How to craft a deep reinforcement learning policy for wind farm flow control

Elie Kadoche[1,2]   **Pascal Bianchi**[1]   **Florence Carton**[2]   **Philippe Ciblat**[1]   **Damien Ernst**[1,3]
[1]Polytechnic Institute of Paris, 19 Place Marguerite Perey, 91120 Palaiseau, France
[2]TotalEnergies OneTech, 2 Place Jean Millier, 92400 Courbevoie, France
[3]Montefiore Institute, University of Liège, 4000 Liège, Belgium
elie.kadoche@ip-paris.fr

Abstract

Within wind farms, wake effects between turbines can significantly reduce overall energy production. Wind farm flow control encompasses methods designed to mitigate these effects through coordinated turbine control. Wake steering, for example, consists in intentionally misaligning certain turbines with the wind to optimize airflow and increase power output. However, designing a robust wake steering controller remains challenging, and existing machine learning approaches are limited to quasi-static wind conditions or small wind farms. This work presents a new deep reinforcement learning methodology to develop a wake steering policy that overcomes these limitations. Our approach introduces a novel architecture that combines graph attention networks and multi-head self-attention blocks, alongside a novel reward function and training strategy. The resulting model computes the yaw angles of each turbine, optimizing energy production in time-varying wind conditions. An empirical study conducted on steady-state, low-fidelity simulation, shows that our model requires approximately 10 times fewer training steps than a fully connected neural network and achieves more robust performance compared to a strong optimization baseline, increasing energy production by up to 14 %. To the best of our knowledge, this is the first deep reinforcement learning-based wake steering controller to generalize effectively across any time-varying wind conditions in a low-fidelity, steady-state numerical simulation setting.

1   Introduction

1.1   Wind farm flow control

A wind turbine converts wind energy into electricity. As wind passes through the blades, its speed decreases, and turbulence increases for a certain distance, creating wake effects. Within wind farms, wake effects of upstream turbines reduce the power output of downstream turbines (because of lower wind speed) and increase their structural fatigue (because of increased turbulence). Consequently, wake interactions can significantly decrease overall wind farm energy production. Wake losses quantify this impact as the percentage of power loss due to wake-induced interference.

Greedy control aims to maximize the power output of each turbine individually by keeping all turbines aligned with the wind direction, without considering wake interactions. To mitigate the negative impact of wake effects, wind farm flow control (WFFC), i.e., coordinated turbine control, can be implemented. One method, known as wake steering, consists in misaligning upstream turbines in relation to the incoming wind in order to move their wakes away from the downstream turbines. This is accomplished through yaw control, i.e., active rotation of a turbines nacelle around its vertical axis.

Figure 1 displays a simulation of wake steering applied to a three turbines wind farm with the wind coming from the west. The darker areas behind each machine correspond to the wake effects. The first two (upstream) machines are slightly misaligned with the wind direction to redirect their wake away from the third (downstream) turbine. In this example, the overall farm energy production is improved by 14 % compared to a standard solution where the three turbines would be aligned with the wind. However, as wind farms grow larger, the number of wake interactions increases, leading to a high-dimensional control problem with complex spatial dependencies. In parallel, time-varying and uncertain wind conditions pose additional challenges, as yaw actuators are subject to rotational constraints that limit how quickly and frequently turbines can reorient to shifting flow directions. Together, these factors make the implementation of WFFC solutions increasingly challenging.
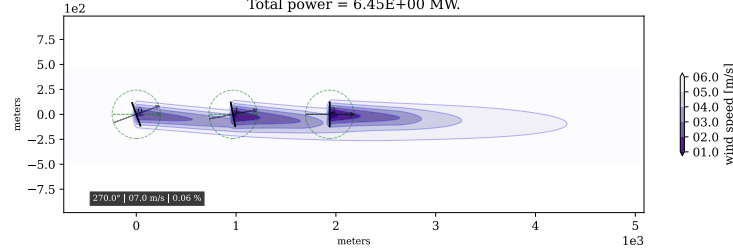

Figure 1: Example of WFFC on a three turbines wind farm viewed from above. The first two turbines are slightly misaligned with the wind to steer their wake away from the third turbine, maximizing energy production.

1.2 Related works

Wake steering is traditionally implemented using lookup tables (LUTs), where precomputed yaw settings are applied based on discrete wind conditions [1]. These methods fail to adapt to real-time wind dynamics, leading to suboptimal power production. Model-based approaches like model predictive control (MPC) [2] offer some adaptability but heavily depend on the accuracy of their underlying wind model and require significant computational resources to solve optimization problems in real time. To overcome these limitations, data-driven learning-based approaches provide more adaptive and robust control strategies, capable of continuously adjusting yaw settings in response to dynamic and uncertain wind conditions.

In particular, model-free reinforcement learning (RL) has emerged as a powerful alternative to traditional wake steering methods [3, 4, 5]. Model-free RL algorithms are inherently more robust to wind farm model uncertainties and learn solely from experience, enabling the discovery of new and unexpected control strategies, which is especially valuable for large-scale wind farms. Additionally, RL seamlessly integrates multi-objective optimization, making it easy to incorporate fatigue reduction alongside power maximization.

Despite these promising advantages, existing RL-based wake steering methods face significant limitations in terms of scalability, sample efficiency, wind dynamics consideration, and wind conditions generalization. While some studies address one or more of these challenges, none comprehensively tackle all of them together. Most existing research on RL for yaw control assumes quasi-constant wind directions and small wind farms [6, 7]. While some studies consider time-varying wind directions, they remain constrained to a limited range [8, 9], leaving uncertainty about their ability to generalize across diverse wind conditions, which is a critical requirement for real-world deployment. In Dong et al. [10], wind conditions are segmented into discrete intervals, each controlled by a separate RL policy. In this work, we aim to develop a single policy generalizing across all wind conditions.

To enhance learning speed and efficiency, some studies investigate the use of multi-agent RL for WFFC [9, 11]. However, these approaches often rely on feed forward neural networks (FNNs), which struggle to fully capture the complexity of WFFC. Li et al. [12] have recently introduced promising results with the use of graph transformers to build deep learning (DL)-based surrogate models for wind power predictions. In this work, we go further by leveraging graph attention networks (GATs) and self-attention mechanisms within a single-agent RL policy, demonstrating significant improvements in sample efficiency, learning performance and generalization.

1.3 Contributions

In this work, we develop a single-agent deep RL policy for wind farm wake steering that generalizes robustly across time-varying and noisy wind conditions. Our contributions are threefold. (1) We introduce a novel RL architecture combining GATs and multi-head self-attention (MHSA) which improves by about a factor 10 the sample efficiency of a traditional FNN and achieves superior performance compared to both a FNN and a traditional GAT. (2) We propose a new training methodology and reward design that enable the policy to generalize across the full 360° range of wind directions under unsteady and noisy conditions. (3) And we employ a proximal policy optimization (PPO) actor-critic framework with a von Mises policy head to compute yaw angles, achieving up to 14 % higher energy capture than a standard wind-tracking strategy in a low-fidelity steady-state simulator.

2 Markov Decision Process

A wind farm is a set of $N \in \mathbb{N}^*$ turbines, indexed by $i \in \{0, \ldots, N-1\}$, each located at fixed spatial coordinates (x^i, y^i) and characterized by a rotor diameter d [m]. The control problem is defined over an episode consisting of $T \in \mathbb{N}^*$ discrete time steps, indexed by $t \in \{0, \ldots, T-1\}$, during which the yaw angle of each turbine is adjusted to optimize performance. The WFFC problem is formalized as a Markov decision process (MDP) represented by a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$, with: $\mathcal{S}$ the set of states, $\mathcal{A}$ the set of actions, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ the transition function, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ the reward function, and $\gamma \in [0, 1]$ a discount factor.

2.1 State

At each time step t, the wind is characterized by a direction $K_t \in [0, 360]$ degrees and a speed $V_t \in [V_{\min}, V_{\max}]$ m/s. This defines the free-stream wind field, which is assumed to be spatially homogeneous across the farm prior to any wake-induced disturbances. To account for measurement or forecasting uncertainty, we also define K_t' and V_t' as the observed or predicted wind direction and speed, respectively, which may differ from the true values due to sensor noise or forecast error. Each turbine i has an absolute nacelle orientation $\beta_t^i \in [0, 360]$ degrees, and a yaw angle $\alpha_t^i \in [-180, 180]$ degrees. As described in Sub-Figure 2a, the yaw angle is the offset between the absolute nacelle orientation and the wind direction, i.e., $\alpha_t^i = (K_t - \beta_t^i + 180) \mod 360 - 180$. The state is $s_t = (X_{W_t}, X_{F_t}, X_{Y_t}, X_{L_t})$ where:

- $X_{W_t} = (K_t', V_t')$ represents the current, measured wind conditions;
- $X_{F_t} = (K_{t+l}', V_{t+l}')_{l \in \{1,2,3\}}$ represents a wind forecast on the next three time steps;
- $X_{Y_t} = (\beta_t^i)_{i \in \{0, \ldots, N-1\}}$ represents the current absolute orientations of each turbine;
- $X_{L_t} = (x^i, y^i, d)_{i \in \{0, \ldots, N-1\}}$ represents the static layout and rotor diameter of each turbine.

2.2 Action

The wake steering controller is characterized by a policy π_θ parameterized by θ, computing the yaw settings from the state such that $\pi_\theta(s_t) = \mathbf{a}_t$. The action $\mathbf{a}_t = (a_t^i)_{i \in \{0, \ldots, N-1\}}$ is the vector of each turbine individual yaw setting. Each action a_t^i corresponds to the rotational movement of turbine i between time step t and $t+1$ and is bounded in $[-20, 20]$ degrees due to mechanical constraints of the yaw actuators. Before action a_t^i is applied (Figure 2a), the yaw angle is α_t^i and the absolute orientation is β_t^i. When action is applied (Figure 2b), the turbine is rotated, giving an updated orientation β_{t+1}^i and an updated yaw angle $\tilde{\alpha}_t^i$. At the end of the time step t (Figure 2c), wind direction evolves from K_t to K_{t+1} and the next yaw angle α_{t+1}^i is computed accordingly.

2.3 Transition

At the beginning of each episode, initial wind direction and speed are sampled from uniform distributions such that $K_0 \sim \mathcal{U}(0, 360)$ and $V_0 \sim \mathcal{U}(V_{\min}, V_{\max})$, respectively. Initial yaw angles are sampled from a uniform distribution such that $\alpha_0^i \sim \mathcal{U}(-20, 20) \; \forall i \in \{0, \ldots, N-1\}$. The absolute nacelle orientations β_0^i are then computed based on K_0. The wind farm layout remains fixed throughout all episodes. At each time step t, and for each turbine i:

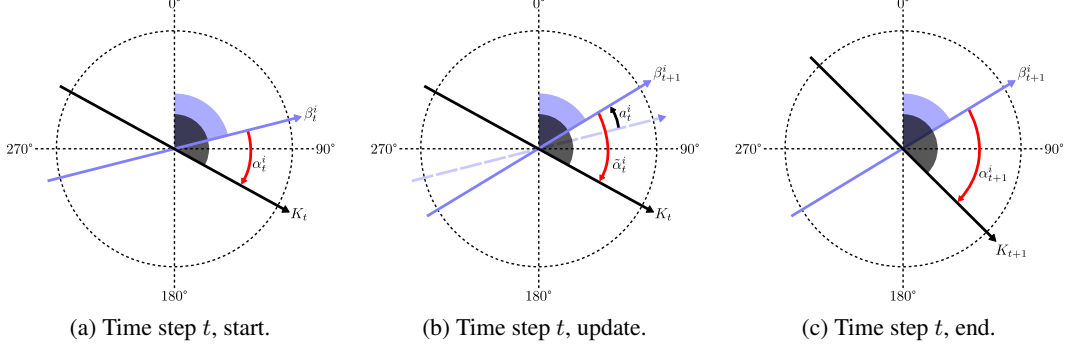| (a) Time step t, start. | (b) Time step t, update. | (c) Time step t, end. |

Figure 2: Graphical representation of a turbine i at time step t with wind direction K_t. After action a_t^i is being applied, the turbine has an updated yaw angle $\tilde{\alpha}_t^i$. At the end of the time step, wind direction evolves from K_t to K_{t+1} and the next yaw angle α_{t+1}^i is computed.

1. the policy computes the yaw settings such that $\pi_\theta(s_t) = \mathbf{a}_t$;
2. absolute orientations are updated $\beta_{t+1}^i = (\beta_t^i + a_t^i) \mod 360$;
3. yaw angles are updated $\tilde{\alpha}_t^i = (K_t - \beta_{t+1}^i + 180) \mod 360 - 180$;
4. the power output of the wind farm P_t in megawatts (MW) is computed;
5. a reward r_{t+1} is computed, and wind conditions evolve from (K_t, V_t) to (K_{t+1}, V_{t+1});
6. yaw angles are updated $\alpha_{t+1}^i = (K_{t+1} - \beta_{t+1}^i + 180) \mod 360 - 180$;

Power computation is based on the current wind conditions (K_t, V_t) and the updated yaw angles $\tilde{\alpha}_t^i$. In this work, we use a steady-state, low-fidelity wind farm simulator to compute the power outputs and a simple model to generate wind data time series, both later described in Sub-Section 4.1. To maintain a consistent and valid discretization of the continuous WFFC problem, we account for turbine rotational constraints by assuming that reorientation occurs rapidly relative to the time step duration, and that wind conditions remain quasi-stationary over each time step.

2.4 Reward

At each time step t, the reward r_{t+1} (Equation 1) is the weighted sum of two terms: one for invalid policies and one for power maximization, such that

$$r_{t+1} = w_0 r_{t+1}^{\text{invalid}} + w_1 r_{t+1}^{\text{power}}. \tag{1}$$

The objective of the r_{t+1}^{invalid} term (Equation 2) is to penalize the total reward when some yaws are outside $[-20, 20]$. Indeed, outside this interval, turbines are shut down for safety reason, resulting in no power output. This makes the power maximization reward uninformative. Therefore, this term is used to guide the policy towards good control strategies, i.e., keeping turbines close to the wind direction. It is defined as

$$r_{t+1}^{\text{invalid}} = \frac{-1}{N} \sum_{i=0}^{N-1} \left(\left(\frac{|\tilde{\alpha}_t^i|}{180} \right)^3 \mathbf{1}_{\tilde{\alpha}_t^i \notin [-20, 20]} \right). \tag{2}$$

The objective of the r_{t+1}^{power} term (Equation 3) is to maximize power production relative to a baseline, using the power ratio $\Delta_{P_t} = (P_t - \bar{P}_t)/\bar{P}_t$ with $\bar{P}_t$ the baseline power output. Unlike using the absolute power output as a reward - which can bias the agent toward high-production wind conditions only - this ratio normalizes improvements relative to a baseline and ensures the agent optimizes wake steering across all conditions, including those where wake losses are less significant. We use as a baseline a perfect wind tracking controller that does not perform any wake steering. It is not subject to the rotational constraints of the turbines, as it always performs simulations with all turbines aligned with the exact wind direction K_t. The baseline wake losses denoted $\bar{\mathcal{L}}_t$ gives some insights about the complexity and importance of WFFC for the given wind conditions. High values indicate a significant reduction in energy production, making wake steering essential.

4

The optimal magnitude of Δ_{P_t} depends on the wake losses: a near-zero ratio can be optimal when wake losses are low (making WFFC unnecessary) but suboptimal when wake losses are high (making WFFC necessary). To address this, we introduce an exponential scaling term, parameterized by p, that adjusts the reward based on the baseline wake losses $\bar{\mathcal{L}}_t$. This term ensures a balanced reward across different wind conditions by restricting the power ratio when the wake losses are significant. Additionally, if power production falls below the baseline, the agent is penalized with a negative reward. An ablation study is given in Figure 9, in Appendix. It is defined as

$$r_{t+1}^{\text{power}} = \Delta_{P_t} \mathbf{1}_{\Delta_{P_t} < 0} + \exp(-p\bar{\mathcal{L}}_t)\Delta_{P_t} \mathbf{1}_{\Delta_{P_t} \geq 0}. \tag{3}$$

3 Models

We train and compare three models in a single-agent, continuous action space setting: a FNN-based model named **V0 model**; a GAT named **V1 model**; and our contribution, the **V2 model**, an attention-based neural network. Each model follows an actor-critic framework, taking the state s_t as input and giving both an actor distribution $(\bar{\mu}_t, \bar{\kappa}_t)$ and a critic value v as outputs. The actor distribution independently parameterizes a von Mises distribution for each turbine, with the location parameters $\bar{\mu}_t := (\mu_t^0, \mu_t^1, \ldots, \mu_t^{N-1})$ and the concentration parameters $\bar{\kappa}_t := (\kappa_t^0, \kappa_t^1, \ldots, \kappa_t^{N-1})$.

During training, turbine actions are sampled from their respective von Mises distribution: $a_t^i \sim \mathcal{V}(\mu_t^i, \kappa_t^i) \, \forall i \in \{0, 1, \ldots, N-1\}$. During testing, each turbine's action is set directly to its location parameter: $a_t^i = \mu_t^i, \, \forall i \in \{0, 1, \ldots, N-1\}$. Due to the inherent symmetry in the WFFC problem, if an effective solution exists near the lower bound of the action space, a corresponding solution near the upper bound is often equally viable. By modeling actions with a circular distribution, like the von Mises, we ensure that the policy can explore these equivalent solutions efficiently. It promotes a more effective and balanced exploration.

To ensure a fair and meaningful comparison, we use approximately the same number of parameters for each model: around 22 million of parameters each. More details are given in the sub-Section A.1 of the Appendix. A hyperbolic tangent activation function, scaled by π, is used for $\bar{\mu}_t$ to ensure bounded actions in $[-\pi, \pi]$. Actions are later denormalized to the turbine's operational range $[-20, 20]$ before being sent to the environment. A softplus activation function is used for $\bar{\kappa}_t$ to ensure values strictly superior to 1. A linear activation function is used for the critic output.

3.1 Model V0

The V0 model (Figure 3) is a FNN composed exclusively of fully connected (FC) layers. Its input is a single concatenated vector comprising the current wind data, wind forecast, and the absolute orientations of the turbines. The layout feature vector X_{L_t} is excluded from the inputs as it remains constant across all episodes. Instead, the model is expected to implicitly learn spatial relationships between turbine coordinates, wind flow, and wake effects during training. Most existing approaches similarly rely on FNNs that process concatenated inputs. However, this strategy may be inefficient, as it requires the model to simultaneously infer complex spatial and temporal dependencies from a high-dimensional, entangled input vector without explicit structural guidance.
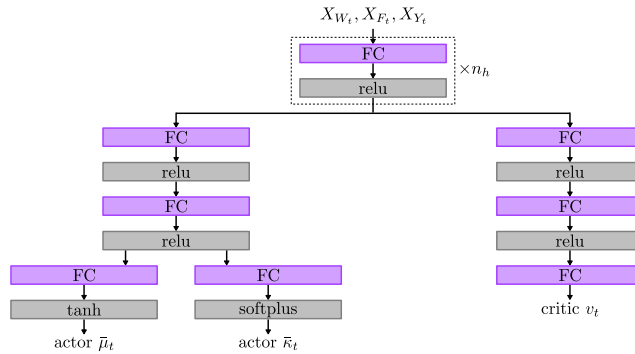

Figure 3: V0 model, a FNN-based architecture.

5

3.2   Model V1

The V1 model (Figure 4) is based on GATs [13]. The input of the V1 model is a graph $g_1(X_{W_t}, X_{F_t}, X_{Y_t}, X_{L_t})$ built from the state. Each node in the graph corresponds to a turbine at a fixed position determined by the layout. The node feature vector for turbine i consists in $(X_{W_t}, X_{F_t}, \beta_t^i)$, which includes current wind data, wind forecast and the turbine's orientation. A directed edge exists from turbine i to turbine j if the distance between them is less than eight turbine diameters and if turbine i is upstream to j. We use a distance of eight turbine diameters to balance wake interaction modeling and graph complexity. Each edge feature vector encodes the normalized distance and relative angle between connected turbines with respect to the wind direction. Graph neural networks (GNNs) have been successfully applied in deep surrogate modeling [12] and various wind farm analysis tasks but remain relatively underexplored for direct WFFC optimization. Still, GNNs may not be optimal since certain data (like wind conditions) are duplicated across all turbine nodes, potentially leading to redundancy.
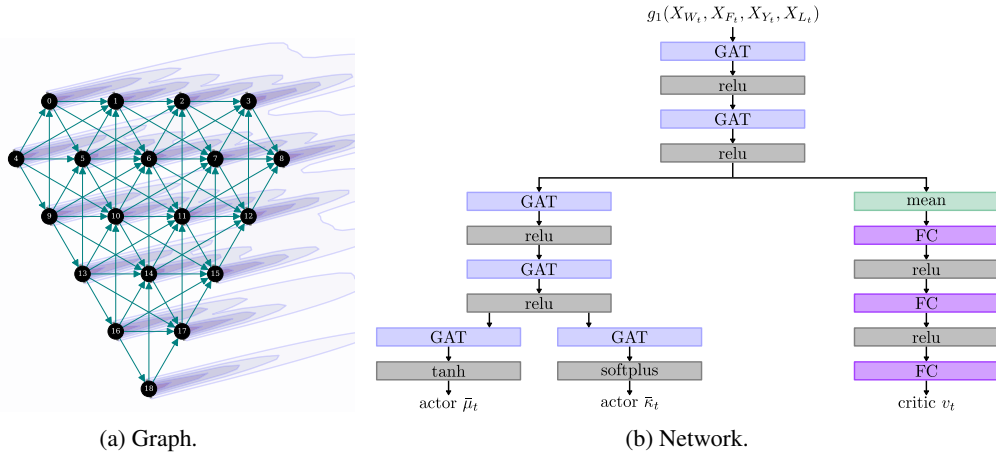

(a) Graph.   (b) Network.

Figure 4: V1 model, based on GATs. Sub-Figure 4a illustrates an example input graph with a wind direction of 256°, where wake effects are shown in the background. Sub-Figure 4b presents the architecture of the V1 model.

3.3   Model V2

Our proposed architecture, described as the V2 model (Figure 5), leverages both a GAT and MHSA blocks Vaswani et al. [14]. To better exploit the multi-modality of the WFFC problem, inputs are split in four different embeddings. 1) A FNN is used to create the wind embedding E_{W_t}. 2) A FNN is used to create the forecast embedding E_{F_t}. 3) A GAT is used to create each turbine positional encoding $E_{pe_t}^i$. The input is a graph similar to the one built by the V1 model (Sub-Section 3.2), without wind speed neither wind forecast. 4) A FNN is used to create each turbine specific embedding $E_{Y_t}^i$ from turbine orientations. The final embedding of each turbine is the sum of all these embeddings. Whereas wind and forecast embeddings are shared between all turbines, positional and turbine embeddings are specific for each turbine. In the context of WFFC, self-attention captures relationships between turbines by identifying which ones are most relevant for yaw control. It allows the model to consider all turbines simultaneously and understand how wake effects propagate across the farm. The multi-head mechanism enhances this by providing multiple perspectives on these interactions. In natural language processing, positional encoding is straightforward, as it follows word order in a sentence. In a wind farm, however, turbine positions depend on wind conditions, yaw angles, and wake effects, making encoding more complex. To address this, we use a GAT to learn positional embeddings, capturing spatial dependencies more effectively. By representing the wind farm as a graph, we encode expert knowledge into the model and provide a structured representation of wake interactions, accelerating learning.
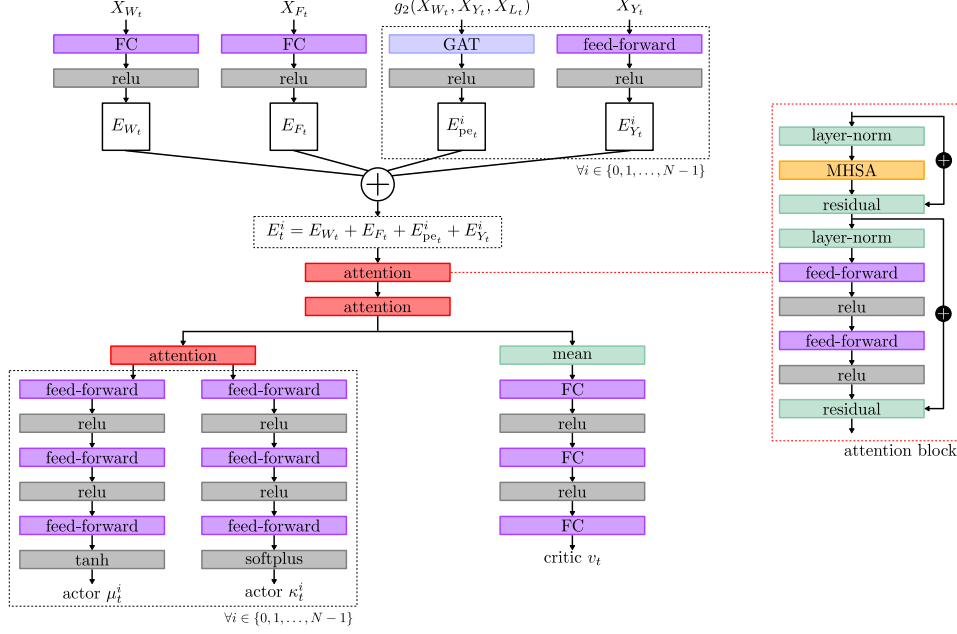
6

Figure 5: V2 model architecture, incorporating GAT and MHSA blocks. FC layers refer to standard dense layers applied once to the input, while feed-forward layers apply the same FC layer independently (a) to each turbine's embedding, (b) in attention blocks, (c) and to each turbine's embedding in the actor branch, after the last attention block.

4 Simulations

4.1 Experimental setting

We employ episodic RL with a horizon of $T = 18$ time steps, each lasting 10 minutes, resulting in a 3 hours control period for the turbines. The reward loss parameter is set to $p = 3$ and the reward weights are $w_0 = 1$ and $w_1 = 100$. States are normalized to the range $[-1, 1]$: wind speeds are scaled accordingly and angles are converted to their sine and cosine representations. We consider a wind farm of $N = 19$ turbines and a custom diamond layout as displayed in Figure 4a, with a distance of four turbines diameters between a turbine and its closest neighbors. Numerical simulations are conducted with FLOw Redirection and Induction in Steady State (FLORIS) [15], a steady-state, low-fidelity simulator developed by National Renewable Energy Laboratory (NREL).

We consider low speed winds, i.e., $V_{\min} = 3$ m/s and $V_{\max} = 10$ m/s because this is where WFFC is the most beneficial for energy production (wake losses have a greater impact). At each time step step $t \geq 1$, we use a simple auto-regressive moving average (ARMA) process of order 1 to generate wind data. The direction is computed such that $K_t = (\epsilon_t + K_{t-1} + 0.1\epsilon_{t-1}) \mod 360$ with $\epsilon_t \sim \mathcal{N}(0, 9)$. The speed is computed such that $V_t = \epsilon'_t + V_{t-1} + 0.1\epsilon'_{t-1}$ with $\epsilon'_t \sim \mathcal{N}(0, 0.01)$. A mirroring function is used for the generated speeds to ensure that values stay in $[V_{\min}, V_{\max}]$. Noisy wind data is obtained by perturbing the original values with noise sampled from uniform distributions: $K'_t = K_t + \epsilon_K$, where $\epsilon_K \sim \mathcal{U}(-3, 3)$, and $V'_t = V_t + \epsilon_V$, where $\epsilon_V \sim \mathcal{U}(-0.1, 0.1)$.

During testing, we use three benchmarks. **Standard**: simple benchmark keeping every turbine aligned as possible with the current measured wind direction K'_t. It does not perform any optimization and does not rely on the wind forecast. **Gauss-Seidel (GS)**: good optimized benchmark, introduced by Fleming et al. [16]. It optimizes turbine yaw settings sequentially, where the initial solution is computed with the standard solution. Then, it sequentially optimizes each turbine from upstream to downstream, keeping the others fixed, by performing a grid search over 40 discretized yaw angles. It does not rely on the wind forecast. **Heuristic**: strong optimized benchmark, introduced by Kadoche et al. [17]. It consists in an improved version of the GS solution, where the objective function is augmented by a heuristic. It does rely on the wind forecast to optimize its solutions on a given horizon, making the comparison with our models more relevant.

7

4.2 Proximal policy optimization

We train each model using a PPO [18] actor-critic method and generalized advantage estimator (GAE) [19]. We use a custom implementation of PPO and a custom RL environment. By parallelizing experience collection in our PPO implementation and vectorizing power computations in the FLORIS simulator, we achieve a 70 speedup in training. Specifically, training the V2 model takes approximately two hours, whereas without parallelization and vectorization, the same training would require 140 hours. For reproducibility purpose, more details regarding the training times (Figure 8), the hyperparameters (Table 1) and our PPO implementation (sub-Section A.5) are given in Appendix.

At each training step, we simulate 360 independent episodes of 3 hours, resulting in 6,480 time steps. To ensure comprehensive coverage of all possible wind conditions, each of the 360 episodes has a different initial wind direction. Each episode is initialized with a wind direction sampled from a discrete set of 360 distinct values, uniformly distributed between 0° and 360°. More specifically, wind directions are sampled in 1° increments, such that the first episode has an initial direction in $[0, 1]$ degrees, the second episode in $[1, 2]$, etc. It ensures that the entire directional space is covered, speeding up generalization and mitigating sampling biases during training.

The discount factor γ determines how much future rewards influence current decisions, where a higher γ prioritizes long-term optimization. Although our goal is to optimize long-term energy production, we use a small discount factor ($\gamma = 0.1$) due to the steady-state nature of the low-fidelity simulation. If training were conducted on a higher-fidelity simulation, where state transitions introduce stronger temporal dependencies, increasing the discount factor would be necessary to properly account for long-term effects.

4.3 Results

Each model is trained for 150 steps, corresponding to a total of 972,000 simulated time steps. Training is conducted across 10 different random seeds to ensure robustness and account for variability in learning performance. The training curves, shown in Figure 6, highlight key differences between models. The V0 model exhibits high variance in the early stages and converges significantly slower than the V1 and V2 models. Because input data is concatenated into a single vector, the V0 model struggles to learn an effective policy efficiently, likely due to the lack of an explicit structural representation of the data, requiring more training steps to stabilize. In contrast, the V2 model demonstrates superior stability and faster convergence, ultimately outperforming both the V0 and V1 models in terms of learning efficiency and final performance.
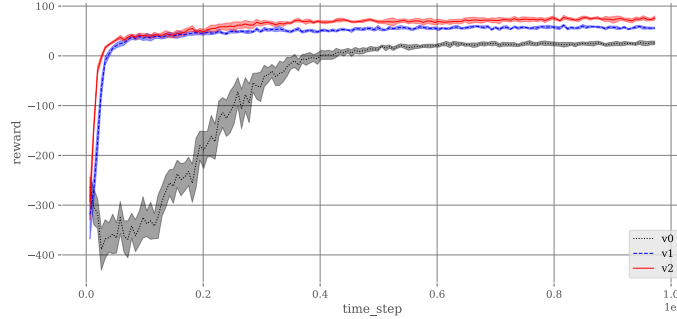

Figure 6: Training curves of each model, showing mean and variance over 10 different seeds. The V1 and V2 models have much faster and stable convergence compared to the V0 model. And the V2 model achieves better performance compared to the V0 and V1 models.

To evaluate the generalization of each solution across all wind conditions, we test them on 360 wind directions, sampled in 1-degree increments. For each direction, we run 10 independent test episodes of 18 time steps, using random seeds not used during training. This ensures that test episodes remain distinct from training and provide comprehensive directional coverage. For each episode, we compute each solution's cumulative power production and quantify its improvement over the standard wind-tracking solution. We then report the mean and variance of these improvements across the 10 seeds for each wind direction and present the results in Figure 7.

The V2 model consistently increases wind farm energy production, achieving gains of up to 14 % in high wake-loss scenarios. It outperforms both the V0 and V1 models across nearly all conditions. Although the heuristic baseline delivers strong performance, its results exhibit greater variance, making it less reliable. The GS approach performs poorly and, in some cases, even yields lower power output than the standard solution. This demonstrates that wind direction can shift too rapidly relative to yaw constraints, making long-term optimization essential for effective control. The V2 model successfully leverages noisy wind forecasts to improve long-term performance. Notably, in strong wake conditions, the V2 model outperforms the heuristic controller while being roughly 200 times more computationally efficient. However, when wake losses are low, power gains are more limited, and the V2 model underperforms compared to the heuristic. The underlying cause of this discrepancy remains unclear: it may come from architectural limitations or from the reward function design. Future works should investigate these factors to further refine the model's performance.

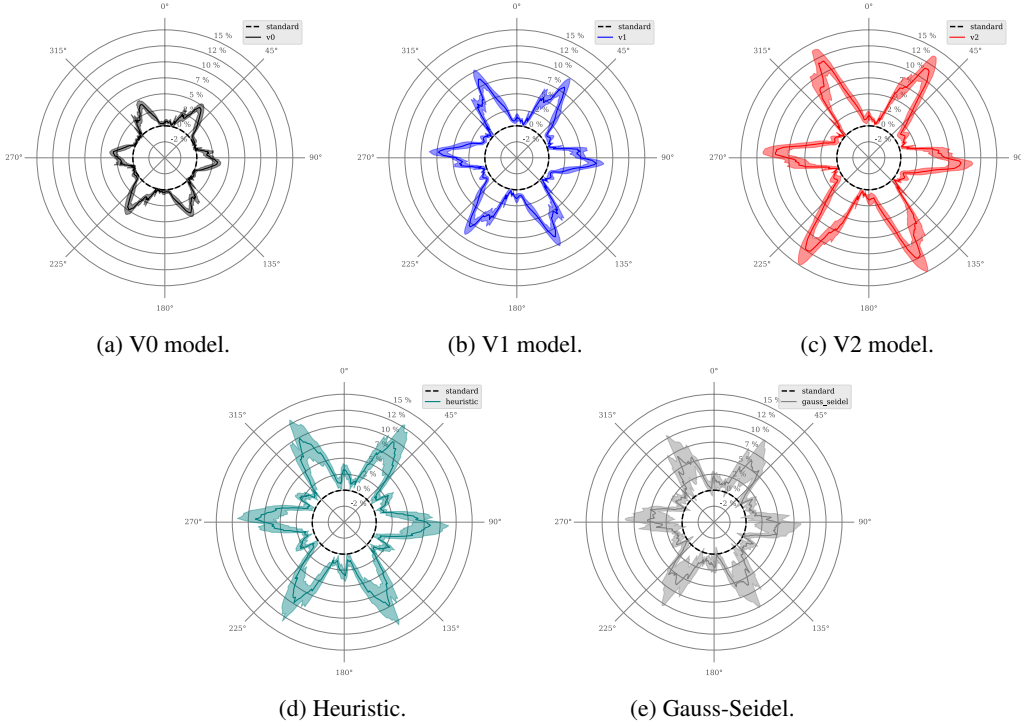| (a) V0 model. | (b) V1 model. | (c) V2 model. |

| (d) Heuristic. | (e) Gauss-Seidel. |

Figure 7: Performance of each solution relative to the standard benchmark. The V2 model significantly outperforms the V0 and V1 models as well as the GS solution. It achieves performance comparable to the heuristic, but with lower variance and higher gains in strong wake loss conditions.

5 Conclusion

In this work, we introduced a novel deep RL architecture for wake steering, based on GATs and MHSA, improving by approximately a factor 10 the sampling efficiency of a FNN. Once trained with PPO, our model computes the yaw angles of each turbine and achieves more robust performance than a strong optimization baseline, increasing energy production by up to 14 %. To the best of our knowledge, this work is the first to achieve complete generalization over time-varying wind conditions, thanks to a novel reward function and training strategy. However, while this work provides encouraging evidence of the potential for deep RL for robust wake steering, the results remain empirical and are based on simplified, steady-state, low-fidelity wake models. For real-world deployment, future work should incorporate turbine fatigue considerations to ensure long-term structural integrity and validate the approach in higher-fidelity and unsteady flow environments that better capture realistic wind dynamics. As the problem grows in complexity, the strengths of RL - such as its ability to optimize over long horizons, adapt to uncertain dynamics, and operate without explicit system models - should further reinforce its suitability for wake steering control.

9

References

[1] P. Fleming et al. "Field test of wake steering at an offshore wind farm". In: *Wind Energy Science* 2.1 (2017), pp. 229–239. DOI: 10.5194/wes-2-229-2017. URL: https://wes.copernicus.org/articles/2/229/2017/.

[2] Dongran Song et al. "Maximum power extraction for wind turbines through a novel yaw control solution using predicted wind directions". In: *Energy Conversion and Management* 157 (2018), pp. 587–599. ISSN: 0196-8904. DOI: https://doi.org/10.1016/j.enconman.2017.12.019. URL: https://www.sciencedirect.com/science/article/pii/S0196890417311676.

[3] Tuhfe Göçmen et al. "Data-driven wind farm flow control and challenges towards field implementation: A review". In: *Renewable and Sustainable Energy Reviews* 216 (2025), p. 115605. ISSN: 1364-0321. DOI: https://doi.org/10.1016/j.rser.2025.115605. URL: https://www.sciencedirect.com/science/article/pii/S1364032125002783.

[4] Jaime Liew et al. "Model-free closed-loop wind farm control using reinforcement learning with recursive least squares". In: *Wind Energy* (2023).

[5] Claire Bizon Monroc et al. "Actor Critic Agents for Wind Farm Control". In: *2023 American Control Conference (ACC)*. 2023, pp. 177–183. DOI: 10.23919/ACC55779.2023.10156453.

[6] Hongyang Dong, Jincheng Zhang, and Xiaowei Zhao. "Intelligent wind farm control via deep reinforcement learning and high-fidelity simulations". In: *Applied Energy* 292 (2021), p. 116928. DOI: 10.1016/j.apenergy.2021.116928.

[7] P. Stanfel et al. "A Distributed Reinforcement Learning Yaw Control Approach for Wind Farm Energy Capture Maximization*". In: *2020 American Control Conference (ACC)*. 2020, pp. 4065–4070. DOI: 10.23919/ACC45564.2020.9147946.

[8] Grigory Neustroev et al. "Deep Reinforcement Learning for Active Wake Control". In: *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems*. AAMAS '22. Virtual Event, New Zealand: International Foundation for Autonomous Agents and Multiagent Systems, 2022, pp. 944–953. ISBN: 978-145-039-2-1-3-6.

[9] Elie Kadoche et al. "MARLYC: Multi-Agent Reinforcement Learning Yaw Control". In: *Renewable Energy* 217 (2023), p. 119129. ISSN: 0960-1481. DOI: https://doi.org/10.1016/j.renene.2023.119129. URL: https://www.sciencedirect.com/science/article/pii/S0960148123010431.

[10] Qiang Dong et al. "Deep reinforcement learning-based adaptive yaw control for wind farms in fluctuating winds". In: *Physics of Fluids* 37.4 (Apr. 2025), p. 047157. ISSN: 1070-6631. DOI: 10.1063/5.0267200. eprint: https://pubs.aip.org/aip/pof/article-pdf/doi/10.1063/5.0267200/20501725/047157_1_5.0267200.pdf. URL: https://doi.org/10.1063/5.0267200.

[11] Venkata Ramakrishna Padullaparthi et al. "FALCON- FArm Level CONtrol for wind turbines using multi-agent deep reinforcement learning". In: *Renewable Energy* (2021). ISSN: 0960-1481. DOI: 10.1016/j.renene.2021.09.023.

[12] Siyi Li et al. "Learning to optimise wind farms with graph transformers". In: *Applied Energy* 359 (2024), p. 122758. ISSN: 0306-2619. DOI: https://doi.org/10.1016/j.apenergy.2024.122758. URL: https://www.sciencedirect.com/science/article/pii/S0306261924001417.

[13] Petar Velikovi et al. "Graph Attention Networks". In: *International Conference on Learning Representations*. 2018. URL: https://openreview.net/forum?id=rJXMpikCZ.

[14] Ashish Vaswani et al. "Attention is All you Need". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[15] NREL. *FLORIS. Version 4.2.2*. 2021. URL: https://github.com/NREL/floris.

[16] Paul A. Fleming et al. "Serial-Refine Method for Fast Wake-Steering Yaw Optimization". In: *Journal of Physics: Conference Series* 2265.3 (May 2022), p. 032109. DOI: 10.1088/1742-6596/2265/3/032109. URL: https://doi.org/10.1088/1742-6596/2265/3/032109.

[17] Elie Kadoche et al. "On the importance of wind predictions in wake steering optimization". In: *Wind Energy Science* 9.7 (2024), pp. 1577–1594. DOI: 10.5194/wes-9-1577-2024. URL: https://wes.copernicus.org/articles/9/1577/2024/.

[18] John Schulman et al. *Proximal Policy Optimization Algorithms*. 2017. arXiv: `1707.06347` `[cs.LG]`.

[19] John Schulman et al. "High-Dimensional Continuous Control Using Generalized Advantage Estimation". In: *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2016. URL: `http://arxiv.org/abs/1506.02438`.

[20] J. King et al. "Control-oriented model for secondary effects of wake steering". In: *Wind Energy Science* 6.3 (2021), pp. 701–714. DOI: `10.5194/wes-6-701-2021`. URL: `https://wes.copernicus.org/articles/6/701/2021/`.

[21] Evan Gaertner et al. "IEA Wind TCP Task 37: Definition of the IEA 15-Megawatt Offshore Reference Wind Turbine". In: (Mar. 2020). DOI: `10.2172/1603478`. URL: `https://www.osti.gov/biblio/1603478`.

[22] Antonin Raffin et al. "Stable-Baselines3: Reliable Reinforcement Learning Implementations". In: *Journal of Machine Learning Research* 22.268 (2021), pp. 1–8. URL: `http://jmlr.org/papers/v22/20-1364.html`.

[23] Eric Liang et al. "RLlib: Abstractions for Distributed Reinforcement Learning". In: *International Conference on Machine Learning (ICML)*. 2018. URL: `https://arxiv.org/pdf/1712.09381`.

[24] Zhanghao Wu et al. "RLlib Flow: Distributed Reinforcement Learning is a Dataflow Problem". In: *Conference on Neural Information Processing Systems (NeurIPS)*. 2021. URL: `https://proceedings.neurips.cc/paper/2021/file/2bce32ed409f5ebcee2a7b417ad9beed-Paper.pdf`.

[25] Charles R. Harris et al. "Array programming with NumPy". In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. ISSN: 1476-4687. DOI: `10.1038/s41586-020-2649-2`. URL: `https://doi.org/10.1038/s41586-020-2649-2`.

[26] Philipp Moritz et al. "Ray: A Distributed Framework for Emerging AI Applications". In: *CoRR* abs/1712.05889 (2017). arXiv: `1712.05889`. URL: `http://arxiv.org/abs/1712.05889`.

A Appendix

A.1 Detailed architectures

V0 model

The V0 model, a FNN-based architecture, is presented in Figure 3. In this work, $n_h = 2$ and the two shared FC layers have output sizes of 1024 and 4096. The shared actor branch comprises two FC layers with output sizes of 2048 and 256. The $\bar{\mu}_t$ and $\bar{\kappa}_t$ actor branches each contain a single FC layer with output size of N. The critic branch consists of three FC layers with output sizes of 2048, 256, and 1.

V1 model

The V1 model, a GAT, is presented in Figure 4. In this work, the two shared GAT layers have three attention heads and output sizes of 1024. The shared actor branch comprises two GAT layers with three attention heads and output sizes of 128 and 64. The $\bar{\mu}_t$ and $\bar{\kappa}_t$ actor branches have both a single GAT layer with one attention head, giving a scalar output for each node (i.e., for each turbine). The critic branch consists of three FC layers with output of sizes 128, 64, and 1.

V2 model

The V2 model, an attention-based neural network, is presented in Figure 5. In this work, each embedding layer has an output size of 256 and the GAT used for positional encoding has a single attention head. The three attention blocks consist of a MHSA layer with three attention heads and an output size of 256, followed by two feed-forward layers: one increases the size four times and the other restores it. Both actor branches contain three feed-forward layers with output sizes of 128, 64, and 1. The critic branch follows the same structure, with three FC layers of output sizes 128, 64, and 1.
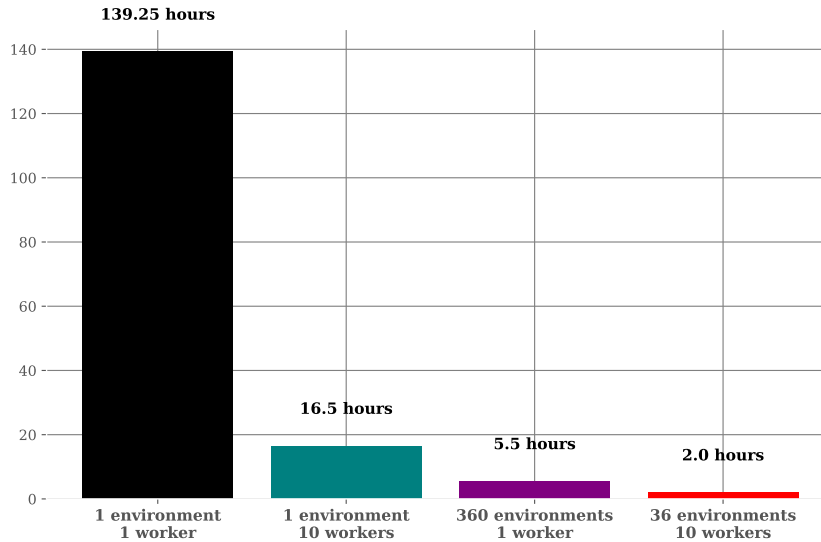
A.2 Training times

Figure 8: Training time of the V2 model under different resource configurations. Parallelizing experience collection in PPO and vectorizing power computations in the FLORIS simulator leads to a 70 speedup for training. Each model is trained on a NVIDIA Grace central processing unit (CPU) and NVIDIA GH200 Hopper graphics processing unit (GPU).

A.3 Hyperparameters

Numerical simulations are conducted with FLORIS [15], a steady-state, low-fidelity simulator developed by NREL. The default Gaussian-curl hybrid model [20] provided by FLORIS is used. The machines are International Energy Agency (IEA) 15 MWs wind turbines [21]. For the PPO, hyperparameters are listed in Table 1.

Table 1: PPO hyperparameters used for each model. Only the learning rates and gradient clipping parameters differ. At each training step, the current learning rate is sampled using linear interpolation between the initial and final values.

Hyperparameter	Model V0	Model V1	Model V2
Training steps	150	150	150
Discount factor	0.1	0.1	0.1
Learning rate (first)	1e-4	1e-5	1e-5
Learning rate (last)	1e-6	1e-7	1e-7
Gradient clipping	10	None	None
GAE λ parameter	0.95	0.95	0.95
Entropy coefficient	0.05	0.05	0.05
Clip parameter (actor)	0.01	0.01	0.01
Clip parameter (critic)	10	10	10
Value loss coefficient	0.1	0.1	0.1
Number of epochs	11	11	11
Train batch size	6480	6480	6480
Mini batch size	360	360	360

A.4 Ablation study

We conduct an ablation study to assess the influence of the exponential and invalid terms within our reward function. Figure 9a displays the performance of the default V2 model. In Figure 9b, we evaluate the V2 model trained with $p = 0$, effectively removing the exponential term's impact. And Figure 9c shows results when the r_{t+1}^{invalid} term is eliminated by setting $w_0 = 0$. The study indicates that the exponential term is crucial for consistent optimization across all wind conditions, as its absence leads to suboptimal performance in scenarios with small wake losses. Furthermore, the invalid term contributes to overall performance, with its removal resulting in a slight global degradation.

(a) Default. (b) Without $\exp(-p\bar{\mathcal{L}}_t)$. (c) Without r_{t+1}^{invalid}.
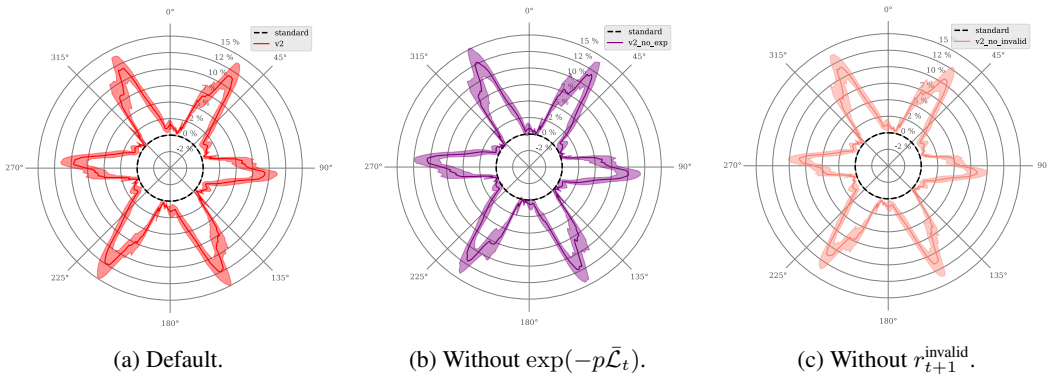
Figure 9: Ablation study for the V2 model. Without the exponential term, i.e., with $p = 0$, performance is degraded for small wake losses conditions. And without the invalid term, i.e., with $w_0 = 0$, performance is globally decreased.

A.5 Proximal policy optimization

PPO [18] is a model-free, on-policy deep RL algorithm based on policy gradients. It improves stability and sample efficiency over standard policy gradient methods by introducing a clipped surrogate objective that constrains policy updates. In this work, we have developed a custom actor-critic implementation of PPO, drawing inspiration from Stable Baselines 3 [22] and RLlib [23, 24]. Our implementation is fully vectorized using NumPy and PyTorch matrix operations [25], and the collection of trajectories is fully parallelized via the Ray library [26] to enable efficient large-scale training. It supports both continuous and discrete action spaces. Let π_θ and V_θ be a policy and value function, respectively, with shared parameters θ. The pseudocode of our PPO training loop is given in Algorithm 1. It consists of two distinct phases: the collection of trajectories and the update of the parameters.

During training, trajectories are collected by executing the current policy π_θ and the current value V_θ in the environment. These trajectories consist of tuples $(s_t, a_t, r_{t+1}, s_{t+1})$ over multiple time steps. At the end of a trajectory, the value function is used to bootstrap the final return only when the episode is truncated (e.g., due to time limits). In that case, the final estimated value $V_\theta(s_T)$ is used as a proxy for future rewards: $r_T \leftarrow r_T + \gamma V_\theta(s_T)$. This ensures consistent return estimation across both terminated and truncated episodes. At the end of an episode, GAE [19] is used to compute the value targets $\hat{V}_t$ and the advantages $\hat{A}_t$. This method provides a biased but low-variance estimator of the advantages. It relied on the temporal difference (TD) residual $\delta_t = r_t + \gamma V_\theta(s_{t+1}) - V_\theta(s_t)$. The advantages are computed as an exponentially-weighted sum $\hat{A}_t = \sum_{k=0}^{T-t-1}(\gamma\lambda)^k \delta_{t+k}$ with λ the GAE parameter. And the value targets are estimated as $\hat{V}_t = \hat{A}_t + V_\theta(s_t)$.

After collecting multiple trajectories, the training process enters an optimization phase consisting of several epochs. During each epoch, the full batch of collected data is shuffled and divided into minibatches. Each minibatch is then used to compute the loss and update the parameters of both the policy and value networks via stochastic gradient descent (SGD). The total loss is defined as a weighted sum of four components: $\mathcal{L} = c_0 \mathcal{L}_{\text{actor}} + c_1 \mathcal{L}_{\text{critic}} + c_2 \mathcal{L}_{\text{entropy}}$, where c_0, c_1, and c_2 are scalar hyperparameters controlling the relative contribution of each term. The probability ratio between the current and old policy is given by $r_t(\theta) = \pi_\theta(a_t|s_t)/\pi_{\theta_{\text{old}}}(a_t|s_t)$ and is central to the actor loss. The four loss terms are defined as follows.

- **Actor loss:** $\mathcal{L}_{\text{actor}} = -\min\left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\varepsilon, 1+\varepsilon)\hat{A}_t\right)$, where $\hat{A}_t$ is the advantage estimate and $\varepsilon \in (0,1)$ is a trust-region hyperparameter. The clipping prevents large policy updates that could destabilize training.

- **Critic loss:** $\mathcal{L}_{\text{critic}} = \text{clip}\left((V_\theta(s_t) - \hat{V}_t)^2, 0, \texttt{vf\_clip\_param}\right)$, where $\hat{V}_t$ is the target return and $\texttt{vf\_clip\_param}$ controls the maximum contribution of the value error to the total loss, ensuring stability in value updates.

- **Entropy loss:** $\mathcal{L}_{\text{entropy}} = -\mathcal{H}(\pi_\theta(s_t))$, where $\mathcal{H}$ denotes the Shannon entropy of the policy. This term encourages exploration by penalizing low-entropy (overly deterministic) policies.

Algorithm 1 Pseudocode of PPO training loop.

Input: Initial policy π_θ and value V_θ
Input: PPO hyper-parameters
1: **for** $\texttt{iter} = 0$ to $\texttt{max\_iters}$ **do**
2:     Collect trajectories using π_θ
3:     Compute value targets $\hat{V}_t$
4:     Compute advantages $\hat{A}_t$
5:     Fix parameters $\theta_{\text{old}} \longleftarrow \theta$
6:     **for** $\texttt{epoch} = 0$ to $\texttt{nb\_epochs}$ **do**
7:         **for** minibatch in batches **do**
8:             Compute ratio $r_t = \pi_\theta(a_t|s_t)/\pi_{\theta_{\text{old}}}(a_t|s_t)$
9:             Compute loss $\mathcal{L} = c_0 \mathcal{L}_{\text{actor}} + c_1 \mathcal{L}_{\text{critic}} + c_2 \mathcal{L}_{\text{entropy}}$
10:             Update parameters $\theta \longleftarrow \theta - \alpha \nabla_\theta \mathcal{L}$
11:         **end for**
12:     **end for**
13: **end for**

The policy outputs $p_0 = (p_0^i)_{i \in \{0,...,N-1\}}$ and $p_1 = (p_1^i)_{i \in \{0,...,N-1\}}$, which parameterize a set of independent von Mises distributions, one per turbine. For each turbine i, the location parameter is defined as $\mu^i = \pi \cdot \tanh(p_0^i)$ and the concentration parameter as $\kappa^i = \text{softplus}(p_1^i)$. During training, actions are sampled independently for each turbine as $a_t^i \sim \mathcal{V}(\mu^i, \kappa^i)$, while during evaluation, actions are set deterministically to the mode, $a_t^i = \mu^i$. The use of von Mises distributions ensures that the yaw angles are naturally constrained within the circular interval $[-\pi, \pi]$.

As the von Mises distribution is rarely used as a policy output in continuous RL, we provide additional details below. Specifically, we present the expressions for its probability density function and entropy. The probability density function is defined as $f(x \mid \mu, \kappa) = \frac{\exp(\kappa \cos(x-\mu))}{2\pi I_0(\kappa)}$, with μ the measure of location, κ the measure of concentration and $I_0(k)$ the modified Bessel function of the first kind of order 0. The Bessel's integral $I_n(\kappa)$ can be written as $I_n(\kappa) = \frac{1}{\pi} \int_0^\pi \cos(nx) \exp(\kappa \cos(x)) \, dx$. The entropy $\mathcal{H}(f)$ is computed as follows.

$$\mathcal{H}(f) = -\int_{-\pi}^{\pi} f(x) \log(f(x)) \, dx \tag{4}$$

$$= -\int_{-\pi}^{\pi} f(x)(\kappa \cos(x-\mu) - \log(2\pi I_0(\kappa))) \, dx \tag{5}$$

$$= -\int_{-\pi}^{\pi} f(x)\kappa \cos(x-\mu) \, dx + \int_{-\pi}^{\pi} f(x) \log(2\pi I_0(\kappa)) \, dx \tag{6}$$

$$= \frac{-\kappa}{2\pi I_0(\kappa)} \int_{-\pi}^{\pi} \cos(x-\mu) \exp(\kappa \cos(x-\mu)) \, dx + \log(2\pi I_0(\kappa)) \int_{-\pi}^{\pi} f(x) \, dx \tag{7}$$

$$= \frac{-\kappa}{2\pi I_0(\kappa)} 2\pi I_1(\kappa) + \log(2\pi I_0(\kappa)) \tag{8}$$

$$= -\kappa \frac{I_1(\kappa)}{I_0(\kappa)} + \log(2\pi I_0(\kappa)). \tag{9}$$