# Wind Farm Yaw Optimization and Real-Time Control using Graph Neural Networks

Rockey Abram Hester V

CMU-RI-TR-21-49

August 19, 2021


The Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA

**Thesis Committee:**
Dr. Matthew Travers, *chair*
Professor Sebastian Scherer
Julian Whitman

*Submitted in partial fulfillment of the requirements for the degree of Master of Science in Robotics.*

# Abstract

In order to mitigate the effects of climate change, it is important to maximize the power output of our wind energy resources. Wind farm control is typically greedy, with all turbines facing directly into the wind. Due to wake interactions, this leads to reductions in power for downwind turbines for some wind conditions. An alternative is wake steering control, which involves misaligning the yaws of upwind turbines to deflect their wakes away from downwind turbines. This reduces the power generated by the upwind turbines but increases the power output of the farm as a whole. Tools such as FLORIS exist to model wakes and compute optimal wake steering yaw configurations for a given wind farm and wind conditions, but their runtime scales quadratically in the number of turbines, which prevents real-time control for large farms or quickly changing wind conditions.

Using YawGNN, a graph neural network with wind turbines as nodes and wake interactions between turbines as edges, we learn to approximate the FLORIS model. YawGNN is able to find yaw configurations that outperform the greedy solution and are comparable to FLORIS in linear time for wind farms with the same number of turbines as it was trained on. Relational inductive biases in graph neural networks also enable generalization to unseen wind farms, including those with different numbers of turbines. YawGNN is able to accurately estimate the power output of wind farms with ten times the number of turbines it was trained on, as well as find yaw configurations for these examples that are on average of higher quality than the greedy solution in linear time. Finally, we show that YawGNN outperforms FLORIS and is applicable across a greater range of farm sizes and wind direction rates of change when used as the model in a model predictive controller for real-time, wake steering control.

# Acknowledgments

First, I would like to thank my advisor, Dr. Matthew Travers, for his guidance in the development of this thesis and confidence in the potential of wind farm graph neural networks. I would also like to thank the remaining members of my thesis committee, Professor Sebastian Scherer and Julian Whitman, for their feedback and support, as well as B.J. Fecich for her invaluable support on the program side and Casper Lyhne for his lifesaving domain knowledge.

Next, I could never thank my friends and family enough for their love and support as I have pursued my education over the years. Thanks especially to George Cazenavette, Bill Peebles, Roshan Pradhan, and Brandon Wang for their very direct support on this thesis, and to my parents for welcoming me back home during a pandemic.

Finally, I have to thank my dog, Gus, for snapping me out of my graduate student sleep schedule and always being there when I needed a break from work or an excuse to get out of the house.

# Contents

*When this dissertation is viewed as a PDF, the page header is a link to this Table of Contents.*

# List of Figures

# List of Tables

# Chapter 1

# Introduction

## 1.1 Background

### 1.1.1 Climate Change and Renewable Energy

According to the Intergovernmental Panel on Climate Change (IPCC), if current warming trends continue, then we will likely reach global warming of 1.5°C above pre-industrial levels sometime between 2030 and 2052. Beyond just increases in average temperature, this level of warming could also result in greater occurrences of extreme heat waves, extreme rainfall, and extreme drought, which could have devastating effects on human health, national security, and the global economy. Effects from reaching global warming of 2°C are even more severe. In order to prevent surpassing 1.5°C of warming, global $CO_2$ emissions must decrease by around 45% from 2010 levels by 2030 and reach net zero around 2050. In all viable paths to reaching this goal, the portion of energy generated by renewable energy sources must increase to around 60% by 2050 [34].

While the share of renewable energy is growing quickly, there is still a long way to go to reach these goals. In 2016, the global share of renewable energy was 24%. In 2018, 17% of the United States' energy was generated from renewable sources. Wind energy makes up a large portion - 6.6% in the US and 5% globally in 2018 [12].

Wind energy is also growing rapidly, with global wind power capacity growing from 591 GW in 2018 to 743 GW in 2020, but, according to the Global Wind Energy

Figure 1.1: Laurel Mountain Wind Farm in West Virginia. If the wind flows over the ridge (the blue arrow in the image), then the turbines will yaw to align with the wind to generate maximum power. If instead the wind flows along the ridge (the red arrow in the image), then yawing to align with the wind causes the turbines to form a column, resulting in wake interactions that reduce power greatly. Source: John Terry


Figure 1.2: Two turbines in serial configuration relative to the wind, or aligned in a column, with wakes shown

Council, wind power capacity needs to be growing three times faster between now and 2030 in order to reach net zero emissions by 2050 [8]. However, installing new turbines is not the only way to increase the energy we receive from wind power. This work is concerned with how to maximize the power output of the wind farms we already have.

### 1.1.2 Wind Farm Modeling and Control

Wind farms are usually controlled by pointing every turbine in the direction of the incident wind, which is defined as the steady state wind direction for the wind that is flowing into, or is incident on, the wind farm. See Figure 1.1. This maximizes the effective rotor area, but it is also a greedy and sometimes suboptimal solution. This is because wind turbines interact with each other via their wakes. For example, if two turbines are in a serial configuration relative to the wind, or one behind the other as in Figure 1.2, then the wake of the upwind turbine acts to slow down the wind for the downwind turbine [28]. This is a large problem as the power generated by a wind turbine is proportional to the cube of the wind speed. On one offshore wind farm, these kinds of wake losses reduced energy production by around 20%. Wake losses also increase the variability of power output and, thus, the need for energy storage and less variable energy production methods like fossil fuels. Spacing turbines far enough apart to eliminate wake losses entirely would increase cost prohibitively, so this problem must be managed via more sophisticated control [18].

In order to do more sophisticated control, we must model wind farm wake interactions. Common approaches to modeling wind farms include high-fidelity, finite-element and finite-volume computational fluid dynamics (CFD) simulations, analytical wake models like the Jensen model [19], and parametric models like FLORIS

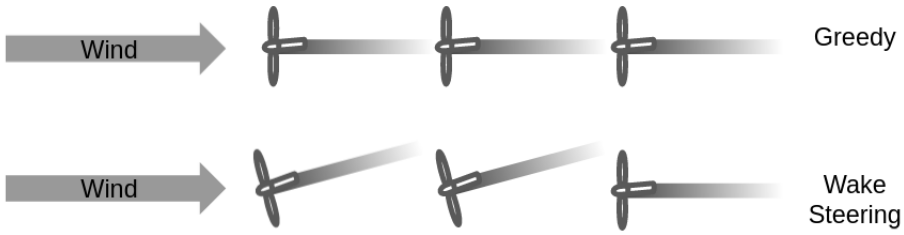



Figure 1.3: A comparison between greedy and wake steering control of wind farms. With greedy control, all turbines yaw to align with the incident wind. In wake steering control, upwind turbines misalign their yaws from the incident wind in order to deflect their wakes away from downwind turbines.

[27] that are built from a variety of analytical wake models. While CFD enables highly accurate forward simulation, it is not ideal for model-based control because it is computationally expensive and not easily amenable to differentiation - one can take finite differences, but this very quickly becomes computationally prohibitive as well. Analytical and parametric wake models have been used to study wake steering control, which is the idea that you can increase a farm's overall power output by intentionally misaligning the yaws of upwind turbines in order to deflect their wakes away from downwind turbines. See Figure 1.3. However, most work thus far has focused on simple serial farm configurations with very complex analytical models in order to maintain differentiability [18]. FLORIS has been used to find optimal yaw configurations for simulated farms with hundreds of turbines, but the runtime is quadratic in the number of turbines, which prevents real-time control for large farms or quickly changing wind conditions. In addition, anayltical models and FLORIS are unable to capture some important real-world aerodynamic effects like speed-up and blockage [7]. Thus, there is a need for a lightweight, differentiable forward-dynamics model, and recent advances in the fields of robotics and machine learning enable us to both build such a model and leverage it for real-time control.

### 1.1.3 Machine Learning and Optimization

Currently in robotics and machine learning, control policies for complex systems are commonly derived by defining an optimization problem with a cost function and policy parameters, and then using some variation of gradient descent to search

the state space for the optimal policy parameters. Over the last decade, neural networks in particular have become extremely popular as models for complex systems in part due to their suitability to automatic differentiation [5], which enables efficient training and optimization with respect to policy parameters. Meanwhile, advances in computing hardware have enabled the use of large quantities of data for training.

In this work, we focus specifically on graph neural networks (GNNs), which are a class of neural networks that are specialized to operate on graph-structured data. For example, we can represent a wind farm as a graph with turbines as nodes and wake interactions as edges as shown in Figure 1.4. Then, our GNN model would consist of multiple individual traditional neural networks, or multi-layer perceptrons (MLPs), that take as input the node, edge, and global features of the graph separately and in sequence rather than as one large feature vector. Computation in a GNN is shown graphically in Figure 1.6 and covered in more detail in Chapter 2. By posing our learning problem in this way, we encode relational inductive biases into the architecture [3], which allow us to learn to reason about how a wind turbine interacts with other wind turbines in general as opposed to how, for example, a wind farm with five turbines operates. Put slightly differently, a GNN does not treat each turbine as if its behavior is unique and attempt to learn the physics of how every single turbine interacts with every single other turbine. Instead it uses data from every wind turbine to learn a model of the physics of a wind turbine in general and applies this knowledge across the farm. This also enables combinatorial generalization - since a single GNN can operate on different graph sizes, we can use a wind farm trained on wind farms of relatively small size to infer on wind farms with larger numbers of turbines. Another benefit of graph computation is that it is decentralized. The graph representation captures the natural sparsity present in wind farm systems as shown in Figure 1.5, resulting in an increase in computational efficiency because computation is only done between turbines that interact.

Graph neural networks have been applied to wind farms before, but focused mainly on power estimation by approximating the Reynolds-Averaged Navier-Stokes equations [6] or FLORIS [31]. To the author's knowledge, graph neural networks have never been applied to wind farms for the purpose of yaw optimization for wake steering control.

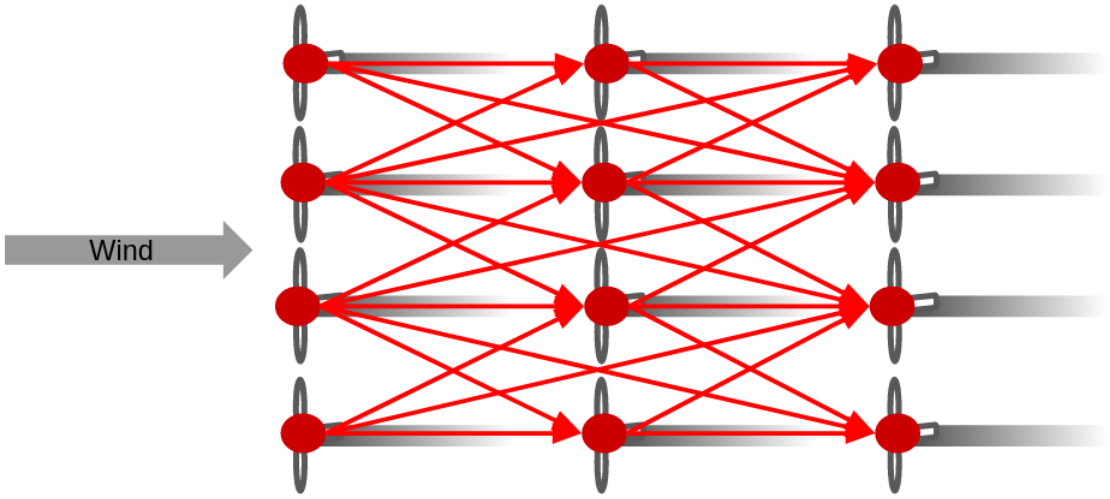



Figure 1.4: Wind farm graph with turbines as nodes and wake interactions as edges

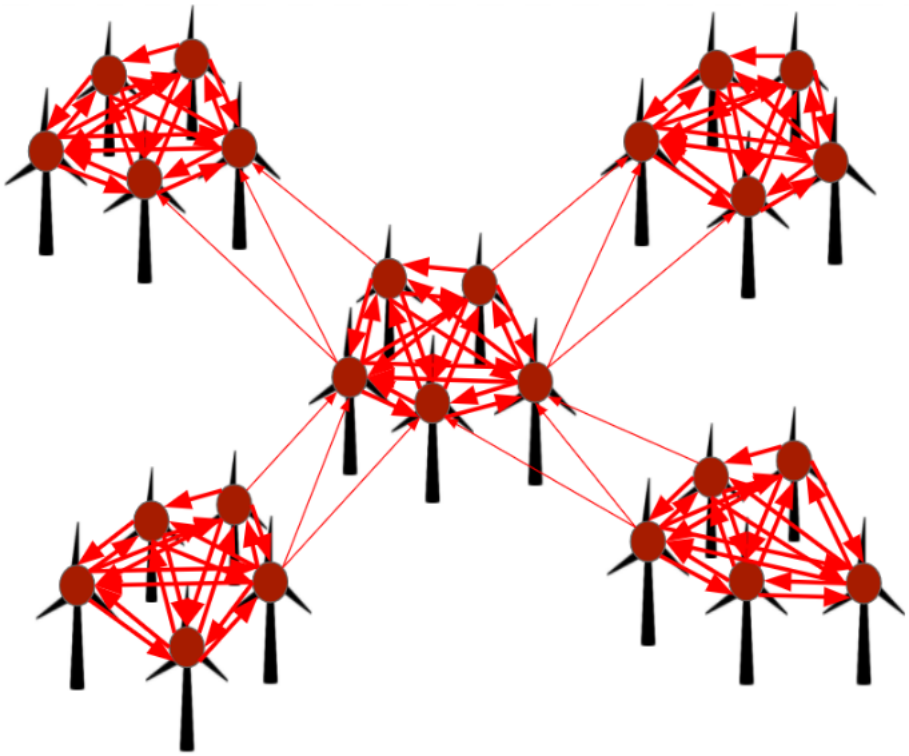
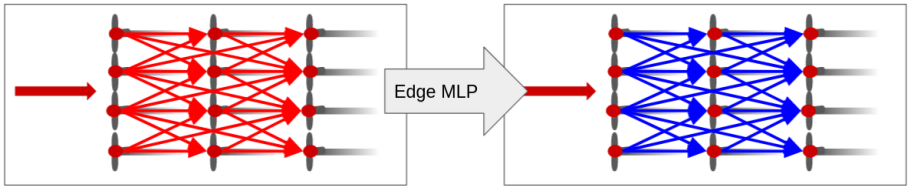
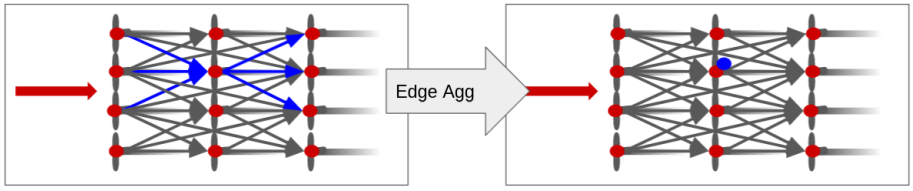


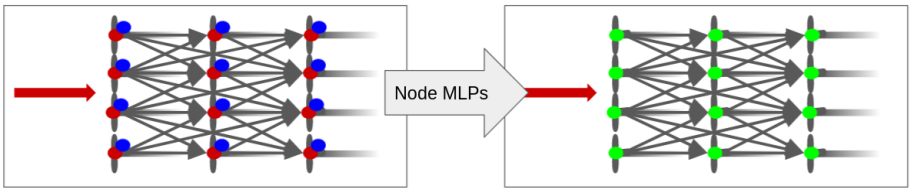
Figure 1.5: Wind farm graph for a sparse wind farm system. The clusters of turbines and interactions between clusters are captured by the edges of the graph.



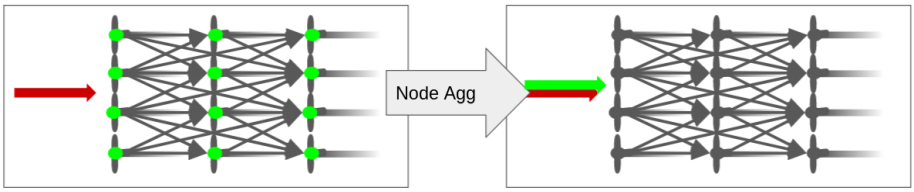

(a) Update the edge features by passing them through the edge MLP



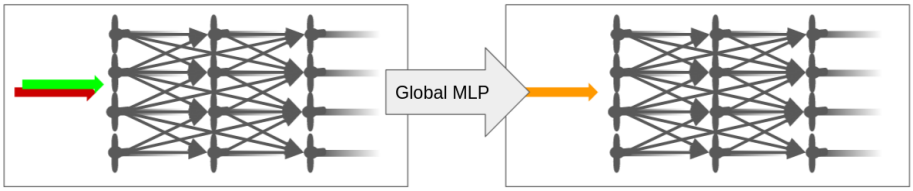

(b) Aggregate updated edge features at nodes, typically via a mean or sum



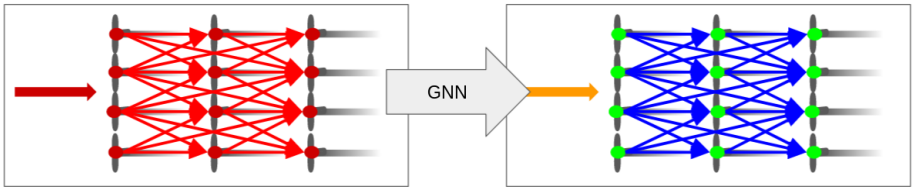

(c) Update the node features by passing the node features and aggregated edge features through the node MLPs




(d) Aggregate updated node features globally, typically via a mean or sum




(e) Update the global features by the passing the global features and aggregated node features through the global MLP




(f) Summary of previous steps - a GNN uses message passing along edges and node and global aggregations to update the graph's edge, node, and global features

Figure 1.6: Computation in a message passing graph neural network with edge, node, and global multi-layer perceptrons (MLPs)

## 1.2 Related Work

### 1.2.1 Wind Farm Control

Pao and Johnson summarize the dynamics and control of wind farms in [28]. Wind farms have three controllable parameters - yaw, blade pitch, and generator torque. Due to gyroscopic forces, wind turbines typically do not yaw faster than 1 degree per second. As a result, yaw controllers are typically simple and do not receive much research interest. Blade pitch and generator torque are able to operate on much higher frequencies and are the primary actuators in traditional wind farm control. Blade pitch rates range from between 18 degrees per second for smaller turbines to 8 degrees per second for larger turbines, while generator torque has a time constant on the order of ten times the speed of the rotor.

Wind farm control is divided into three regimes depending on the wind speed. For low wind speeds, called Region 1, the turbine is not run at all because the losses in the system are high relative to the power that could be extracted. For high wind speeds, called Region 3, the power extracted from the wind is kept constant by holding the torque constant. This is to prevent surpassing limits on electrical and mechanical loads. For speeds in between, Region 2, the wind turbine attempts to maximize its power by varying the blade pitch to track an optimal ratio between the speed of the blade tips and the wind speed, or tip-speed ratio. The exact wind speeds separating Regions 1, 2, and 3 depend on the turbine in question [28].

In [21], the National Renewable Energy Laboratory (NREL) details the specifications of a theoretical 5-MW wind turbine that is representative of various industry wind turbines called the "NREL offshore 5-MW baseline wind turbine." This turbine was developed for use as a reference in studies in wind farm modeling, design, control, and evaluation and is used in all of the experiments in this work. It has a nominal yaw rate of 0.3 degrees per second. Region 1 is defined as wind speeds less than 3 meters per second, and Region 3 is defined as wind speeds greater than 11.5 meters per second.

### 1.2.2 Wind Farm Modeling

There are three primary modeling paradigms for wind farms - analytic wake models, high-fidelity computational fluid dynamics (CFD) simulations, and parametric wake models. Analytic wake models are derived from first principles and describe various aspects of the wakes generated by turbines. Analytic wake models include the Jensen model [19], the Jiménez model [20], the multizone model [14], the Gaussian model [26], and the curled model [25]. Analytic models are naturally limited in their ability to fully describe wake interactions between wind turbines, but are appealing for their speed. Developing new analytic wake models is still a very active area of research.

High-fidelity CFD simulations such as SOWFA [10][11] are the industry standard for modeling wind farms, but require supercomputers and multiple days to run, which makes them unsuited for use in control and optimization.

FLORIS (FLOw Redirection and Induction in Steady-state) [27][13] is a parametric wake model that implements the aforementioned analytic wake models in a toolbox for the purpose of studying wind farm design and control. These wake models are then parameterized using 15 parameters estimated using turbine power production data describing the turbine power, wake deflection, wake expansion, and wake velocity. While FLORIS ultimately has similar concerns with generalization to the real world as analytic wake models, its combination of first principles methods and data makes it appealing for studying wake steering control.

### 1.2.3 Wake Steering Control

The development of FLORIS and new analytic wake models, along with the fact that larger wind turbines have become more common and suffer more severely from wake interaction losses, have led to an increased amount of interest in wake steering control [18].

Park and Law solve the cooperative control problem by using a Bayesian Ascent algorithm to maximize an analytic wind farm power function [30]. This method is able to increase power monotonically with each iteration. To deal with the increase in computational complexity as the number of turbines in the farm increases, [30] exploits the natural sparsity in wind farm systems by grouping turbines into clusters based on their level of interaction and solves multiple problems in parallel. This

method is limited by its use of a purely analytic power function. In addition, while the notion of exploiting sparsity is valuable, clustering turbines inevitably omits important interactions between clusters as the farm layout becomes more complex than just a grid.

Howland et. al. develop a wake steering control method that maximizes an analytic power function using gradient ascent and demonstrates it on a wind farm in Alberta, Canada [18]. Applying this wake steering control increased power for wind directions with increased wake losses by between 7% and 13% for average wind speeds and as much as 47% for low wind speeds. It also reduced the standard deviation of the power production in these detrimental wind directions by as much as 72%. This method is limited by its reliance on historical data for the optimization process, making it too site-specific for use as a general tool. While most methods have not been demonstrated on operational wind farms, it is important to note this wind farm consisted of only 6 turbines aligned in a column.

Gebraad et. al. [13] use an early version of FLORIS based on the Jensen [19] and Jiménez [20] wake models as the internal model in a wind farm controller. It uses a model-based variation of a model-free, game theoretic optimization approach introduced in [24] to compute optimal yaw configurations and uses SOWFA to validate that the resulting yaws increase power production for the farm as well as reduce load on the turbines.

Bay et. al. [4] extend the results from [13] by adding the curled wake model [25] to FLORIS and demonstrating that optimal yaws found using FLORIS with the curled wake model result in larger increases in power production than yaws found using previous wake models such as FLORIS with the Gaussian wake model [26]. The yaws found using this method also differ qualitatively from previous methods, decreasing in magnitude as you move downstream. The major limitation of this method is that the optimization algorithm used, sequential least squares quadratic programming (SLSQP), is quadratic in time complexity. For farms with 100 turbines, computing an optimal yaw configuration can take on the order of an hour, which greatly reduces the method's applicability to real-time wind farm operations where the wind can change direction in minutes.

### 1.2.4 Graph Neural Networks

Battaglia et. al. [3] review the development of graph neural networks (GNNs) and argue that combinatorial generalization, or the ability for a model to generalize to unseen combinations of seen elements, which GNNs are well suited for, should be a key focus of future research. GNNs were first introduced in [32] and are a class of neural networks that are specialized to learn relationships and perform computations over graph-structured data by incorporating relational inductive biases into the architecture of the model. An inductive bias is a set of assumptions that cause a learning algorithm to favor some solutions over others using some outside knowledge not found in the data. A relational inductive bias, then, is an inductive bias that causes a learning process to enforce, rather than infer, relationships between discrete entities. While convolutional and recurrent neural network layers both encode relational inductive biases, the relational inductive bias encoded by graphs are much stronger [3].

GNNs enforce relational inductive biases in three primary ways [3].

1. Since GNNs take graphs as input, the input rather than the architecture determines whether or not two entities interact.
2. Graphs are represented as sets of nodes, edges, and global features, and since sets are invariant to permutations, so, too, are GNNs.
3. Since a GNN's per-edge and per-node functions are applied to all nodes and edges in the same way, GNNs are predisposed to combinatorial generalization

GNNs' aptitude for combinatorial generalization has become well-supported in the literature. Battaglia et. al. [2] showed that GNNs trained to make one-step predictions for complicated systems such as n-body problems, rigid-body collisions, and non-rigid dynamics are able to accurately simulate thousands of steps into the future, as well as generalize to systems with different numbers and layouts of entities with no additional training. Hamrick et. al. [16] showed that a GNN trained using reinforcement learning to glue blocks together to stabilize a tower could generalize to unseen numbers of blocks. Wang et. al. [35] used a GNN to match state-of-the-art performance in MuJoCo environments and demonstrate significantly more generalizable policies than other state-of-the-art methods.

Additionally, GNNs trained to approximate particle-based fluid representations

have shown the ability to transfer to the modeling and manipulation of real fluids. Schenck and Fox [33] developed Smooth Particle Networks (SPNets) - a fully differentiable fluid dynamics framework which enables efficient forward dynamics simulation of physical interactions among particles. The trained model is then used in trajectory optimization to control and manipulate fluids. Li et. al. [22] similarly have developed Dynamic Particle Interaction Networks (DPI-Nets) to model and manipulate rigid bodies and deformable solids in addition to fluids.

### 1.2.5 Wind Farm Graph Neural Networks

This work is a direct extension of Physics-induced graph neural networks (PGNNs) [31], which addresses the limitations of the aforementioned wind farm modeling approaches by training a message passing GNN [15] to approximate the FLORIS model for arbitrary turbine layouts and wind conditions. Wind farms are represented as graphs with wind turbines as nodes and interactions between turbines as edges. A key contribution is the use of an analytic wake model parameterized by the edge features of the graph (the downstream and radial distances between turbines that share an edge) [29] as a weighting function for the edge features during the edge update portion of the GNN's computation. This "physics-induced bias" is comparable in function to attention. It is shown that the version of the model including the physics-induced bias outperforms a model with a purely data-driven edge weighting function in estimating the power produced by unseen wind farm configurations with size both in and out of the training distribution. The trained model is then used as a differentiable surrogate wake model for optimizing wind farm layouts. However, wind turbine yaw and wake deflections are not modeled, so PGNN is unable to be used for wake steering control.

Bleeg [6] also demonstrates GNNs being used to model wind farm wake interactions. However, in this work the GNN is trained to approximate the Reynolds-Averaged Navier-Stokes (RANS) model (a high-fidelity flow model) rather than FLORIS in order to model more complicated aerodynamic effects like blockage [7] more efficiently than typical high-fidelity approaches. The trained GNN achieves good agreement with RANS, but extensions to be able to optimize wind farm layouts or yaws for maximum power production were left to future work.

## 1.3 Research Question

This work aims to answer four primary questions:

1. **Can we use a GNN to approximate the FLORIS wake modeling utility for arbitrary turbine layout, turbine yaws, wind direction, and wind speed?**

   Park and Park showed that a GNN can approximate the FLORIS wake modeling utility for arbitrary turbine layout, wind direction, and wind speed in [31]. The ability to additionally model yaw is essential to doing control.

2. **Can a trained wind farm GNN be used to compute optimal yaw configurations more efficiently than FLORIS without sacrificing solution quality?**

3. **How well is a trained wind farm GNN able to generalize to farms of larger size than it was trained on - both in power estimation and yaw optimization?**

4. **How well does a trained wind farm GNN perform when used as the model in a model predictive controller for real-time wake steering control of wind farms?**

   Additionally, are there farm sizes (in number of turbines) or wind direction rates of change where YawGNN is able to be used for real-time control but FLORIS is not? Does YawGNN outperform FLORIS in cases where both are usable for real-time control?

# Chapter 2

# Methods

## 2.1 Data

To train our GNN, which we call YawGNN, we use synthetic data generated by randomly sampling wind farm configurations (number of turbines, turbine positions, and turbine yaws) and wind conditions (speed and direction). The number of turbines, turbine positions, wind speed, and wind direction are all sampled from uniform distributions. Yaws are sampled from a normal distribution centered at 0 with a standard deviation of 20.

These parameters are then used to generate a wind farm graph (as pictured in Figure 1.4) using the graph data structure from the Deep Graph Library [9], which is an open source, framework agnostic library for building graph neural networks. Each node in the graph represents a turbine. Edges are directed, and determining whether or not two turbines share an edge for a given wind direction and yaw angle is done by computing if the downwind turbine lies in the "influential region" of the upwind turbine as defined by a threshold distance and threshold angle. We use the same thresholds as [31], which are conservative - a distance of 6300 meters and an angle of 90 degrees.

At each node in the graph, we store the wind speed, the wind direction, the x-y position of the turbine, the x-y position of the turbine rotated to align with the wind direction, the yaw angle of the turbine in radians, and the power of the turbine as computed using FLORIS and normalized by the power produced by a single-turbine

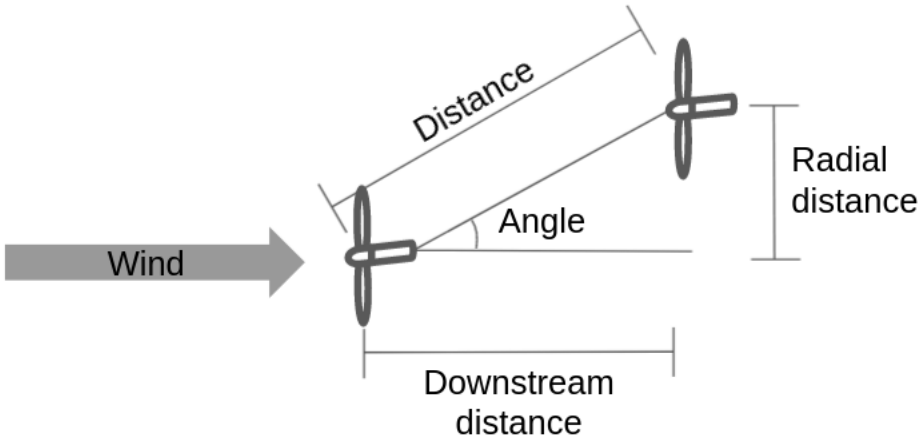



Figure 2.1: The downstream and radial distances between turbines are the key edge features in YawGNN. Also pictured are the distance and angle between turbines, which determine whether or not they share an edge.

wind farm at that wind speed. The maximum power produced by a turbine, then, is 1, and the maximum total power produced by a wind farm is equal to the number of turbines in the farm. We use the default settings for FLORIS for the NREL 5MW reference turbine [21].

At each edge in the graph, we store the x-y positions of the source and destination turbines and the x and y distances between the source and destination turbines, both in the world frame and in the wind direction-aligned frame. The x and y distances in the wind direction-aligned frame are referred to as the downstream and radial distances and are shown in Figure 2.1.

Finally, we also store some global information in the graph, namely the wind speed and wind direction.

## 2.2 Graph Neural Network Architecture

To investigate our research questions, we build off of the Physics-Induced Graph Neural Network (PGNN) defined in [31]. The original PGNN does not incorporate yaw, which is necessary for control. We can incorporate yaw in our model by including it as one of the node features. However, to properly include the physics-induced bias

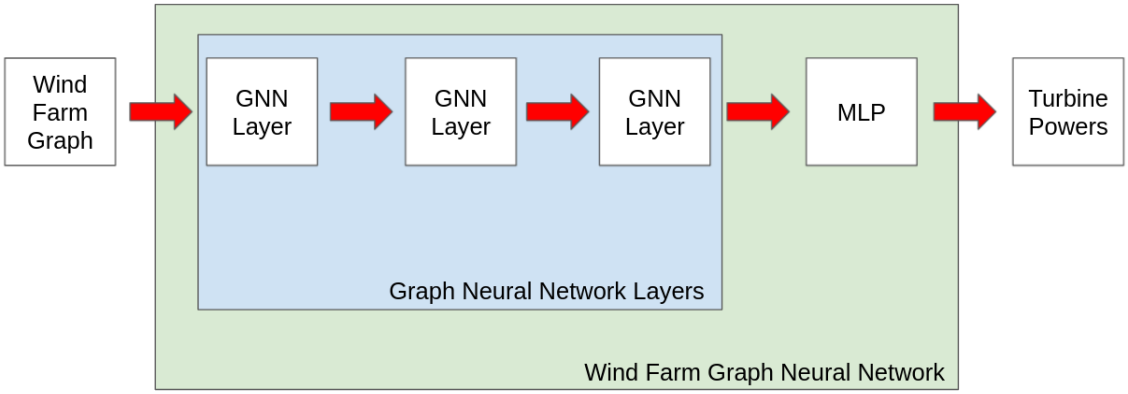



Figure 2.2: Graph neural network architecture

we would need to formulate a new weight function that computes the wake deficit factor as a function of downstream distance, radial distance, and yaw. As such, we omit the physics-induced bias and use a purely data-driven model.

Our model (outlined in Figure 2.2) is implemented using PyTorch, with the PyTorch Geometric MetaLayer class forming the basis of our graph layers. The input to the model is the aforementioned wind farm graph, specifically a subset of the node, edge, and global features stored in the data structure.

Referring to Figure 2.2, the input is first passed through the GNN portion of the model (three graph layers are pictured, but we tested with up to five). Each graph layer consists of an edge MLP, two node MLPs (one for aggregated incoming edge features and one for aggregated outgoing edge features), and a global MLP, each with a single hidden layer with 256 hidden units and Leaky ReLU activation. There are residual connections between all graph layers except for between the first and second layers. Compared to PGNN, our model is simpler as it has only a single hidden layer in its GNN MLPs rather than three.

Computation in each graph layer proceeds as in Algorithm 1. Note that the node update is implemented in PyTorch Geometric using a scatter mean. Computation in each graph layer is summarized as follows:

1. Pass edge features through edge MLP
2. Aggregate updated edge features at nodes

3. Pass node features and aggregated updated edge features through node MLPs
4. Aggregate updated node features globally
5. Pass global features and aggregated updated node features through global MLP
6. Return updated edge, node, and global features

These steps are analogous to computation in an analytical wake model. Updating the edge features is analogous to computing the wake deficit factor between each pair of interacting turbines. Updating the node features is analogous to computing the average wind speed at each turbine. Finally, updating the global feature is analogous to computing the steady state wind speed for the farm.

Again referring to Figure 2.2, the final block of our model is just another MLP. We use three hidden layers with 256, 256, and 128 hidden units and ReLU activation. The MLP is implemented such that the input to the first layer is of size equal to the number of node features (as opposed to the number of node features times the number of turbines). This is so that the model will be agnostic to the number of turbines in the wind farm and can be used for combinatorial generalization. Applying the final MLP weights on the wind farm graph is analogous to computing the power as a function of average wind speed in an analytical wake model. Finally, the output of the model is the estimated average, normalized power at each turbine.

For the node features, we use wind speed and yaw angle in radians. For the edge features, we use the x and y distances between the source and destination turbines and the downstream and radial distances between the source and destination turbines. The downstream and radial distances are key as they implicitly encode the wind direction for the farm. There is no need to explicitly encode the wind direction, because changing the wind direction can be thought of as equivalent to changing the layout of the wind farm in this representation. For the global feature, we use just the steady state wind speed. We divide the distance features by 1000 so that all features are of a similar magnitude.

## 2.3 Training and Evaluation

To train our GNN, we formulate our training objective as a supervised node regression problem. Our loss function is the sum of mean squared error (MSE) between the

**Algorithm 1** Graph Neural Network Layer Computation

**input** initial wind farm graph $\mathcal{G}$
set of edge features $\mathcal{E}$
set of node features $\mathcal{N}$
set of global features $g$
edge indices $\mathcal{U}$
node indices $\mathcal{I}$
incoming edge indices for node $i$ $\mathcal{R}_{i_{in}}$
outgoing edge indices for node $i$ $\mathcal{R}_{i_{out}}$
edge MLP $MLP_{\mathcal{E}}$
incoming edge node MLP $MLP_{\mathcal{N}_{in}}$
outgoing edge node MLP $MLP_{\mathcal{N}_{out}}$
global MLP $MLP_g$
edge aggregating function $f_e$
node aggregating function $f_n$
**output** updated wind farm graph $\mathcal{G}'$
**for** $(i, j)$ **in** $\mathcal{U}$ **do**
  $e'_{ij} \leftarrow MLP_{\mathcal{E}}(e_{ij}, n_i, n_j, g)$
**end**
**for** $i$ **in** $\mathcal{I}$ **do**
  $\mathcal{E}'_i \leftarrow e_{ji} \quad \forall j \in \mathcal{R}_{i_{in}}$
  $e'_i \leftarrow f_e(\mathcal{E}'_i)$
  $n'_i \leftarrow MLP_{\mathcal{N}_{in}}(n_i, e'_i)$
  $\mathcal{E}''_i \leftarrow e_{ij} \quad \forall j \in \mathcal{R}_{i_{out}}$
  $e''_i \leftarrow f_e(\mathcal{E}''_i)$
  $n''_i \leftarrow MLP_{\mathcal{N}_{out}}(n'_i, e''_i, g)$
**end**
$\mathcal{E}'' \leftarrow e''_{ij} \quad \forall (i, j) \in \mathcal{U}$
$\mathcal{N}'' \leftarrow n''_i \quad \forall i \in \mathcal{I}$
$n'' \leftarrow f_n(\mathcal{N}'')$
$g' \leftarrow MLP_g(n'', g)$
$\mathcal{G}' \leftarrow (\mathcal{N}'', \mathcal{E}'', g')$

predicted powers output by our model and the powers stored in the node data of our example graphs. To solve the optimization problem, we use AdamP [17], a variation of the popular Adam algorithm that prevents gradient step sizes from decaying prematurely. We choose an arbitrary batch size of 32 and generate 32 new example wind farm graphs for every gradient step. We are able to do this because randomly sampling wind farm layouts and estimating their powers using FLORIS is very inexpensive for small farms (less than one tenth of a second for a farm with 10 turbines).

We experimented with many different values for hyperparameters and settled on 5000 gradient steps, five GNN layers, an initial learning rate of 0.001, no L2 regularization, and a Cosine Annealing learning rate scheduler [23]. These models are all fairly lightweight - all were trained on a Thinkpad P50 laptop and it only took a few hours to complete 5000 gradient steps.

To evaluate our model during training, we generate validation and test datasets with 200 examples each prior to runtime and record the MSE losses for the power predictions from our model. We also evaluate performance qualitatively by visually comparing the estimated powers computed by our model to those computed by FLORIS for a given wind farm example over the full range of wind directions.

## 2.4 Power Estimation with Random Yaw Configurations

To answer our first research question, we begin by training our model on small numbers of turbines (first two, then five) where the farm layout is the same in every example. Once we are able to accurately estimate powers for fixed farm layouts and random wind speeds, wind directions, and turbine yaws, we move to the case where the farm layout differs in every example. Then, we compare our model's performance on power estimation to FLORIS in both accuracy and runtime. Finally, in order to evaluate the effect of modeling varying yaw on the accuracy of the model for power estimation, we compare our model's power estimation accuracy to that of a model trained only on examples where all yaws were 0 degrees relative to the wind direction.

**Algorithm 2** YawGNN Yaw Optimization via Gradient Descent

**input** initial turbine yaw $\psi_i \quad \forall i \in \mathcal{I}$
initial wind farm graph $\mathcal{G}$
node indices $\mathcal{I}$
node features $\mathcal{N}$
number of updates $T$
step size $\eta$
**output** updated turbine yaw $\psi_i \quad \forall i \in \mathcal{I}$
**for** $t \leftarrow 1$ **to** $T$ **do**
  $\mathcal{P} \leftarrow YawGNN(\mathcal{G})$
  $\frac{\delta \mathcal{P}}{\delta \psi} = autograd(\mathcal{P}, \mathcal{N})_\psi$
  **for** $i$ **in** $\mathcal{I}$ **do**
    $\psi_i \leftarrow \psi_i + \eta \frac{\delta \mathcal{P}}{\delta \psi_i}$
  **end**
  $\mathcal{G} \leftarrow$ Assign the updated yaws $\psi_i \quad \forall i \in \mathcal{I}$
**end**

## 2.5 Yaw Optimization

To answer our second research question, we take a similar approach - first testing how well models trained on only a single wind farm layout are able to compute optimal yaw configurations, then expanding to optimize yaw configurations for models trained on random wind farm layouts. Our yaw optimization algorithm, shown in Algorithm 2, performs 250 steps of gradient descent using PyTorch's built-in automatic differentiation feature to compute the gradient of the network output (power) with respect to the yaw features. Four different initial conditions are tested - all zero yaws (the greedy solution), the optimal yaws computed using FLORIS, random yaws sampled from a normal distribution centered at zero, and yaws $\pm 20$ degrees away from zero. We evaluate our solutions against FLORIS's built-in yaw optimization tool, which utilizes Sequential Least Squares Quadratic Programming (SLSQP) with an iteration limit of 50.

## 2.6 Combinatorial Generalization

To answer our third research question, we evaluate how well models trained on examples with between 2 and 10 turbines are able to predict the power output and compute optimal yaw configurations for wind farms with up to 100 turbines. An important variable is the farm length and width, or the grid size. If the grid size is too small, then more wind farms will be crammed into the same area and the overall power output will drop. There are two approaches to making sure the trained model is able to generalize to these cases. First, we can make the assumption that the density of turbines in a real farm will be roughly constant regardless of the number of turbines and evaluate the combinatorial generalization performance of our model only on cases that are physically realistic. Alternatively, we can include the grid size (or some other feature describing the grid size, such as the aforementioned farm area and turbine density features) as a feature in the model and vary the grid size in the training data. We take the former approach and leave the latter to future work.

## 2.7 Real-Time Control

Finally, to answer our fourth research question, we evaluate how well our model performs compared to FLORIS when each is used as the model in a model predictive controller for real-time, wake steering control of wind farms. We also compare the two model predictive controllers to the greedy control method. We simulate real-time control in two parts for a wind farm where the wind direction is changing 60 degrees at a given rate.

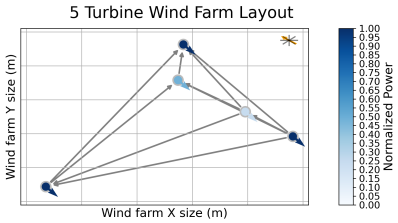
In the first part, we compute the target yaws. We compute the optimal yaw configuration for the current wind direction using the given model (YawGNN or FLORIS) and record the resulting yaws and the runtime of the optimization. Then, we advance the wind direction by the amount it would have changed during the runtime of the optimization and compute the optimal yaw configuration for the new wind direction, repeating this process until the wind direction has finished its 60 degrees of change. Target yaws for the greedy controller are simply 0 degrees (relative to the wind direction).

In the second part, we simulate the control of the wind farm using the previously

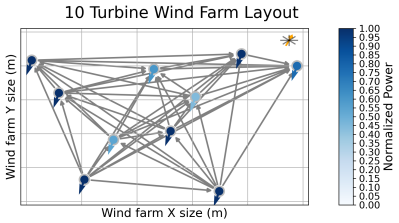
computed target yaws as the wind direction changes 60 degrees at the given rate with a step size of 1 second. The wind farm is given the next target yaw at the time step corresponding to when it was finished being computed during the first part. Control commands are computed as the difference between the target yaw configuration and the current yaw configuration. Each wind turbine is able to yaw a maximum of 0.3 degrees per second [21]. At the end of the simulation, we compute the power total power produced by each control method and the change in power relative to greedy control for YawGNN and FLORIS.

We experiment with four different farm sizes (5, 10, 15, and 20 turbines) and five different rates of change for the wind direction (1, 2, 3, 4, and 5 degrees per minute). The rates of change were chosen based on operational wind farm data [1] with datasets from multiple different days. We computed the average 10 minute change in wind direction for each dataset, and then computed the minimum, average, and maximum *average 10 minute change in wind direction* as 0.5, 7, and 14 degrees per minute, respectively. However, our initial experiments showed that neither YawGNN nor FLORIS were able to improve power for rates of change above 5 degrees per minute for any of the chosen farm sizes, so we limited our final experiments to the aforementioned subset.

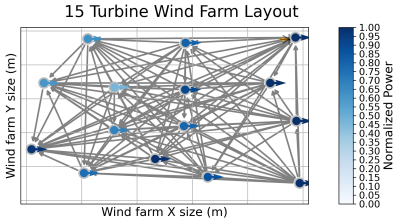
Figure 2.3 shows the wind farm graphs of the farms used in the experiments, and Figure 2.4 shows the power curves for the 60 degree wind direction ranges used in the experiments. We chose these ranges due to their high variability and local minima, which would both pose a difficult control problem and increase the chance of wake steering control yielding improvement over the greedy solution.



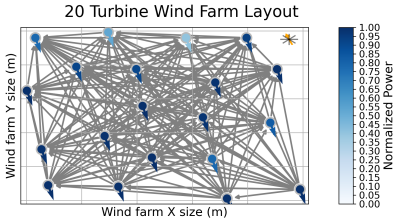

(a) Layout of 5 turbine farm with turbines yawed to align with wind from 130 degrees




(b) Layout of 10 turbine farm with turbines yawed to align with wind from 200 degrees

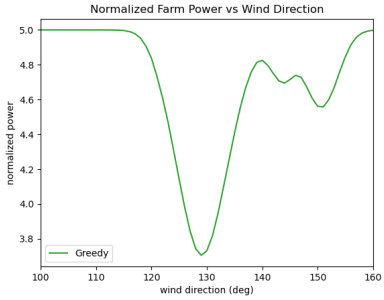



(c) Layout of 15 turbine farm with turbines yawed to align with wind from 90 degrees

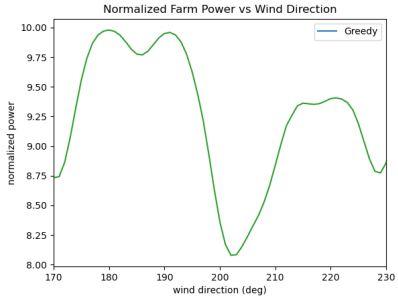



(d) Layout of 20 turbine farm with turbines yawed to align with wind from 160 degrees

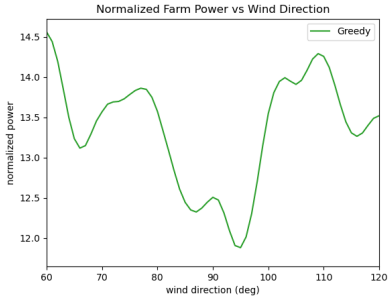
Figure 2.3: Wind farm graphs with normalized powers for farms used in real-time control experiments yawed to the middle of their respective wind direction ranges as used in the experiments. Circles are turbines with arrows pointing in the direction they're facing, edges are between turbines that within a threshold angle and distance defining wake interaction, and the colors of the circles range from white for zero power to dark blue for full power.



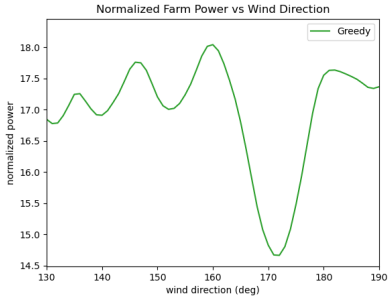

(a) Normalized power for 5 turbine farm as wind direction varies between 100 and 160 degrees




(b) Normalized power for 10 turbine farm as wind direction varies between 170 and 230 degrees




(c) Normalized power for 15 turbine farm as wind direction varies between 60 and 120 degrees




(d) Normalized power for 20 turbine farm as wind direction varies between 130 and 190 degrees

Figure 2.4: Normalized power versus wind direction for greedy control for the 60 degree ranges used in the real-time control experiments

# Chapter 3

# Results

## 3.1 Power Estimation

### 3.1.1 Power Estimation with Varying Yaw

First, we demonstrate power estimation for the simplest case - two turbines in series with constant wind conditions where only the yaw of the upwind turbine varies as shown in Figure 3.1. Figure 3.2 shows that our model is able to match FLORIS nearly exactly and, more importantly for our purposes, that the yaw that maximizes the farm's power output is not zero, but rather a yaw in which the upwind turbine deflects its wake away from the downwind turbine.

Next, we test our model's ability to estimate power output when trained on a

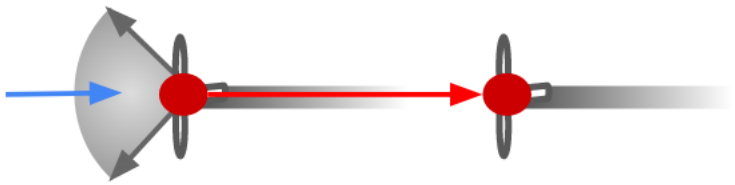


Figure 3.1: Toy example for power estimation and yaw optimization where the incident wind direction is fixed and only the yaw of the upwind turbine varies

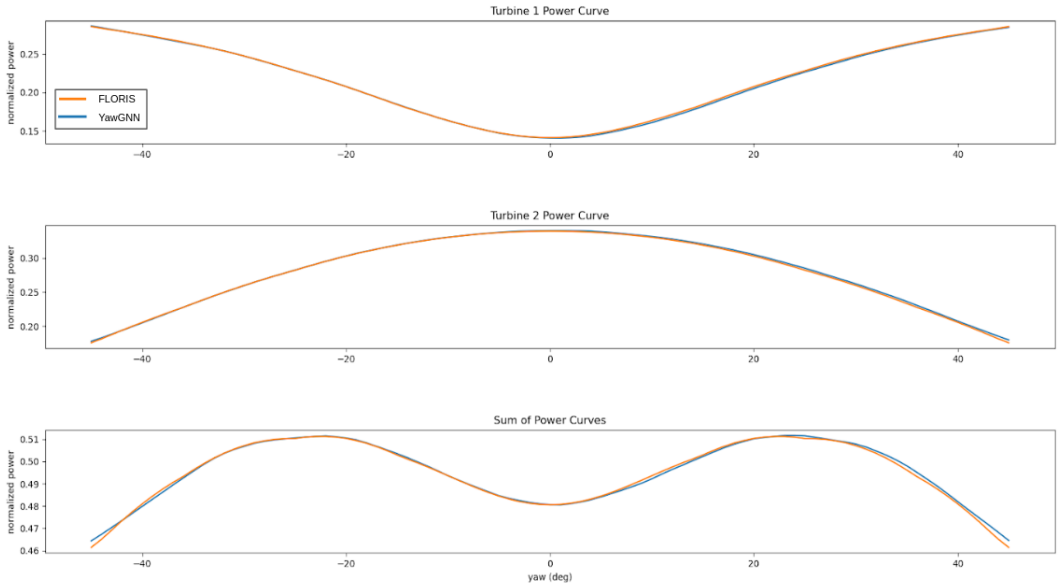



Figure 3.2: FLORIS and YawGNN power curves for a 2 turbine wind farm in 0°, 8 m/s wind as upwind yaw varies between ±45°

slightly more complicated case, shown in Figure 3.3 - five wind turbines with constant layout and varying wind direction and speed. As shown in Figure 3.4, the GNN is able to accurately estimate the power output of the farm across the full range of wind directions.

Finally, we use our model to estimate power output in the general case - five wind turbines with random layout, yaws, wind direction, and wind speed. Figure 3.5 demonstrates that our model is able to approximate FLORIS nearly as well as in the toy examples, with some slight disagreement in high frequency regions. Importantly, the trends remain the same between the two models even where the power predictions disagree.

To quantitatively evaluate the impact of varying turbine yaws on our model's power estimation performance, we estimate the power for wind farms with between 5 and 100 turbines (100 randomly sampled examples for each increment of 5 turbines) using a GNN trained on examples with between 2 and 10 turbines. Then, we compute the percent error of our model's prediction relative to the FLORIS prediction. We perform this process for a model trained on examples where the yaw was random and for a model trained on examples where the yaw was fixed. For each model, we

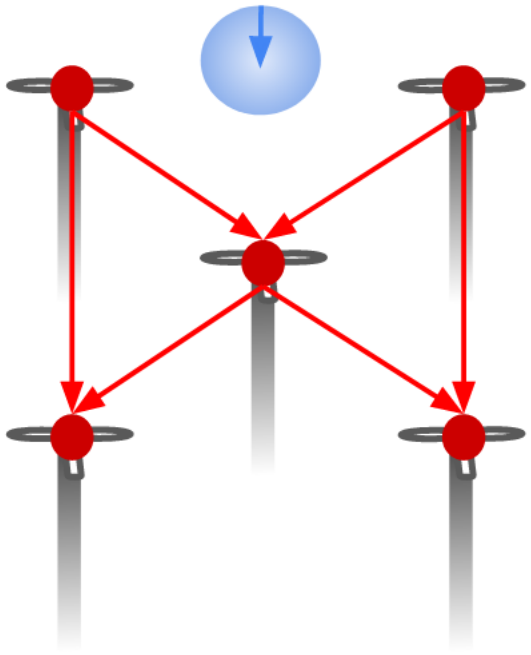


Figure 3.3: Toy example for power estimation where the yaw is fixed and the incident wind direction varies between 0 and 360 degrees

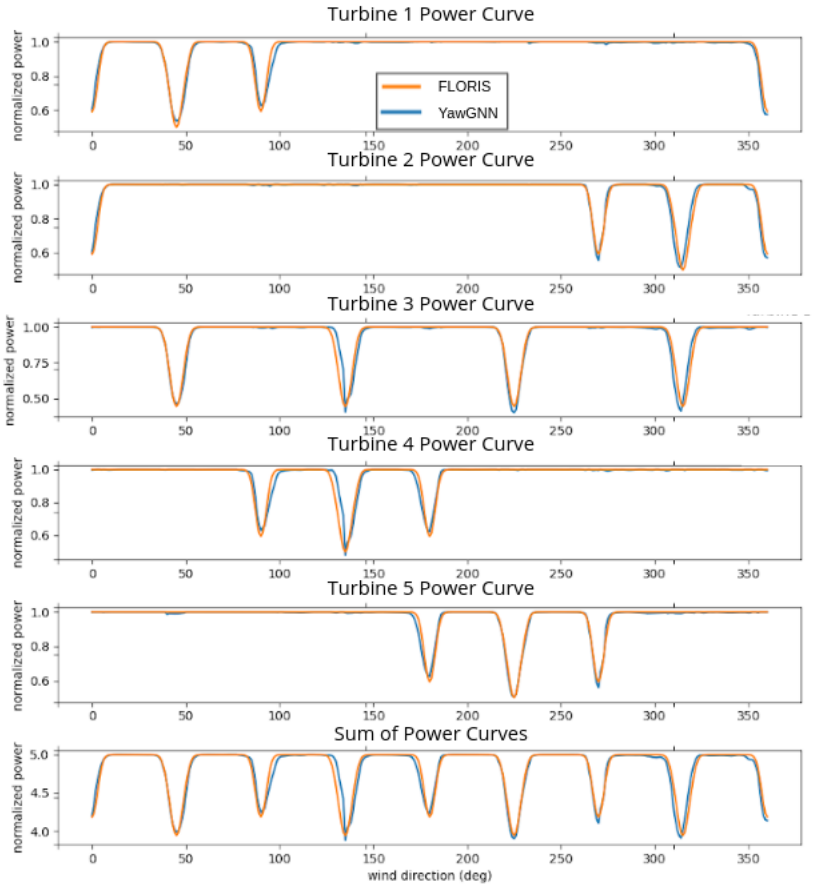



Figure 3.4: FLORIS and YawGNN power curves for a 5 turbine wind farm with varying wind direction

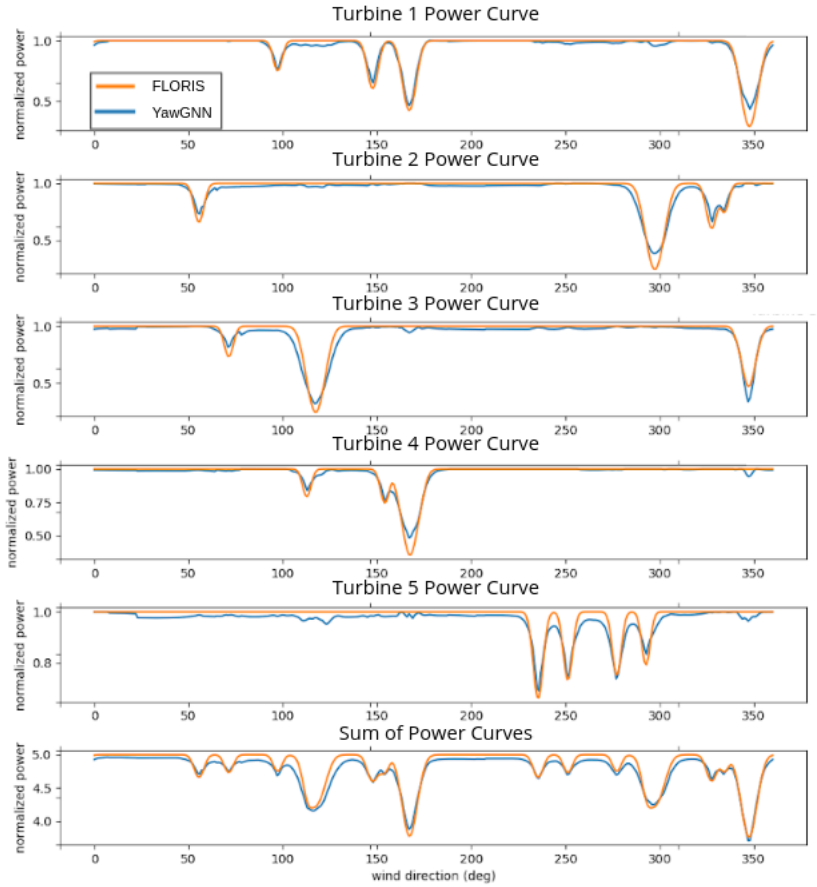



Figure 3.5: FLORIS and YawGNN power curves for a randomly sampled 5 turbine wind farm with varying wind direction

| Average Power Estimation Percent Error | | | | |
|---|---|---|---|---|
| **Train on Yaw?** | **Test on Yaw?** | **Average** | **In Dist.** | **Out of Dist.** |
| Yes | Yes | 1.35 | 1.44 | 1.34 |
| Yes | No | 1.23 | 0.61 | 1.30 |
| No | Yes | 11.71 | 10.96 | 11.79 |
| No | No | 2.19 | 0.24 | 2.40 |

Table 3.1: Average YawGNN power estimation percent error relative to FLORIS;
"Average" column gives the average percent error over all farm sizes;
"In Dist." column gives the average percent error for farms of 10 turbines or less;
"Out of Dist." column gives the average percent error for farms with greater than 10 turbines

evaluate its performance on examples where the yaw is random and on examples where the yaw is fixed. Results are shown in Table 3.1.

As expected, the model not trained on examples with misaligned yaws performs terribly when it is shown examples with misaligned yaws, having an overall average percent error of 11.71%. Interestingly, the model trained on random yaws and evaluated on random yaws has roughly the same performance whether examples are in distribution (10 turbines or fewer) or out of distribution (greater than 10 turbines) with percent errors relative to FLORIS of 1.44% and 1.34%, respectively. The model trained on fixed yaws and evaluated on fixed yaws has significantly better in distribution performance and significantly worse out of distribution performance relative to the model trained on random yaws and evaluated on random yaws, with percent errors relative to FLORIS of 0.24% and 2.40%, respectively. The model trained on random yaws and evaluated on fixed yaws has better in distribution performance, 0.61%, and roughly the same out of distribution performance, 1.30%, as the same model evaluated on random yaws.

### 3.1.2 Comparison to FLORIS

To more directly compare the power estimation performance of our model to FLORIS, we train a GNN on wind farms with between 2 and 10 turbines with random turbine layouts, turbine yaws, and wind directions. Then, we randomly sample wind farms with between 5 and 100 turbines (100 examples for each increment of 5 turbines), use FLORIS and YawGNN to estimate the power of each wind farm, and record the

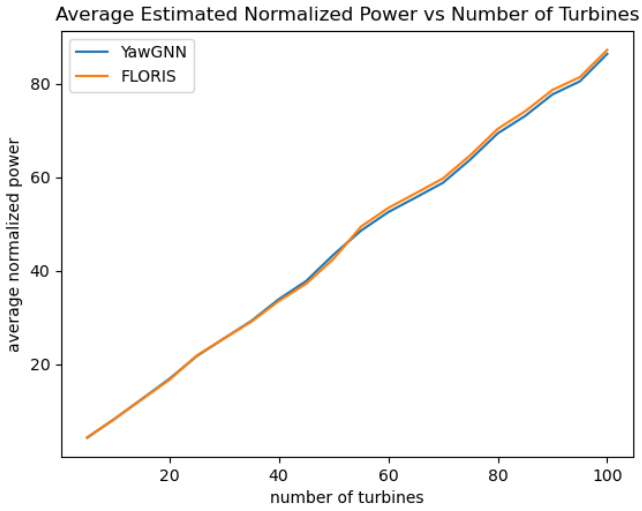



Figure 3.6: FLORIS and GNN power estimates for randomly sampled wind farms as the number of turbines increases

respective power estimates and runtimes. Figure 3.6 shows that our GNN power estimates track well with the FLORIS estimate all the way up to 100 turbines even though the model had never seen a wind farm with more than 10 turbines. Looking more closely, Figure 3.7 shows that the average error in normalized power between the YawGNN prediction and the FLORIS prediction stabilizes around 1 turbine equivalent. Figure 3.8 shows the average percent error between the YawGNN prediction and the FLORIS prediction. The average percent error in distribution is 1.41% and the average percent error out of distribution is 1.34%. The variability in Figure 3.8 is explained by discrete increases in the physical area of the farm at 10, 20, and 50 turbines in order to accommodate the increasing number of turbines. Figure 3.9 shows that while both models' power estimation scale linearly with the number of turbines, the FLORIS power estimation takes around two orders of magnitude longer to compute.

Next, we show more qualitative power estimation results for combinatorial generalization of a GNN trained on 2 to 10 turbines with random layouts, yaws, and wind directions. Figure 3.10 shows the FLORIS and GNN power curves for a 20 turbine farm and Figure 3.11 shows the FLORIS and GNN power curves for a 100 turbine farm. In both cases, the trends match nearly exactly, although there is some

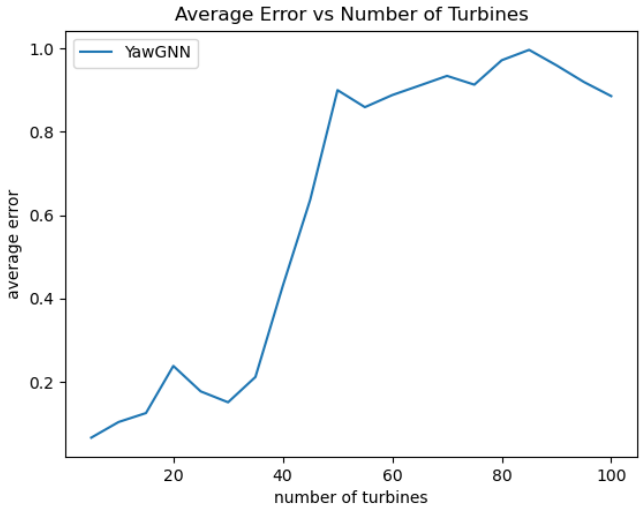



Figure 3.7: Average error (in normalized power) between the YawGNN and FLORIS power predictions for randomly sampled wind farms as the number of turbines increases

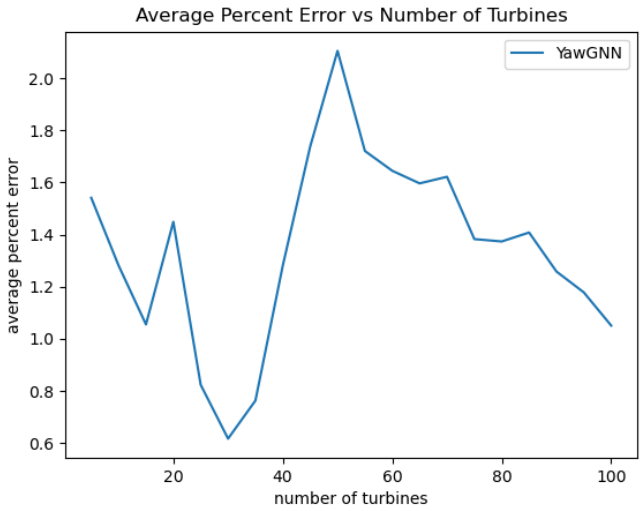



Figure 3.8: Average percent error between the YawGNN and FLORIS power predictions for randomly sampled wind farms as the number of turbines increases

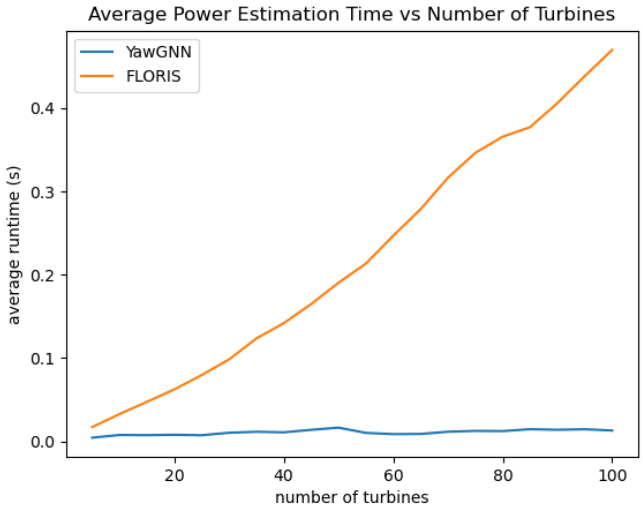



Figure 3.9: Runtimes for FLORIS and GNN power estimations for randomly sampled wind farms as the number of turbines increases

disagreement between the models about the extent of the peaks and troughs.

However, in order to achieve this level of agreement, the grid size was increased. The GNN was trained on examples with a grid size of 3,000 meters, while the 20 turbine example has a grid size of 10,000 meters and the 100 turbine example has a grid size of 50,000 meters. When the model attempts to predict the power for a 20 turbine example with a grid size of 3,000 meters, it correctly predicts the shape of the power curve but overestimates the power produced. It seems that its features and experience are insufficient to inform it that placing more turbines in the physical same area reduces the power due to increased wake interactions. As the grid size is gradually increased to 10,000 meters, the GNN power prediction comes more and more into alignment with the FLORIS power curve. This is a limitation of the current features and training data.

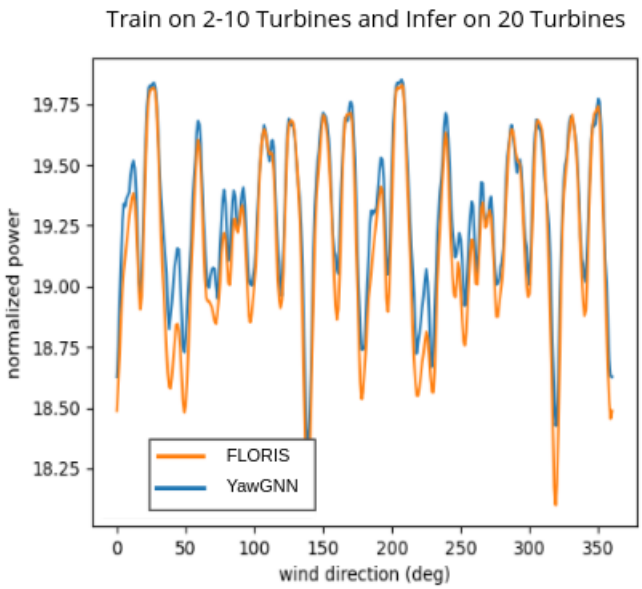



Figure 3.10: FLORIS and GNN power curves for a wind farm with 20 turbines where the GNN was only trained on wind farms with 2 to 10 turbines

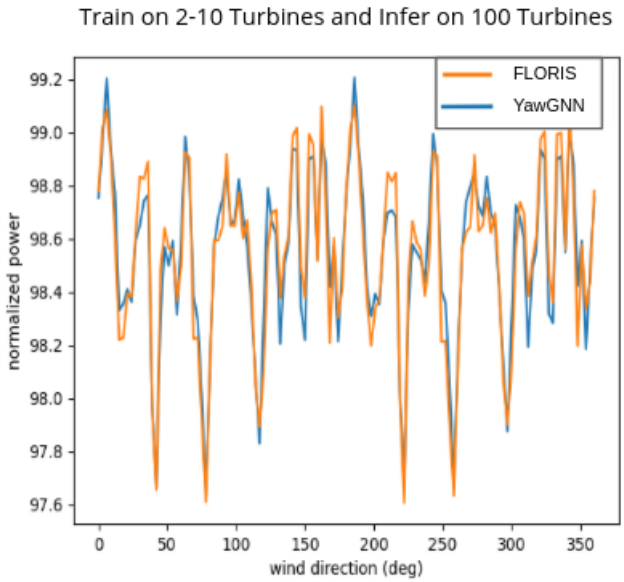



Figure 3.11: FLORIS and GNN power curves for a wind farm with 100 turbines where the GNN was only trained on wind farms with 2 to 10 turbines

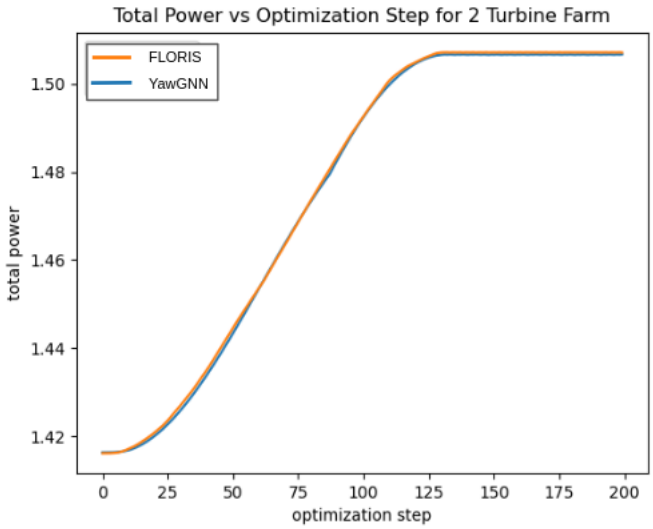



Figure 3.12: Wind farm power output computed using FLORIS and our GNN as optimization progresses for the 2 turbine case

## 3.2 Yaw Optimization

### 3.2.1 Fixed Layout and Wind Conditions

For yaw optimization, we again started with the simplest case - two turbines in series with constant wind conditions where only the yaw of the upwind turbine varies. Figures 3.12 and 3.13 show the progress of our auto differentiation gradient descent algorithm on this example. Throughout the optimization, the power predicted by our model nearly exactly matches the power computed by FLORIS, which indicates that the local optima discovered by our model should coincide with a local optima according to FLORIS as well. The algorithm also converges smoothly to an optima. Referring back to Figure 3.2, we see that the power is indeed greatest at the yaw angle of around $-22°$ computed by our algorithm.

### 3.2.2 Random Layout and Wind Conditions

When the model is trained on random layouts, and wind directions, we are able to optimize much more interesting scenarios. Figure 3.14 shows the power curves

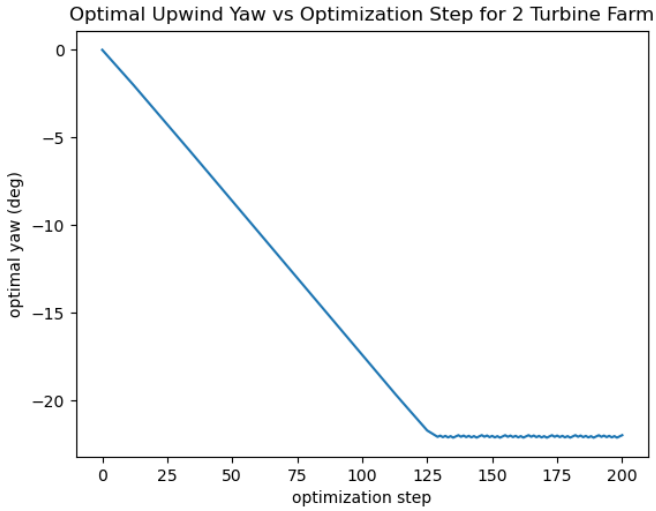



Figure 3.13: Upwind yaw angle as optimization progresses for the 2 turbine case

associated with the optimal yaw configurations as computed by the greedy method (just point all turbines directly into the wind), FLORIS, and our auto differentiation gradient descent method. Our algorithm is able to consistently find solutions of comparable quality to the FLORIS optimal solution, but only when the initial conditions are yaws far from zero (we use both $\pm 20°$ and then take the solution that gives the higher power as computed using FLORIS). Initializing from zero yaw or random yaws normally distributed about zero often results in suboptimal yaw configurations due to poor local minima because zero is the greedy solution and comprises a local minima in the objective function.

We can clearly see that for most wind directions, the greedy solution is in fact optimal, in which case FLORIS and our algorithms simply converge to the greedy solution. However, in certain wind directions (namely, 0°, 126°, 180°, and 306°), the greedy solution is highly suboptimal because there are a lot of wake losses resulting from the geometry at that wind direction. In these cases, FLORIS and our algorithms find much higher quality solutions.

Figure 3.15 shows the layout of the same farm from Figure 3.14 with wind coming from 126°, one of the wind directions where the greedy solution is highly suboptimal. Even though the difference between the greedy yaws and optimal yaws appears slight, the yaws computed using YawGNN result in an 8% increase in power over the greedy

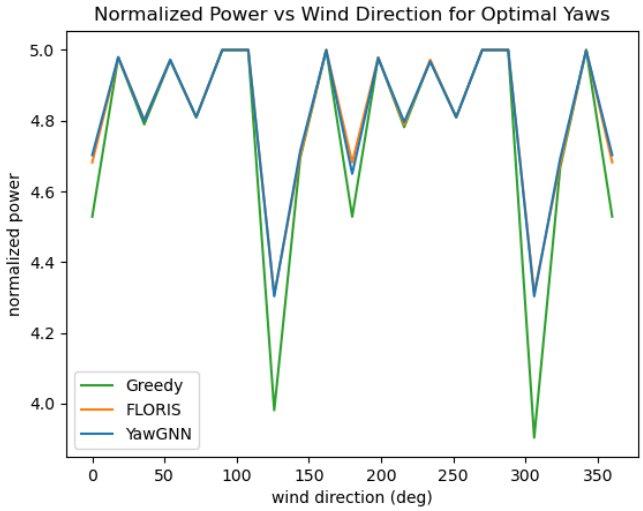


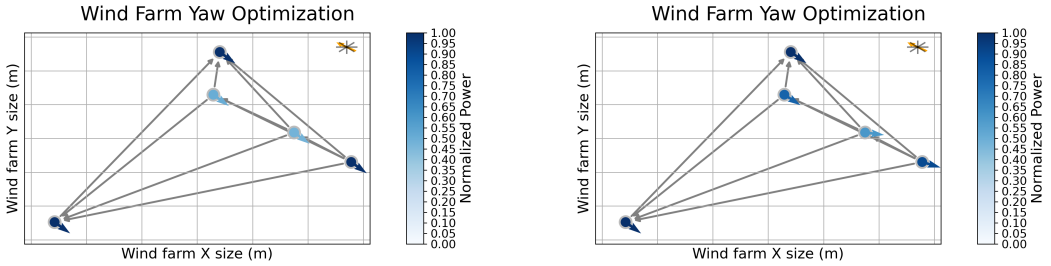

Figure 3.14: Power curves for optimal yaw configurations as computed by the greedy method, YawGNN, and FLORIS

yaws. If we assume that these wind conditions are present for 1% of the year, then using the YawGNN yaws would result in a yearly increase in power of 47.87 MWh, which is equivalent to powering an additional 4.39 houses per year.

We evaluate the performance of our yaw optimization algorithm more generally using 5 farms for each increment of 5 between 5 and 100 turbines. We train a model only on wind farms with between 2 and 10 turbines, then compute the optimal yaws for 20 evenly spaced wind directions between 0 and 360 degrees. For in distribution examples, the computed yaws increase power relative to the greedy yaws by an average of 1.03%. For out of distribution examples, the computed yaws increase power relative to the greedy yaws by an average of 0.21%.

### 3.2.3 Comparison to FLORIS

To evaluate the combinatorial generalization abilities of our yaw optimization algorithm and compare its performance and runtime to FLORIS, we repeat the experiments that we ran for power estimation for yaw optimization. We use the same GNN trained on wind farms with between 2 and 10 turbines with random turbine layouts, turbine yaws, and wind directions and generate two datasets. The first dataset consists of




(a) Greedy yaws

(b) YawGNN optimal yaws

Figure 3.15: Wind farm graphs with turbines yawed to the greedy yaws and YawGNN optimal yaws for the same 5 turbine example from Figure 3.14 with wind coming from 126°. In (b), note the decrease in power (lighter color) relative to (a) for the upwind turbine and increase in power (darker color) relative to (a) for the downwind turbines.

5 farms each for 2, 4, 6, 8, and 10 turbines. We run the FLORIS and YawGNN yaw optimization algorithms on them for 20 different wind directions evenly spaced between 0 and 360 degrees and record the runtime and the percent increase in power relative to greedy. Figure 3.16 shows the average percent increases in power relative to greedy for the FLORIS and YawGNN yaws on this dataset. Figure 3.17 shows the average runtimes for the FLORIS and YawGNN yaw optimization algorithms. We see that there is a tradeoff between runtime and solution quality even for these in distribution examples. The average percent increase in power for FLORIS is 0.89%, while YawGNN is significantly faster but only increases power by 0.65% on average.

The second dataset consists of 5 farms each for farm sizes of between 5 and 100 turbines in increments of 5. We run the YawGNN yaw optimization algorithm on all of them for 20 different wind directions evenly spaced between 0 and 360 degrees and record the runtime and percent increase in power relative to greedy. We also run the FLORIS yaw optimization algorithm on only one farm and one wind direction per number of turbines and record the runtime. Figure 3.18 shows the average percent increase in power for yaws computed using YawGNN on this dataset. Figure 3.19 shows the YawGNN and FLORIS runtimes on this dataset, while Figure 3.20 shows only the YawGNN runtimes because the difference in runtime between FLORIS and YawGNN becomes so great as the number of turbines increases that the runtime for YawGNN appears in Figure 3.19. While the FLORIS SLSQP optimization method

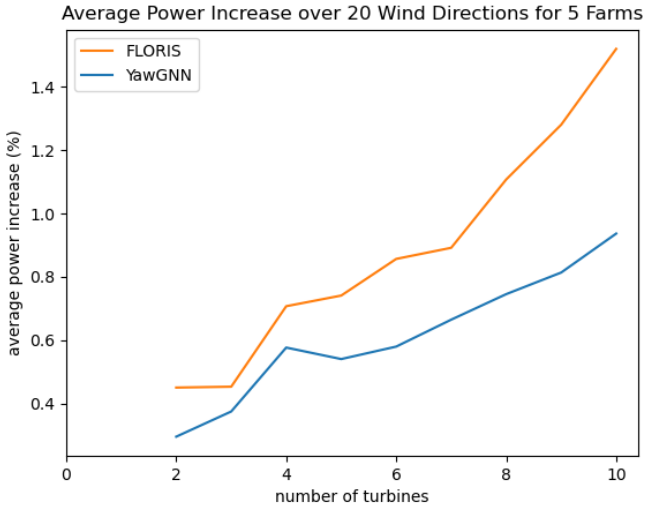



Figure 3.16: Average percent increase in power relative to greedy for optimal yaws computed using FLORIS and YawGNN as number of turbines increase

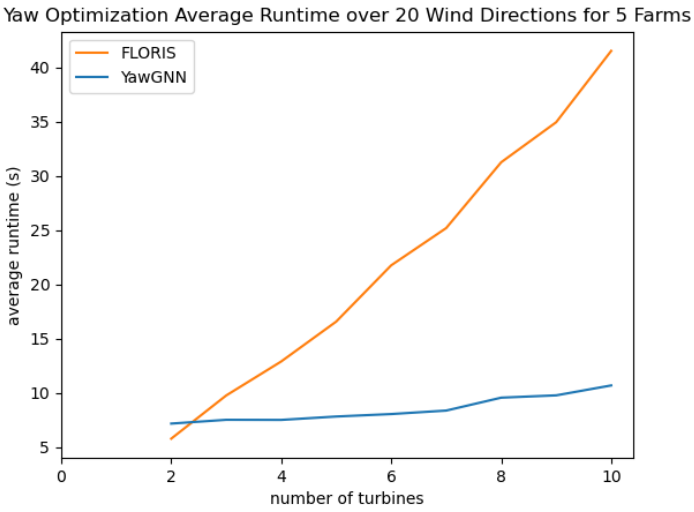



Figure 3.17: Average runtime for FLORIS and YawGNN yaw optimization as number of turbines increase

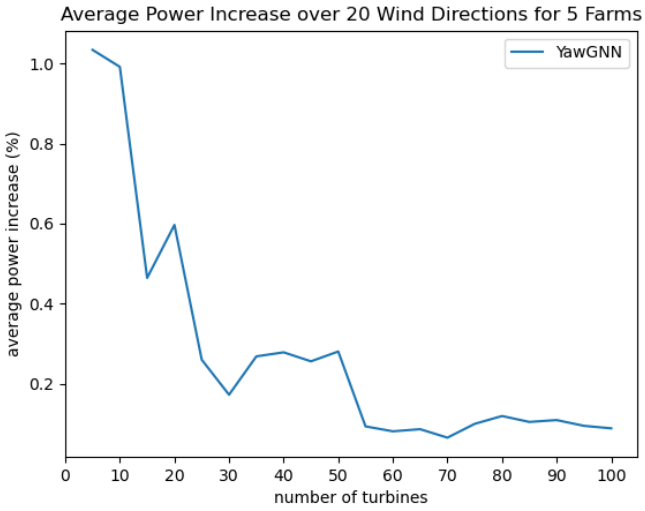



Figure 3.18: Average percent increase in power relative to greedy for optimal yaws computed using YawGNN for farms with up to 100 turbines

scales quadratically in the number of turbines, our GNN auto differentiation gradient descent method is functionally linear in time (in theory, the number of edges in the wind farm graph scales quadratically with the number of turbines, but we never see any examples with enough edges for this to make an impact). It takes FLORIS 40 minutes to compute a solution for 100 turbines, while taking our GNN only 25 seconds - a difference of two orders of magnitude. We will see that this difference in runtime enables real-time control of wind turbines for farm sizes and wind direction rates of change that were not possible using FLORIS, even for farm sizes where the difference in runtime is not so stark.

## 3.3 Real-Time Control

Finally, we demonstrate real-time wake steering control of wind farms using the greedy, FLORIS, and YawGNN methods. Figures 3.21 and 3.22 show the percent increases in power relative to greedy and the raw increases in power in kilowatts, respectively, for our experiments. We see that FLORIS's applicability to real-time control degrades quickly with increasing numbers of turbines and increasing wind direction frequency. It is only able to improve the power output of the farm for wind

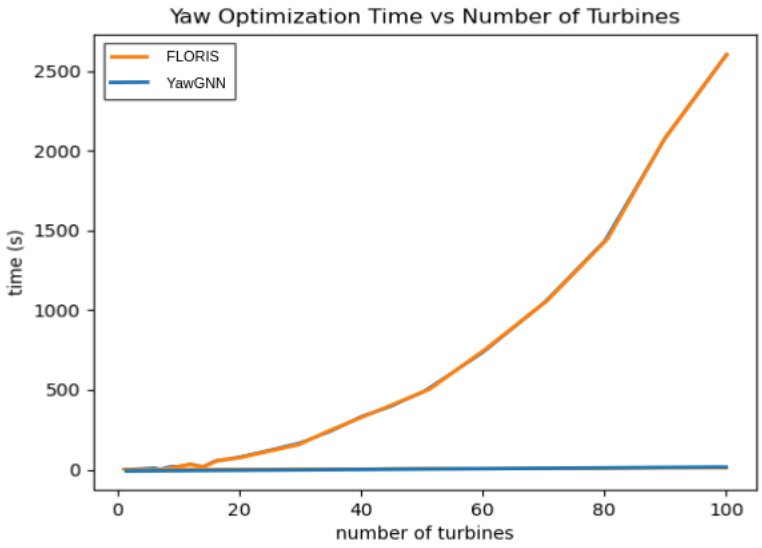



Figure 3.19: Yaw optimization runtimes for FLORIS and YawGNN for farms with up to 100 turbines

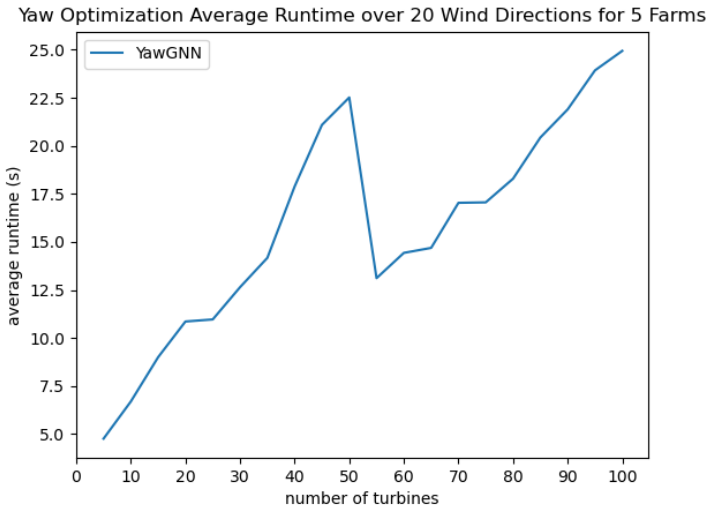



Figure 3.20: Yaw optimization runtimes for only YawGNN for farms with up to 100 turbines; the sharp drop in runtime at 50 turbines is due to an increase in the farm area to accommodate more turbines, which reduces the number of edges in subsequent examples

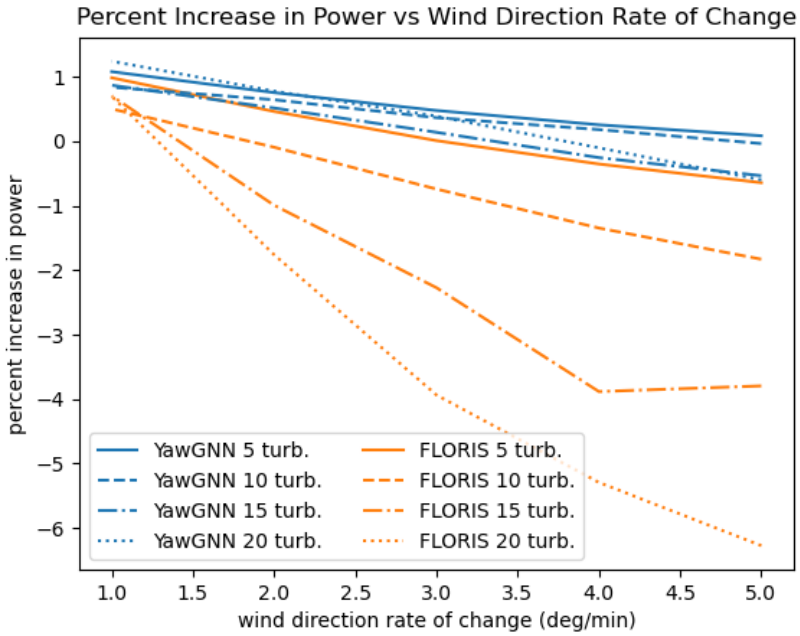



Figure 3.21: Percent increase in power relative to greedy for YawGNN and FLORIS real-time control algorithms for different wind direction frequencies and farm sizes

conditions changing at a rate greater than 1 degree per minute for the example with 5 turbines. Due to its slower runtime, the increase in power that FLORIS provides is lower than YawGNN for every example. In contrast, YawGNN is able to improve the power output for every farm size tested with the wind direction changing up to 3 degrees per minute, and is able to improve power for faster wind direction frequencies for farms with 5 and 10 turbines.

Figures 3.23, 3.24, 3.25, and 3.26 show the power curves for the four example wind farms being controlled by the greedy, FLORIS, and YawGNN methods at wind direction rates of change of 1, 3, and 5 degrees per minute. FLORIS's slower runtime is apparent in these figures as it reacts slower and slower to changes in the wind direction as the number of turbines increases. The response time of the YawGNN controller is much more consistent in comparison. While the FLORIS controller is able to reach yaws that generate more power than the YawGNN for some wind directions for some examples, the YawGNN controller always generates more total power in the experiment.

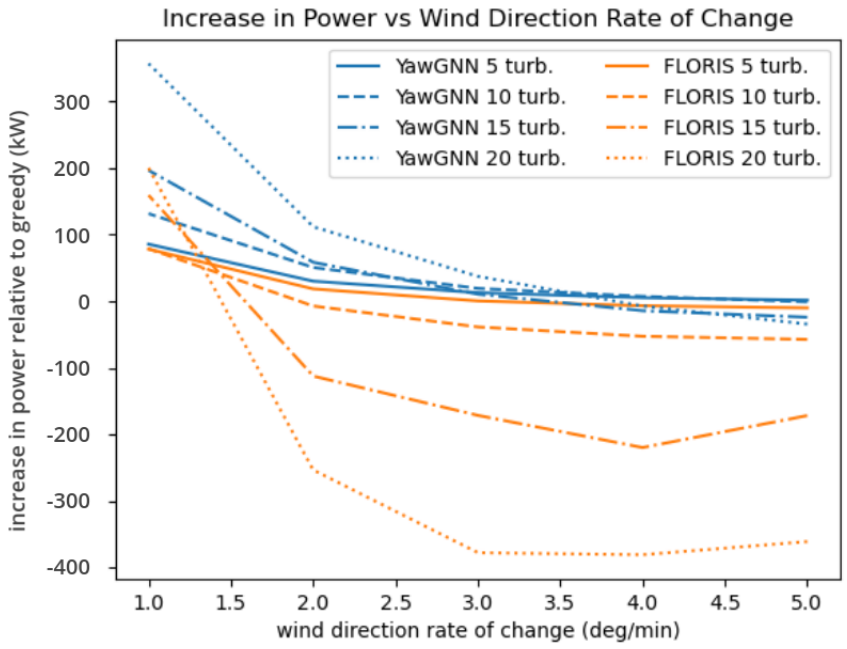
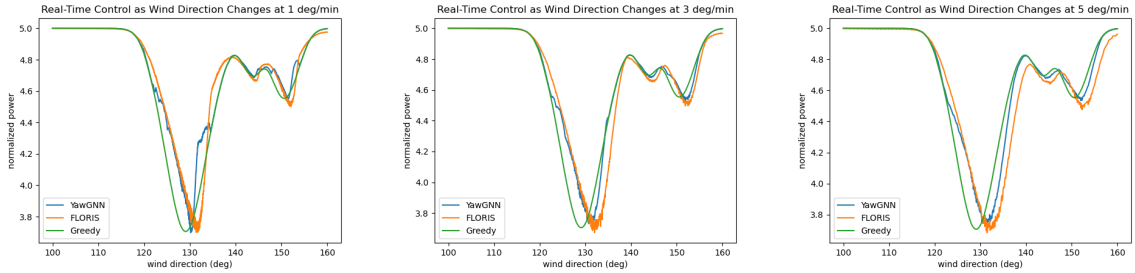
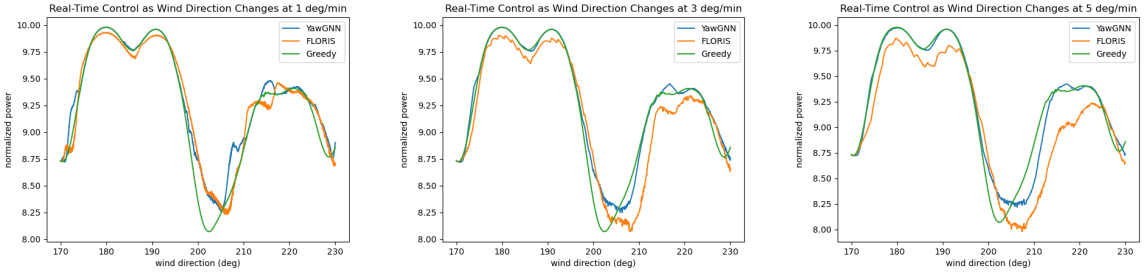



Figure 3.22: Increase in power in kilowatts relative to greedy for YawGNN and FLORIS real-time control algorithms for different wind direction frequencies and farm sizes



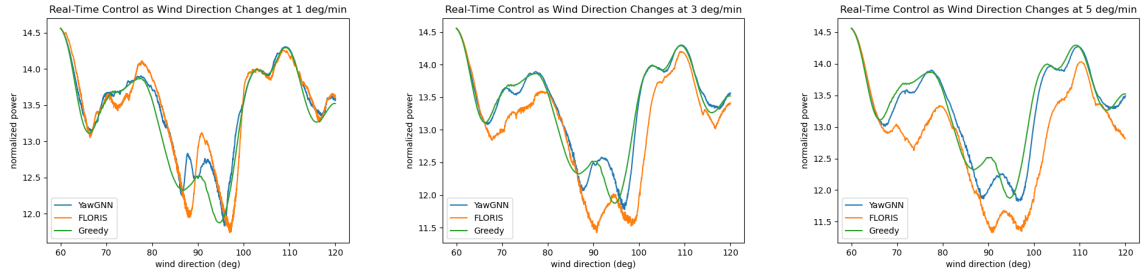

(a) Control for 1°/min (b) Control for 3°/min (c) Control for 5°/min

Figure 3.23: Power curves for 5 turbine farm under greedy, FLORIS, and YawGNN real-time control for wind directions from 100° to 160° at three different rates of change



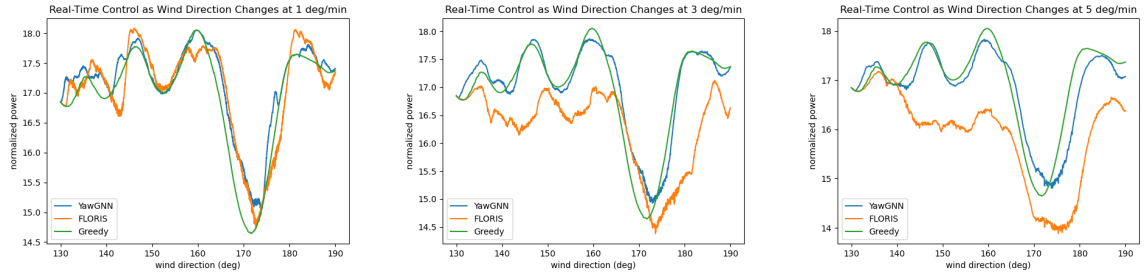

(a) Control for 1°/min (b) Control for 3°/min (c) Control for 5°/min

Figure 3.24: Power curves for 10 turbine farm under greedy, FLORIS, and YawGNN real-time control for wind directions from 170° to 230° at three different rates of change




(a) Control for 1°/min (b) SControl for 3°/min (c) Control for 5°/min

Figure 3.25: Power curves for 15 turbine farm under greedy, FLORIS, and YawGNN real-time control for wind directions from 60° to 120° at three different rates of change




(a) Control for 1°/min (b) Control for 3°/min (c) Control for 5°/min

Figure 3.26: Power curves for 20 turbine farm under greedy, FLORIS, and YawGNN real-time control for wind directions from 130° to 190° at three different rates of change

| Additional Houses Powered per Year by YawGNN | | | | |
|---|---|---|---|---|
| | **5 turbines** | **10 turbines** | **15 turbines** | **20 turbines** |
| **1 deg/min** | 6.85 | 10.52 | 15.73 | 28.58 |
| **2 deg/min** | 4.79 | 8.08 | 9.33 | 17.88 |
| **3 deg/min** | 3.02 | 4.57 | 2.48 | 8.92 |
| **4 deg/min** | 1.61 | 2.25 | -4.64 | -2.40 |
| **5 deg/min** | 0.53 | -0.48 | -9.74 | -13.86 |

Table 3.2: Number of additional houses powered per year by YawGNN control under assumption that selected 60 degree ranges are representative of 10% of yearly wind conditions and remaining 90% are greedy-optimal

| Additional Houses Powered per Year by FLORIS | | | | |
|---|---|---|---|---|
| | **5 turbines** | **10 turbines** | **15 turbines** | **20 turbines** |
| **1 deg/min** | 6.25 | 6.29 | 12.74 | 16.04 |
| **2 deg/min** | 2.93 | -1.19 | -18.06 | -40.66 |
| **3 deg/min** | 0.05 | -9.36 | -41.28 | -90.86 |
| **4 deg/min** | -2.26 | -16.97 | -70.53 | -122.12 |
| **5 deg/min** | -4.10 | -23.06 | -68.93 | -144.75 |

Table 3.3: Number of additional houses powered per year by FLORIS control under assumption that selected 60 degree ranges are representative of 10% of yearly wind conditions and remaining 90% are greedy-optimal

Tables 3.2 and 3.3 show the additional houses that could be powered by the YawGNN and FLORIS controllers, respectively, for each farm and wind direction frequency in the experiments under the assumption that the 60 degree ranges used in the experiments are representative of 10% of the year's wind conditions and that greedy control is optimal for the remainder. These results show that there is a tangible benefit to using YawGNN for control as, for example, a wind farm with 20 turbines could power an additional 28.58 houses per year in wind conditions varying at 1 degree per minute, while FLORIS powers an additional 16.04 houses. With double the rate of change in the wind direction, YawGNN is still able to power an additional 17.88 houses relative to greedy, while FLORIS is no longer usable for real-time control at all.

# Chapter 4

# Discussion

## 4.1 Conclusions

To conclude, we return to answer our research questions:

1. **Can we use a GNN to approximate the FLORIS wake modeling utility for arbitrary turbine layout, turbine yaws, wind direction, and wind speed?**

   We have shown that we are able to estimate power for wind farms with random yaws with comparable accuracy to those with fixed yaws. We have further shown that power estimation using our GNN is significantly faster than power estimation using FLORIS.

2. **Can a trained wind farm GNN be used to compute optimal yaw configurations more efficiently than FLORIS without sacrificing solution quality?**

   For wind farms with the same number of turbines as trained on, we have shown that we can use our GNN to compute optimal yaw configurations that are of comparable FLORIS. More importantly, while the runtime of using FLORIS to compute optimal yaw configurations is quadratic in the number of turbines due to its use of SLSQP, our GNN is able to compute optimal yaw configurations in functionally linear time.

3. **How well is a trained wind farm GNN able to generalize to farms**

**of larger size than it was trained on - both in power estimation and yaw optimization?**

We have shown that a GNN trained on wind farms with between 2 and 10 turbines is able to accurately estimate power for wind farms with up to 100 turbines as long as the grid size is increased to accommodate the additional turbines. Yaw optimization with this GNN is able to improve upon the greedy solution on average, but has not been shown to be able to compute the FLORIS optimal solution.

4. **How well does a trained wind farm GNN perform when used as the model in a model predictive controller for real-time wake steering control of wind farms?**

   We have shown that YawGNN is able to be used for real-time wake steering control of wind farms. While FLORIS is unable to be used for frequencies greater than 3 degrees per minute for wind farms with 5 turbines or for frequencies greater than 1 degree per minute for wind farms with 20 turbines, YawGNN can improve power relative to the greedy solution for wind farms with 5 turbines at frequencies up to 5 degrees per minute and wind farms with 20 turbines at frequencies up to 3 degrees per minute. YawGNN outperformed FLORIS in every example where both methods were applicable.

## 4.2 Future Work

There are many possible directions for future research. First and foremost, it would be beneficial to expand beyond just approximating the FLORIS model and instead (or additionally) train on real-world or CFD data. This would enable the modeling of more complex phenomena such as speed-up and blockage effects, which has never been done in the context of control and optimization.

To improve the performance of both YawGNN and FLORIS on real-time control, work could be done to predict how the wind direction will change rather than just reacting to it.

It would also be useful to couple [31]'s use as a design tool with our GNN's use as a control tool and perform simultaneous control and design optimization. This is

feasible since our model can be differentiated with respect to the downstream and radial distances between turbines (or with respect to the x and y positions directly) in addition to yaw. This could greatly open up the design space for wind farm designers since they would be able to quickly compute wake steering controllers for proposed wind farm layouts, making layouts that were previously unappealing potentially optimal.

To improve the yaw optimization solutions found by YawGNN, we could experiment with different optimization techniques or with optimizing from many different initial conditions in parallel.

While we demonstrated the ability of our model to compute optimal yaw configurations, further work could be done using our model to find optimal paths from initial to final yaw configurations that minimize wake losses in transition when, for example, wind conditions change rapidly.

To improve the combinatorial generalization of power estimation using YawGNN, we could include features representing the physical farm size, such as the length, width, area, or turbines per square meter and train on examples where the physical size of the farm varies.

Finally and most simply, to make these results more directly applicable to industry, one could train our GNN using wind farms with much larger numbers of turbines from the beginning. While it is good that a model trained on wind farms with 2 to 10 turbines can estimate power accurately and compute yaw configurations that improve upon the greedy solution, given more time and computational resources there is no reason not to train on wind farms with 20 or 100 turbines from the beginning in order to potentially compute comparable solutions to FLORIS for much larger farms orders of magnitude faster. This would also potentially enable modeling and control of the largest wind farms with thousands of turbines.

# Bibliography


[1] A2e Data Archive, Office of Energy Efficiency Portal for U.S. Department of Energy, and Renewable Energy. Atmosphere to electrons (a2e) wfip2/lidar.z03.b0, 2016. 2.7

[2] Peter W. Battaglia, Razvan Pascanu, Matthew Lai, Danilo Rezende, and Koray Kavukcuoglu. Interaction networks for learning about objects, relations and physics, 2016. 1.2.4

[3] Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Caglar Gulcehre, Francis Song, Andrew Ballard, Justin Gilmer, George Dahl, Ashish Vaswani, Kelsey Allen, Charles Nash, Victoria Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matt Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. Relational inductive biases, deep learning, and graph networks, 2018. 1.1.3, 1.2.4

[4] Christopher Bay, Jennifer King, Paul Fleming, Rafael Mudafort, and Luis Martínez Tossas. Unlocking the full potential of wake steering: Implementation and assessment of a controls-oriented model. *Wind Energy Science Discussions*, pages 1–20, 05 2019. doi: 10.5194/wes-2019-19. 1.2.3

[5] Atilim Gunes Baydin, Barak A. Pearlmutter, and Alexey Andreyevich Radul. Automatic differentiation in machine learning: a survey. *CoRR*, abs/1502.05767, 2015. URL http://arxiv.org/abs/1502.05767. 1.1.3

[6] James Bleeg. A graph neural network surrogate model for the prediction of turbine interaction loss. *Journal of Physics: Conference Series*, 1618:062054, sep 2020. doi: 10.1088/1742-6596/1618/6/062054. URL https://doi.org/10.1088/1742-6596/1618/6/062054. 1.1.3, 1.2.5

[7] James Bleeg, Mark Purcell, Renzo Ruisi, and Elizabeth Traiger. Wind farm blockage and the consequences of neglecting its impact on energy production. *Energies*, 11:1609, 06 2018. doi: 10.3390/en11061609. 1.1.2, 1.2.5

[8] Global Wind Energy Council. Global wind report 2021, 2021. URL https://gwec.net/global-wind-report-2021/. 1.1.1

[9] DGL. Deep graph library. URL https://www.dgl.ai/. 2.1

[10] 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. doi: https://doi.org/10.1002/we.1810. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/we.1810. 1.2.2

[11] Paul A. Fleming, Pieter M.O. Gebraad, Sang Lee, Jan-Willem van Wingerden, Kathryn Johnson, Matt Churchfield, John Michalakes, Philippe Spalart, and Patrick Moriarty. Evaluating techniques for redirecting turbine wakes using sowfa. *Renewable Energy*, 70:211–218, 2014. ISSN 0960-1481. doi: https://doi.org/10.1016/j.renene.2014.02.015. URL https://www.sciencedirect.com/science/article/pii/S0960148114000950. Special issue on aerodynamics of offshore wind energy systems and wakes. 1.2.2

[12] Center for Climate and Energy Solutions. Renewable energy. URL https://www.c2es.org/content/renewable-energy/. 1.1.1

[13] P. Gebraad, F. Teeuwisse, J. Wingerden, P. Fleming, S. Ruben, Jason R. Marden, and L. Pao. Wind plant power optimization through yaw control using a parametric model for wake effects—a cfd simulation study. *Wind Energy*, 19: 95–114, 2016. 1.2.2, 1.2.3

[14] Pieter Gebraad and J. W. Wingerden. A control-oriented dynamic model for wakes in wind plants. *Journal of Physics: Conference Series*, 524:012186, 06 2014. doi: 10.1088/1742-6596/524/1/012186. 1.2.2

[15] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry, 2017. 1.2.5

[16] Jessica B. Hamrick, Kelsey R. Allen, Victor Bapst, Tina Zhu, Kevin R. McKee, Joshua B. Tenenbaum, and Peter W. Battaglia. Relational inductive bias for physical construction in humans and machines, 2018. 1.2.4

[17] Byeongho Heo, Sanghyuk Chun, Seong Joon Oh, Dongyoon Han, Sangdoo Yun, Gyuwan Kim, Youngjung Uh, and Jung-Woo Ha. Adamp: Slowing down the slowdown for momentum optimizers on scale-invariant weights, 2021. 2.3

[18] Michael Howland, Sanjiva Lele, and John Dabiri. Wind farm power optimization through wake steering. *Proceedings of the National Academy of Sciences*, 116: 201903680, 07 2019. doi: 10.1073/pnas.1903680116. 1.1.2, 1.1.2, 1.2.3

[19] N. Jensen. A note on wind generator interaction. 1983. 1.1.2, 1.2.2, 1.2.3

[20] 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. doi: https://doi.org/10.1002/we.380. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/we.380. 1.2.2, 1.2.3

[21] J. Jonkman, S. Butterfield, W. Musial, and G. Scott. Definition of a 5-mw reference wind turbine for offshore system development, 2009. URL https://digital.library.unt.edu/ark:/67531/metadc894033/. 1.2.1, 2.1, 2.7

[22] Yunzhu Li, Jiajun Wu, Russ Tedrake, Joshua B. Tenenbaum, and Antonio Torralba. Learning particle dynamics for manipulating rigid bodies, deformable objects, and fluids, 2019. 1.2.4

[23] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts, 2017. 2.3

[24] Jason R. Marden, Shalom D. Ruben, and Lucy Y. Pao. A model-free approach to wind farm control using game theoretic methods. *IEEE Transactions on Control Systems Technology*, 21(4):1207–1214, 2013. doi: 10.1109/TCST.2013.2257780. 1.2.3

[25] Luis Martínez Tossas, Jennifer Annoni, Paul Fleming, and Matthew Churchfield. The aerodynamics of the curled wake: A simplified model in view of flow control. *Wind Energy Science Discussions*, pages 1–17, 08 2018. doi: 10.5194/wes-2018-57. 1.2.2, 1.2.3

[26] Amin Niayifar and Fernando Porté-Agel. A new analytical model for wind farm power prediction. *Journal of Physics: Conference Series*, 625:012039, jun 2015. doi: 10.1088/1742-6596/625/1/012039. URL https://doi.org/10.1088/1742-6596/625/1/012039. 1.2.2, 1.2.3

[27] NREL. Floris. version 2.4, 2021. URL https://github.com/NREL/floris. 1.1.2, 1.2.2

[28] Lucy Y. Pao and Kathryn E. Johnson. A tutorial on the dynamics and control of wind turbines and wind farms. In *2009 American Control Conference*, pages 2076–2089, 2009. doi: 10.1109/ACC.2009.5160195. 1.1.2, 1.2.1

[29] Jinkyoo Park and Kincho H. Law. Layout optimization for maximizing wind farm power production using sequential convex programming. *Applied Energy*, 151:320–334, 2015. ISSN 0306-2619. doi: https://doi.org/10.1016/j.apenergy.2015.03.139. URL https://www.sciencedirect.com/science/article/pii/S0306261915004560. 1.2.5

[30] Jinkyoo Park, Soon-Duck Kwon, and Kincho H. Law. A data-driven approach for cooperative wind farm control. In *2016 American Control Conference (ACC)*, pages 525–530, 2016. doi: 10.1109/ACC.2016.7524967. 1.2.3

[31] Junyoung Park and Jinkyoo Park. Physics-induced graph neural network: An application to wind-farm power estimation. *Energy*, 187:115883, 2019. ISSN 0360-

5442. doi: https://doi.org/10.1016/j.energy.2019.115883. URL https://www.sciencedirect.com/science/article/pii/S0360544219315555. 1.1.3, 1.2.5, 1, 2.1, 2.2, 4.2

[32] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009. doi: 10.1109/TNN.2008.2005605. 1.2.4

[33] Connor Schenck and Dieter Fox. Spnets: Differentiable fluid dynamics for deep neural networks, 2018. 1.2.4

[34] H. O. Pörtner D. Roberts J. Skea P.R. Shukla A. Pirani W. Moufouma-Okia C. Péan R. Pidcock S. Connors J. B. R. Matthews Y. Chen X. Zhou M. I. Gomis E. Lonnoy T. Maycock M. Tignor T. Waterfield (eds.) V. Masson-Delmotte, P. Zhai. Ipcc, 2018: Global warming of 1.5°c. an ipcc special report on the impacts of global warming of 1.5°c above pre-industrial levels and related global greenhouse gas emission pathways, in the context of strengthening the global response to the threat of climate change, sustainable development, and efforts to eradicate poverty, 2018. 1.1.1

[35] Tingwu Wang, Renjie Liao, Jimmy Ba, and S. Fidler. Nervenet: Learning structured policy with graph neural networks. In *ICLR*, 2018. 1.2.4