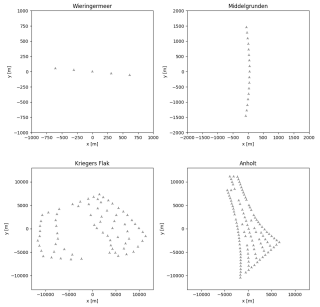

Figure 24: Farm layouts used for the 16T vs 16R study.

yaw misalignment. The subtle power improvement of staying at zero yaw seemingly does not offset the increase in turbine loads due to moving away from the local minimum.

Lillgrund & Horns Rev. We apply the same RL training setup on the Lillgrund and Horns Rev wind farms. Figures 25 and 26 show the relative improvements for each agent. Note that in these figures, each agent is only evaluated based on its optimised objective. Despite all agents improving compared to baseline, their relative improvements are significantly lower than the performances in the 16-turbine farm. We find two possible explanations for this phenomenon. Training might require more iterations, possibly extending up to 48 or 72 hours of runtime, to converge to a solution. However, that is only *if* they will ever converge to a (sub)optimal solution. Lillgrund and Horns Rev include 48 and 80 turbines, meaning an equal amount of agents has to be optimised for in the multi-agent system. Here, we run into the limitations of fully centralised training, which seemingly has trouble exploring and converging to an optimal policy. This theory is further supported by the fact that the agents trained on the Horns Rev farm perform even worse than the Lillgrund agents, indicating a negative trend as farm size increases. Even though all agents still manage to find improvements, the performances are all but impressive.

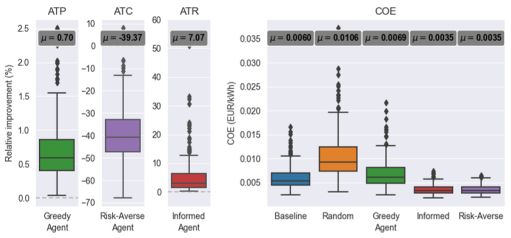

Figure 25: Performance metrics of the Lillgrund agents.

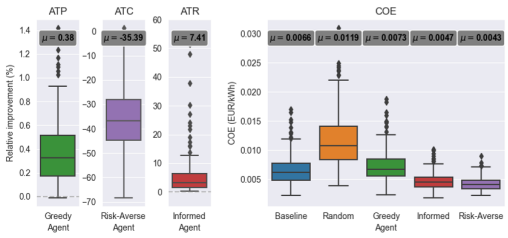

Figure 26: Performance metrics of the Horns Rev agents.

Generalisability. The excellent performance of the 16-turbine agents compared to the Lillgrund and Horns Rev agents naturally spawns the question of whether the learned policies can be transferred to circumvent the limitations of centralised learning. During the construction of the RL agent's deep learning architecture, no assumptions were made on the number of turbines and, therefore, on the graph topology. In theory, the architecture would function just fine on *any* number of connected turbines in *any* topology and connectivity. The question is, however, whether in the context of wind farm control, using such trained graph-based agents enjoys the same topology-agnostic properties as the architecture it is built on. In other words, whether the trained agent *generalises* to wind farm topologies it has not yet encountered during training. This would effectively mean that the problems with centralised learning can be circumvented by leveraging the convergence property in small farms whilst still enjoying performance improvements when transferred to larger farms. To investigate this, we transfer the agent trained on the 16-turbine farm to both the Lillgrund and Horns Rev farms and evaluate its performance. The KPIs are compared in figures 27 and 28.

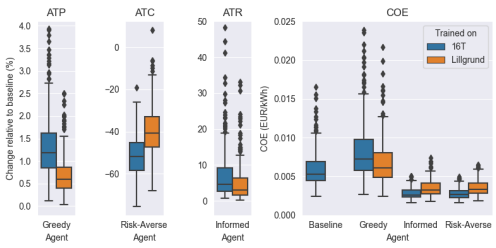

Figure 27: Performance metrics of the 16T and Lillgrund agents on the Lillgrund farm.

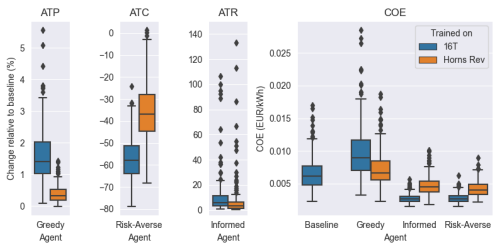

Figure 28: Performance metrics of the 16T and Horns Rev agents on the Horns Rev farm.

Indeed, the 16-turbine agents manage to exceed the performance of the agents specifically trained on the farms themselves. In fact, they manage to nearly equal their respective performances on the 16-turbine farm. This indicates that the trained agents infer actions based on *relative positions* in the farm and do not condition a policy on absolute positions in the farm like an MLP-based architecture would. This property stems from the edge encoding in the input graph, which enables the agents to learn which relative positions require what actions to mitigate wake effects. When presented with unseen layouts, the agent can apply these rules as long as relative positions within the farm are available. Figures 29 and 30 depict the yaw angles and power improvement of the 16-turbine greedy agent during the worst-case wind directions, indeed showing sensible and consistent actions.

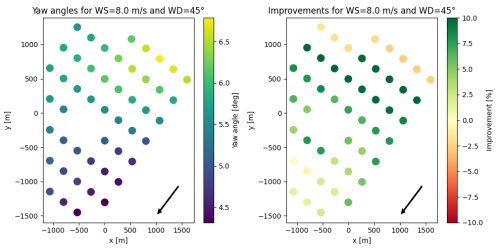

Figure 29: Policy on Lillgrund under worst-case wind direction; $I = \alpha = 0.1$.

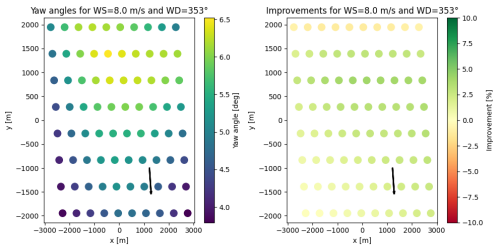

Figure 30: Policy on Horns Rev under worst-case wind direction; $I = \alpha = 0.1$.

The agent only encounters the same grid-like layout during training with fixed relative positions. The question is thus whether it can genuinely generalise well to layouts with different turbine spacings or grid structures. To investigate this, we re-run the training on the 16-turbine farm but randomise the layout upon each environment reset. This is only possible due to the generalizability property we discovered earlier, allowing the agent to learn even under changing wind farm layouts. Ideally, the increased heterogeneity in edge features will allow the trained agents to generalise to various wind farm topologies better. We repeat training for all agents on the 16-turbine randomised (16R) farms and evaluate agents from both training methods on four new wind farms illustrated in Figure 24.

As is evident from these results, the greedy 16R agent can improve power production slightly compared to the 16T greedy agent. It seemingly exhibits better generalizability regarding wake effects in various farm topologies. As for the other agents, however, performance is essentially equal, and no performance improvements are noticeable. The likely cause for this invariance to the training setup stems from the fact that most performance improvements for the informed and risk-averse agents come from reducing loads on a turbine level. These changes can primarily be made regardless of farm topology, barring effects which might occur as part of (partial) wakes. As such, generalizability hardly improves when the agent is trained on more heterogeneous edges. This theory is further supported by the nearly identical cost reduction achieved by the 16T informed and risk-averse agents on both the 16-turbine, Lillgrund and Horns Rev farms. Altogether, training on randomised layouts mainly improves wake steering ability, which is reflected in its increase in greedy power production.

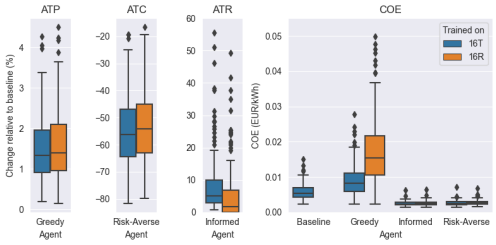

Figure 31: KPIs on the Wieringermeer farm.

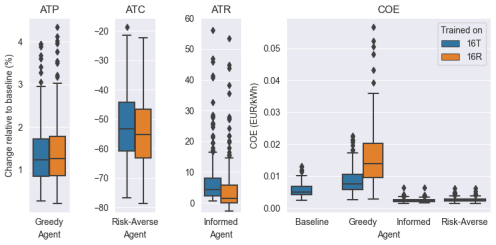

Figure 32: KPIs on the Middelgrunden farm.

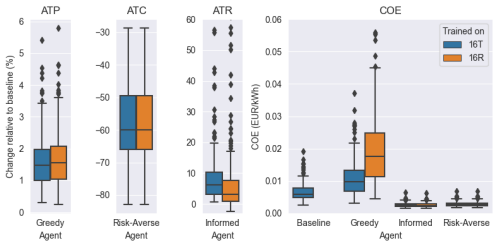

Figure 33: KPIs on the Kriegers Flak farm.

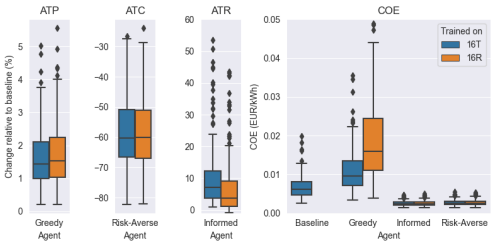

Figure 34: KPIs on the Anholt farm.

6.2 Finite Horizon

Optimising for a fixed and finite wind farm lifetime under realistic sparse costs can bring a completely different optimisation problem compared to finding the best trade-off between component utilisation and power production. Strategic control of component degradation can move entire replacement costs outside the horizon, something which is impossible in the dense cost case. It might, for example, be beneficial to ensure components fail just outside the considered time horizon, optimising power as much as possible without causing an additional replacement due to the yawing behaviour. As such, we investigate the performance of the 'infinite-horizon' agents from the previous section under the more realistic finite-horizon, sparse-cost case. Each agent is evaluated on the 16-turbine farm for a total period of 40 years, plotting their accumulated total balance of profits and costs as a function of time. Each agent is evaluated 20 times on the same 20 seeds, and the mean of the 20 runs is plotted in Figure 35. Below the figures is a timeline indicating which of the agents would obtain the highest cumulative profit *if* the wind farm were only to be considered up until that point. Furthermore, the right plot subtracts the cumulative profit of the baseline 'zero-yaw' agent from all curves.

Figure 35: 40-year analysis of the cumulative profits generated by each policy.

All things considered, the greedy policy initially is optimal as all effects of fatigue and failure can essentially be ignored. It maximises power production profit at the cost of increased component degradation. However, the baseline policy quickly becomes the best for choosing between lifetimes of 9 to 22 years as the greedy agent's components fail. At the 22-year mark, its own components start to break down, whereas the informed policy manages to yet prevent this from happening. Despite initially losing out on power production profit through a more conservative yaw strategy, it can go all the way to approximately 34 years without having accumulated enough failures to once again drop below the baseline performance. At this point, its failure cycle has caught up with that of the baseline policy, and the effects of reduced power production once again emerge. Essentially, in the region just beyond the design lifetime of the turbine components, a policy that preserves components can manage to postpone failure costs as much as possible and thereby maximise profit. This will not always be optimal, though; an example is the risk-averse policy, which minimises damage but is never optimal. This is because it does not consider power production and, therefore, loses out on too much energy production. However, in *very long term*, risk-averse and informed policies will again surpass all other policies once they lag behind in enough failure cycles despite their conservative power production strategy. Ultimately, the informed policy will find the best balance between profits and costs, thereby maximising wind farm net profit.

7 Discussion

Some aspects of the environment and/or agents require some discussion. Firstly, despite the agents' ability to improve on their respective objective functions compared to the baseline, it is difficult to judge whether they have truly found optimal wake steering behaviour or they have found a local minimum. The agents' ability to generalise to a large variety of input parameters, e.g. wind conditions, electricity prices and farm layouts, likely means they have to compromise on some optimality. Comparing their results with that of a single-step gradient descent algorithm, which should, given enough time, be able to find optimal actions, should help shed some light on their relative performance. Furthermore, some effects related to the yawing movement itself, e.g. wear due to rotation, transient yawing loads or a limit on rotation speed, were not considered in this environment. Including these effects can potentially cause the agents to be more conservative in their yaw movements or anticipate future wind conditions. Additionally, both tower-base loads (fore-aft and side-to-side) were considered individually; in reality, they act in the nacelle's rotating frame of reference, which would cause these loads to act in different directions in the world-fixed frame. This can be solved by either projecting both loads back into a world-fixed frame at all times or combining both into a single load value acting on a single tower-base component.

Following the results of this work, we propose four concrete future research opportunities. First of all, the RL training can be adjusted to account for fixed and finite horizons using the sparse-cost maintenance model. This would mean optimising for a given time horizon, possibly given some initial system state. This effectively comes down to finding the control policy that considers both short-term effects (immediate power production optimisation) and long-term effects (postponing breakdowns or increasing them if the resulting power benefit exceeds the costs). Agents trained through this method would likely perform better for the specific horizon under consideration. However, we identify three primary challenges in this type of optimisation. One is the presence of a *credit assignment* problem spawned by the extremely sparse and delayed costs, which can depend on as many as one million actions throughout the horizon. Potential solutions to this can come in the form of *curriculum learning* (gradually moving from dense to sparse costs during training), *reward redistribution* using frameworks like RUDDER [53] or *potential-based reward shaping* [54]. Problem two is the baseline removal trick which needs adjustment to work with sparse costs. Immediate costs are no longer known, making it difficult to provide a state-dependent baseline for costs. As we saw, baseline removal is vital for a stable learning process. The third problem is the clash between short-term, 'tactical' optimisation for power and long-term, 'strategic' optimisation of component degradation (costs). We propose using *temporal abstraction* techniques of Hierarchical Reinforcement Learning (HRL) to tackle this problem. HRL has already seen promising applications in cooperative tasks [55] and multi-agent cooperative systems with delayed rewards [56]. HRL would involve decomposing the agent into a lower-level *tactical* and a higher-level *strategic* meta-controller level. The strategic level decides on which wake steering strategy (conservative, balanced, aggressive, ...) the tactical level should use.

Secondly, we propose casting the problem into a Partially Observable Markov Decision Process (POMDP) framework, as we argue that knowing the full system state is unrealistic. More specifically, the damage states D are impossible to measure in real life as they are not physical variables. Instead, we propose inferring a belief state over the damage states D using some measurement of the fatigue loads. These fatigue loads could, in real life, be either approximated using numerical models or measured using strain sensors. An LSTM, for example, can be used for belief state inference based on observed DEL values sampled from distributions around the simulation's true values to simulate measurement uncertainty or noise. Thirdly, we propose decoupling pitch control from the turbine-level surrogates and adding pitch angles as an additional action for the agent. This would enable it to use joint pitch and yaw control on all turbines, allowing for more control over produced wakes and experienced loads. Prior research has shown that pitch control can contribute to wake control [57–59], providing the agent with more authority to achieve optimal results. Finally, we propose transitioning from a fully centralised learning paradigm to a Centralised Training with Decentralised Execution (CTDE) paradigm to tackle the former's inability to tackle exploding action spaces. This might ultimately enable training on larger farms to achieve better performance that will likely surpass that of the generalised 16-turbine agents. The agent architecture in this work can be easily adjusted to facilitate this by keeping the state encoder but having separate policy decoder networks for each of the agents involved. The value network will remain the same and needs no adjustment, thereby providing information sharing through the shared critic (value) function.

8 conclusion

We investigated the applicability of graph-based deep reinforcement learning to train maintenance-informed wake steering controllers. We use a Centralized Training with Centralized Execution (CTCE) RL framework with PPO and an agent architecture consisting of a Graph Neural Network with encoders and decoders. Our 'greedy ' agent is able to improve power production by 1.5% on average, with peak improvements of up to 20%, though it achieved worse net profit due to increased component failure costs. Both informed and risk-averse agents decreased costs by as much as 50%. The informed agent, whilst doing so, also managed to keep power losses low compared to the baseline policy. Both informed and risk-averse policies reduce the cost of energy from 0.0061 to 0.0025 EUR/kWh, leading to better long-term wind farm profit compared to baseline, random and greedy policies. The informed agent, having produced more energy overall, achieves the highest revenue with increases up to 20%, and is thus the best-performing revenue optimisation controller.

When training on larger farms like Lillgrund (48 turbines) and Horns Rev (80 turbines), the fully centralised training technique starts to run into scalability issues. While the agents can still improve in their respective objectives compared to the baseline, performance decreases as wind farm size increases. However, the 16-turbine agents generalise very well to unseen farms, indicating the agent learns policies based on *relative positions* of turbines in a farm. Despite being trained on a smaller farm, they surpass the performance of the agents specifically trained on the larger farms. Finally, the randomised layout training scheme improves power production on unseen farms. The other agents - informed and risk-averse - however, see nearly identical performance and thus no changes.

The optimality of the trained agents, however, is difficult to judge. Given the high degree of generalizability to various wind conditions, electricity prices and farm layouts, it likely compromises on some level of optimality. Furthermore, the yawing movement itself is not considered in the degradation model despite being an important source of component wear and action constraints. Similarly, the tower base loads likely need adjustment to account for the mismatch between nacelle-fixed and world-fixed frames of reference. We propose future work, including finite-horizon optimisation, partial observability of the environment, decentralised training to tackle scalability issues and the idea of combined pitch and yaw control for more wake steering authority.

In conclusion, we constructed graph-based, maintenance-informed, wind condition-agnostic and topology-agnostic wake steering controllers. Using a reward function that considers both profits through power production and costs through maintenance due to component degradation, we trained an informed agent that maximises long-term wind farm profit and generalises to unseen farm layouts. However, there is room for optimisation in finite-horizon problems.

References

[1] IEA. Renewables 2023 – Analysis, January 2024. URL https://www.iea.org/reports/renewables-2023. 1

[2] Yu-Ting Wu and Fernando Porté-Agel. Modeling turbine wakes and power losses within a wind farm using LES: An application to the Horns Rev offshore wind farm. *Renewable Energy*, 75:945–955, March 2015. ISSN 0960-1481. doi: 10.1016/j.renene.2014.06.019. URL https://www.sciencedirect.com/science/article/pii/S0960148114003590. 1

[3] R. J. Barthelmie, S. T. Frandsen, M. N. Nielsen, S. C. Pryor, P.-E. Rethore, and H. E. Jørgensen. Modelling and measurements of power losses and turbulence intensity in wind turbine wakes at Middelgrunden offshore wind farm. *Wind Energy*, 10(6):517–528, 2007. ISSN 1099-1824. doi: 10.1002/we.238. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/we.238. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/we.238. 1

[4] R. J. Barthelmie, K. Hansen, S. T. Frandsen, O. Rathmann, J. G. Schepers, W. Schlez, J. Phillips, K. Rados, A. Zervos, E. S. Politis, and P. K. Chaviaropoulos. Modelling and measuring flow and wind turbine wakes in large wind farms offshore. *Wind Energy*, 12(5):431–444, 2009. ISSN 1099-1824. doi: 10.1002/we.348. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/we.348. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/we.348. 1

[5] Sang Lee, Matthew Churchfield, Frederick Driscoll, Senu Sirnivas, Jason Jonkman, Patrick Moriarty, Bjorn Skaare, Finn Gunnar Nielsen, and Erik Byklum. Load Estimation of Offshore Wind Turbines. *Energies*, 11(7):1895, July 2018. ISSN 1996-1073. doi: 10.3390/en11071895. URL https://www.mdpi.com/1996-1073/11/7/1895. Number: 7 Publisher: Multidisciplinary Digital Publishing Institute. 2

[6] Elie Kadoche, Sébastien Gourvénec, Maxime Pallud, and Tanguy Levent. MARLYC: Multi-Agent Reinforcement Learning Yaw Control. *Renewable Energy*, 217:119129, November 2023. ISSN 0960-1481. doi: 10.1016/j.renene.2023.119129. URL https://www.sciencedirect.com/science/article/pii/S0960148123010431. 2, 3

[7] Rick Damiani, Scott Dana, Jennifer Annoni, Paul Fleming, Jason Roadman, Jeroen van Dam, and Katherine Dykes. Assessment of wind turbine component loads under yaw-offset conditions. *Wind Energy Science*, 3(1):173–189, April 2018. ISSN 2366-7443. doi: 10.5194/wes-3-173-2018. URL https://wes.copernicus.org/articles/3/173/2018/. Publisher: Copernicus GmbH. 2, 3

[8] Davide Medici. *Experimental studies of wind turbine wakes : power optimisation and meandering*. PhD thesis, KTH Royal Institute of Technology, 2005. URL https://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-598. Publisher: KTH. 3

[9] Michael F. Howland, Sanjiva K. Lele, and John O. Dabiri. Wind farm power optimization through wake steering. *Proceedings of the National Academy of Sciences*, 116(29):14495–14500, July 2019. doi: 10.1073/pnas.1903680116. URL https://www.pnas.org/doi/full/10.1073/pnas.1903680116. Publisher: Proceedings of the National Academy of Sciences. 3

[10] Paul Fleming, Jennifer Annoni, Jigar J. Shah, Linpeng Wang, Shreyas Ananthan, Zhijun Zhang, Kyle Hutchings, Peng Wang, Weiguo Chen, and Lin Chen. Field test of wake steering at an offshore wind farm. *Wind Energy Science*, 2(1):229–239, May 2017. ISSN 2366-7443. doi: 10.5194/wes-2-229-2017. URL https://wes.copernicus.org/articles/2/229/2017/. Publisher: Copernicus GmbH. 3

[11] Paul Fleming, Jennifer King, Katherine Dykes, Eric Simley, Jason Roadman, Andrew Scholbrock, Patrick Murphy, Julie K. Lundquist, Patrick Moriarty, Katherine Fleming, Jeroen van Dam, Christopher Bay, Rafael Mudafort, Hector Lopez, Jason Skopek, Michael Scott, Brady Ryan, Charles Guernsey, and Dan Brake. Initial results from a field campaign of wake steering applied at a commercial wind farm – Part 1. *Wind Energy Science*, 4(2):273–285, May 2019. ISSN 2366-7443. doi: 10.5194/wes-4-273-2019. URL https://wes.copernicus.org/articles/4/273/2019/. Publisher: Copernicus GmbH. 3

[12] National Renewable Energy Laboratory (NREL). FLORIS: FLOw Redirection and Induction in Steady State, . URL https://www.nrel.gov/wind/floris.html. 3

[13] Haohua Zong and Fernando Porté-Agel. Experimental investigation and analytical modelling of active yaw control for wind farm power optimization. *Renewable Energy*, 170:

1228–1244, June 2021. ISSN 0960-1481. doi: 10.1016/j.renene.2021.02.059. URL https://www.sciencedirect.com/science/article/pii/S0960148121002275. 3, 6, 15

[14] P. M. O. Gebraad, F. W. Teeuwisse, J. W. van Wingerden, P. A. Fleming, S. D. Ruben, J. R. Marden, and L. Y. Pao. Wind plant power optimization through yaw control using a parametric model for wake effects—a CFD simulation study. *Wind Energy*, 19(1):95–114, 2016. ISSN 1099-1824. doi: 10.1002/we.1822. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/we.1822. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/we.1822. 3

[15] Pieter Gebraad, Jared J. Thomas, Andrew Ning, Paul Fleming, and Katherine Dykes. Maximization of the annual energy production of wind power plants by optimization of layout and yaw-based wake control. *Wind Energy*, 20(1):97–107, 2017. ISSN 1099-1824. doi: 10.1002/we.1993. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/we.1993. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/we.1993. 3

[16] Paul Stanfel, Kathryn Johnson, Christopher Bay, and Jennifer King. A Distributed Reinforcement Learning Yaw Control Approach for Wind Farm Energy Capture Maximization: Preprint. 2020. URL https://research-hub.nrel.gov/en/publications/a-distributed-reinforcement-learning-yaw-control-approach-for-win-2. 3

[17] Van-Hai Bui, Thai-Thanh Nguyen, and Hak-Man Kim. Distributed Operation of Wind Farm for Maximizing Output Power: A Multi-Agent Deep Reinforcement Learning Approach. *IEEE Access*, 8:173136–173146, 2020. ISSN 2169-3536. doi: 10.1109/ACCESS.2020.3022890. URL https://ieeexplore.ieee.org/document/9189872. Conference Name: IEEE Access. 3

[18] Hongyang Dong, Jincheng Zhang, and Xiaowei Zhao. Intelligent wind farm control via deep reinforcement learning and high-fidelity simulations. *Applied Energy*, 292:116928, June 2021. ISSN 0306-2619. doi: 10.1016/j.apenergy.2021.116928. URL https://www.sciencedirect.com/science/article/pii/S0306261921004086. 3

[19] National Renewable Energy Laboratory (NREL). SOWFA: Simulator fOr Wind Farm Applications, . URL https://www.nrel.gov/wind/nwtc/sowfa.html. 3, 5

[20] Venkata Ramakrishna Padullaparthi, Srinarayana Nagarathinam, Arunchandar Vasan, Vishnu Menon, and Depak Sudarsanam. FALCON- FArm Level CONtrol for wind turbines using multi-agent deep reinforcement learning. *Renewable Energy*, 181:445–456, January 2022. ISSN 0960-1481. doi: 10.1016/j.renene.2021.09.023. URL https://www.sciencedirect.com/science/article/pii/S0960148121013227. 3

[21] Knud A. Kragh and Morten H. Hansen. Load alleviation of wind turbines by yaw misalignment. *Wind Energy*, 17(7):971–982, 2014. ISSN 1099-1824. doi: 10.1002/we.1612. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/we.1612. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/we.1612. 3

[22] Paul Fleming, Pieter M.O. Gebraad, Sang Lee, Jan-Willem van Wingerden, Kathryn Johnson, Matt Churchfield, John Michalakes, Philippe Spalart, and Patrick Moriarty. Simulation comparison of wake mitigation control strategies for a two-turbine case. *Wind Energy*, 18(12):2135–2143, 2015. ISSN 1099-1824. doi: 10.1002/we.1810. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/we.1810. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/we.1810.

[23] Xin Shen, Xiaocheng Zhu, and Zhaohui Du. Wind turbine aerodynamics and loads control in wind shear flow. *Energy*, 36(3):1424–1434, March 2011. ISSN 0360-5442. doi: 10.1016/j.energy.2011.01.028. URL https://www.sciencedirect.com/science/article/pii/S0360544211000296. 3

[24] Shitang Ke, Tongguang Wang, Yaojun Ge, and Hao Wang. Wind-induced fatigue of large HAWT coupled tower–blade structures considering aeroelastic and yaw effects. *The Structural Design of Tall and Special Buildings*, 27(9):e1467, 2018. ISSN 1541-7808. doi: 10.1002/tal.1467. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/tal.1467. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/tal.1467. 3

[25] Min-Soo Jeong, Sang-Woo Kim, In Lee, Seung-Jae Yoo, and K. C. Park. The impact of yaw error on aeroelastic characteristics of a horizontal axis wind turbine blade. *Renewable Energy*, 60:256–268, December 2013. ISSN 0960-1481. doi: 10.1016/j.renene.2013.05.014. URL https://www.sciencedirect.com/science/article/pii/S0960148113002590. 3

[26] Mou Lin and Fernando Porté-Agel. Power Maximization and Fatigue-Load Mitigation in a Wind-turbine Array by Active Yaw Control: an LES Study. *Journal of Physics: Conference Series*, 1618(4):042036, September 2020. ISSN 1742-6596. doi: 10.1088/1742-6596/1618/4/042036. URL https://dx.doi.org/10.1088/1742-6596/1618/4/042036. Publisher: IOP Publishing. 3

[27] Brandon L. Ennis, Jonathan R. White, and Joshua A. Paquette. Wind turbine blade load characterization under yaw offset at the SWiFT facility. *Journal of Physics: Conference Series*, 1037(5):052001, June 2018. ISSN 1742-6596. doi: 10.1088/1742-6596/1037/5/052001. URL https://dx.doi.org/10.1088/1742-6596/1037/5/052001. Publisher: IOP Publishing. 3

[28] Ervin Bossanyi. Combining induction control and wake steering for wind farm energy and fatigue loads optimisation. *Journal of Physics: Conference Series*, 1037(3):032011, June 2018. ISSN 1742-6596. doi: 10.1088/1742-6596/1037/3/032011. URL https://dx.doi.org/10.1088/1742-6596/1037/3/032011. Publisher: IOP Publishing. 3

[29] Mike T. van Dijk, Jan-Willem van Wingerden, Turaj Ashuri, and Yaoyu Li. Wind farm multi-objective wake redirection for optimizing power production and loads. *Energy*, 121: 561–569, February 2017. ISSN 0360-5442. doi: 10.1016/j.energy.2017.01.051. URL https://www.sciencedirect.com/science/article/pii/S0360544217300518. 3

[30] Ruiyang He, Hongxing Yang, and Lin Lu. Optimal yaw strategy and fatigue analysis of wind turbines under the combined effects of wake and yaw control. *Applied Energy*, 337: 120878, May 2023. ISSN 0306-2619. doi: 10.1016/j.apenergy.2023.120878. URL https://www.sciencedirect.com/science/article/pii/S0306261923002428. 3

[31] International Energy Agency (IEA). Wind energy generation systems - Part 3-1: Design requirements for fixed offshore wind turbines, October 2019. 4, 6, 7

[32] James Carroll, Alasdair McDonald, and David McMillan. Failure rate, repair time and unscheduled O&M cost analysis of offshore wind turbines. *Wind Energy*, 19(6):1107–1119, 2016. ISSN 1099-1824. doi: 10.1002/we.1887. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/we.1887. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/we.1887. 4, 11

[33] Cuong Dao, Behzad Kazemtabrizi, and Christopher Crabtree. Wind turbine reliability data review and impacts on levelised cost of energy. *Wind Energy*, 22(12):1848–1871, 2019. ISSN 1099-1824. doi: 10.1002/we.2404. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/we.2404. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/we.2404. 4, 11

[34] Mads M. Pedersen, Alexander Meyer Forsting, Paul van der Laan, Riccardo Riva, Leonardo A. Alcayaga Romàn, Javier Criado Risco, Mikkel Friis-Møller, Julian Quick, Jens Peter Schøler Christiansen, Rafael Valotta Rodrigues, Bjarke Tobias Olsen, and Pierre-Elouan Réthoré. PyWake 2.5.0: An open-source wind farm simulation tool. *DTU Wind, Technical University of Denmark*, February 2023. URL https://gitlab.windenergy.dtu.dk/TOPFARM/PyWake. 5

[35] National Renewable Energy Laboratory (NREL). OpenFAST/openfast, June 2024. URL https://github.com/OpenFAST/openfast. original-date: 2016-08-31T20:07:10Z. 5, 6

[36] Gregory Duthé, Francisco de Nolasco Santos, Imad Abdallah, Pierre-Élouan Réthore, Wout Weijtjens, Eleni Chatzi, and Christof Devriendt. Local flow and loads estimation on wake-affected wind turbines using graph neural networks and PyWake. *Journal of Physics: Conference Series*, 2505(1):012014, May 2023. ISSN 1742-6596. doi: 10.1088/1742-6596/2505/1/012014. URL https://dx.doi.org/10.1088/1742-6596/2505/1/012014. Publisher: IOP Publishing. 6, 8

[37] IEA Wind Task 37. IEAWindTask37/IEA-3.4-130-RWT. URL https://github.com/IEAWindTask37/IEA-3.4-130-RWT. 6

[38] NREL. ROSCO, June 2024. URL https://github.com/NREL/ROSCO. original-date: 2019-11-08T15:47:14Z. 6

[39] Rad Haghi and Curran Crawford. Data-driven surrogate model for wind turbine damage equivalent load. *Wind Energy Science Discussions*, pages 1–34, December 2023. doi: 10.5194/wes-2023-157. URL https://wes.copernicus.org/preprints/wes-2023-157/. Publisher: Copernicus GmbH. 6

[40] Nikolay Dimitrov. Surrogate models for parameterized representation of wake-induced loads in wind farms. *Wind Energy*, 22(10):1371–1389, 2019. ISSN 1099-1824. doi: 10.1002/we.2362. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/we.2362. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/we.2362. 6, 7

[41] Majid Bastankhah and Fernando Porté-Agel. A new analytical model for wind-turbine wakes. *Renewable Energy*, 70:116–123, October 2014. ISSN 0960-1481. doi: 10.1016/j.renene.2014.01.002. URL https://www.sciencedirect.com/science/article/pii/S0960148114000317. 8

[42] Erik Quaeghebeur, René Bos, and Michiel B. Zaaijer. Wind farm layout optimization using pseudo-gradients. *Wind Energy Science*, 6(3):815–839, June 2021. ISSN 2366-7443. doi: 10.5194/wes-6-815-2021. URL https://wes.copernicus.org/articles/6/815/2021/. Publisher: Copernicus GmbH. 8

[43] Ángel Jiménez, Antonio Crespo, and Emilio Migoya. Application of a LES technique to characterize the wake deflection of a wind turbine in yaw. *Wind Energy*, 13(6):559–572, 2010. ISSN 1099-1824. doi: 10.1002/we.380. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/we.380. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/we.380. 8

[44] Guohao Li, Chenxin Xiong, Ali Thabet, and Bernard Ghanem. DeeperGCN: All You Need to Train Deeper GCNs, June 2020. URL http://arxiv.org/abs/2006.07739. arXiv:2006.07739 [cs, stat]. 8

[45] Ørsted. Offshore wind data. URL https://orsted.com/en/what-we-do/renewable-energy-solutions/offshore-wind/offshore-wind-data. 9

[46] Nord Pool Group. Nord Pool | Day-ahead prices. URL https://data.nordpoolgroup.com/auction/day-ahead/prices?deliveryDate=latest&currency=EUR&aggregation=Hourly&deliveryAreas=AT. 10

[47] International Energy Agency (IEA). IEA Wind TCP Task 37, 2016. URL https://iea-wind.org/task37/. 11

[48] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-Dimensional Continuous Control Using Generalized Advantage Estimation, October 2018. URL http://arxiv.org/abs/1506.02438. arXiv:1506.02438 [cs]. 12

[49] Daniel Seita. Going Deeper Into Reinforcement Learning: Fundamentals of Policy Gradients, March 2017. URL https://danieltakeshi.github.io/2017/03/28/going-deeper-into-reinforcement-learning-fundamentals-of-policy-gradients/. 12

[50] A. Rupam Mahmood, Dmytro Korenkevych, Gautham Vasan, William Ma, and James Bergstra. Benchmarking Reinforcement Learning Algorithms on Real-World Robots. In *Proceedings of The 2nd Conference on Robot Learning*, pages 561–591. PMLR, October 2018. URL https://proceedings.mlr.press/v87/mahmood18a.html. ISSN: 2640-3498. 12

[51] Neil De La Fuente and Daniel A. Vidal Guerra. A Comparative Study of Deep Reinforcement Learning Models: DQN vs PPO vs A2C, July 2024. URL http://arxiv.org/abs/2407.14151. arXiv:2407.14151 [cs]. 12

[52] Simone Tamaro, Filippo Campagnolo, and Carlo L. Bottasso. On the power and control of a misaligned rotor – beyond the cosine law. *Wind Energy Science*, 9(7):1547–1575, July 2024. ISSN 2366-7443. doi: 10.5194/wes-9-1547-2024. URL https://wes.copernicus.org/articles/9/1547/2024/. Publisher: Copernicus GmbH. 15

[53] Jose A. Arjona-Medina, Michael Gillhofer, Michael Widrich, Thomas Unterthiner, Johannes Brandstetter, and Sepp Hochreiter. RUDDER: Return Decomposition for Delayed Rewards, September 2019. URL http://arxiv.org/abs/1806.07857. arXiv:1806.07857 [cs, math, stat]. 19

[54] Andrew Y. Ng, Daishi Harada, and Stuart J. Russell. Policy Invariance Under Reward Transformations: Theory and Application to Reward Shaping. In *Proceedings of the Sixteenth International Conference on Machine Learning*, ICML '99, pages 278–287, San Francisco, CA, USA, June 1999. Morgan Kaufmann Publishers Inc. ISBN 978-1-55860-612-8. 19

[55] Mohammad Ghavamzadeh, Sridhar Mahadevan, and Rajbala Makar. Hierarchical multi-agent reinforcement learning. *Autonomous Agents and Multi-Agent Systems*, 13(2):197–229, September 2006. ISSN 1573-7454. doi: 10.1007/s10458-006-7035-4. URL https://doi.org/10.1007/s10458-006-7035-4. 19

[56] Hongyao Tang, Jianye Hao, Tangjie Lv, Yingfeng Chen, Zongzhang Zhang, Hangtian Jia, Chunxu Ren, Yan Zheng, Zhaopeng Meng, Changjie Fan, and Li Wang. Hierarchical Deep Multiagent Reinforcement Learning with Temporal Abstraction, July 2019. URL http://arxiv.org/abs/1809.09332. arXiv:1809.09332 [cs]. 19

[57] Jaejoon Lee, Eunkuk Son, Byungho Hwang, and Soogab Lee. Blade pitch angle control for aerodynamic performance optimization of a wind farm. *Renewable Energy*, 54: 124–130, June 2013. ISSN 0960-1481. doi: 10.1016/j.renene.2012.08.048. URL https://www.sciencedirect.com/science/article/pii/S0960148112005186. 20

[58] Deepu Dilip and Fernando Porté-Agel. Wind Turbine Wake Mitigation through Blade Pitch Offset. *Energies*, 10(6):757, June 2017. ISSN 1996-1073. doi: 10.3390/en10060757. URL https://www.mdpi.com/1996-1073/10/6/757. Number: 6 Publisher: Multidisciplinary Digital Publishing Institute.

[59] Joeri A. Frederik, Bart M. Doekemeijer, Sebastiaan P. Mulders, and Jan-Willem van Wingerden. The helix approach: Using dynamic individual pitch control to enhance wake mixing in wind farms. *Wind Energy*, 23(8):1739–1751, 2020. ISSN 1099-1824. doi: 10.1002/we.2513. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/we.2513. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/we.2513. 20

B

'Master' DEL Convergence Analysis

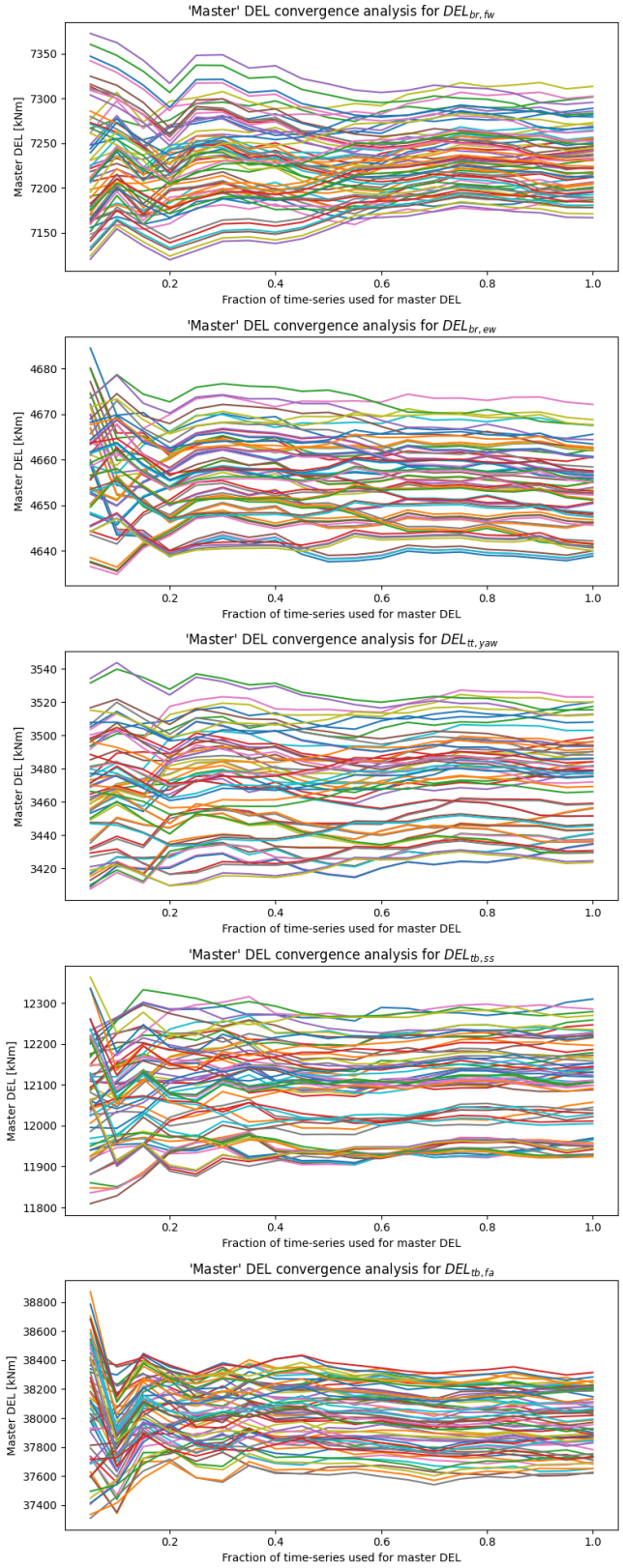

Figure B.1: Convergence analysis of 'master' lifetime DEL; data comes from 25-year-long simulated time series, of four simulations with 16 turbines each.

C

Fatigue curve fitting with gradient descent

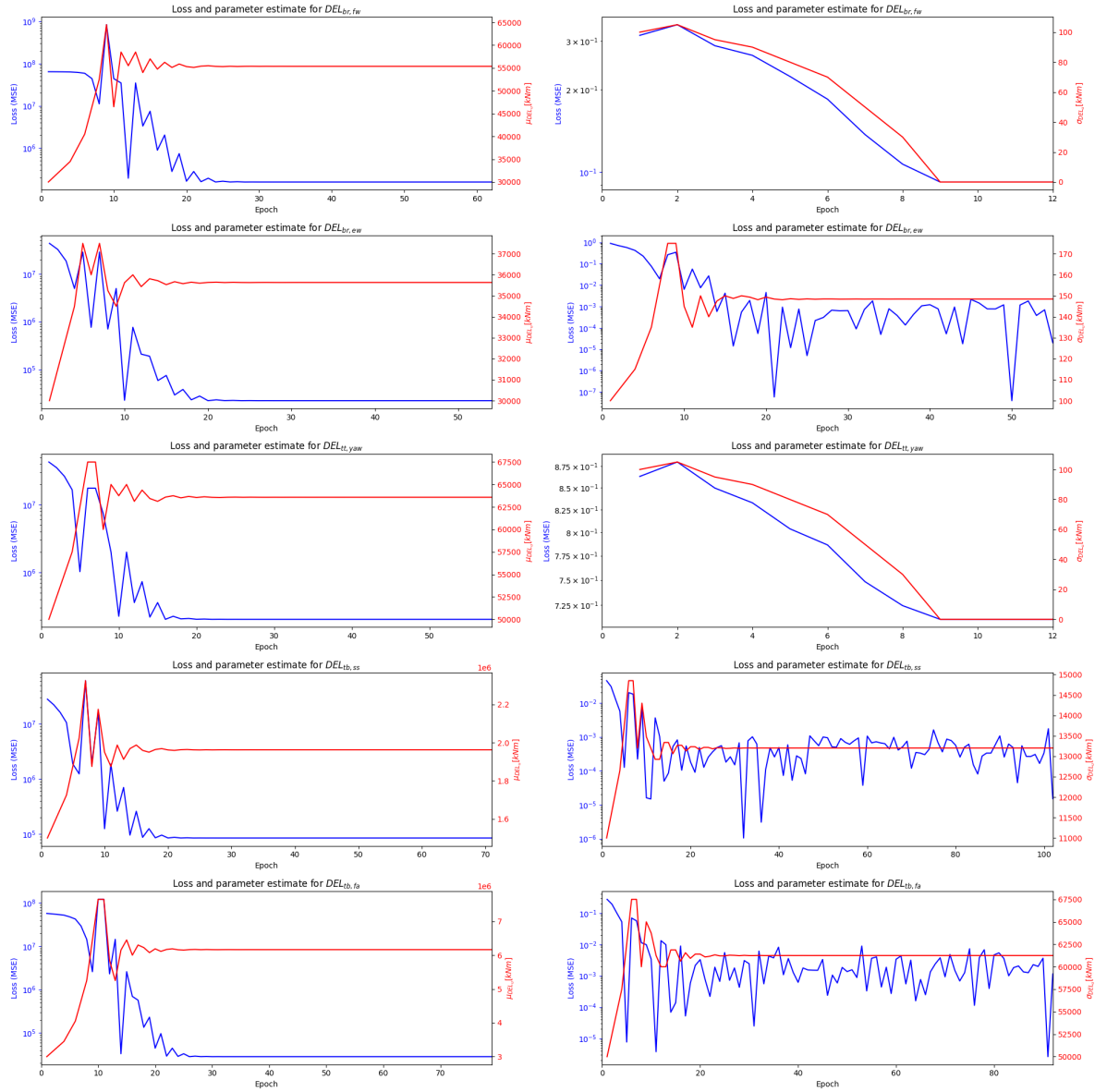

Figure C.1: 'Training' curves of Nelder-Mead gradient descent for finding the mean and standard deviation of the DEL_u distributions; values fitted for a target lifetime of 22 years with standard deviation of 1 year.

D

Convergence study of simulation length vs DEL

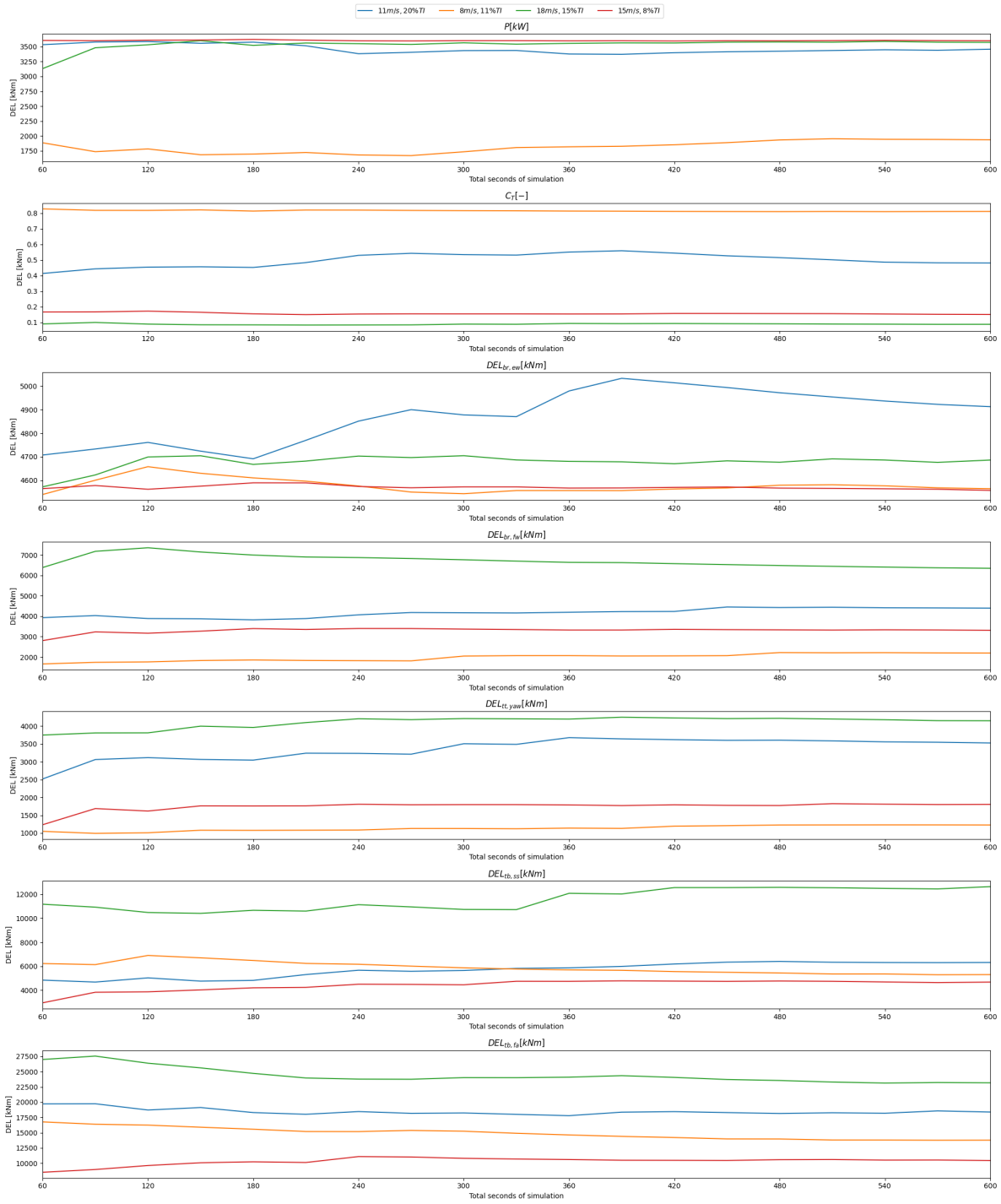

Figure D.1: Convergence study of simulation time versus the calculated DEL. Four different simulations were run for 600 seconds each and the DEL was calculated for increasing fractions of the total simulation data. Power and thrust coefficient are also shown.