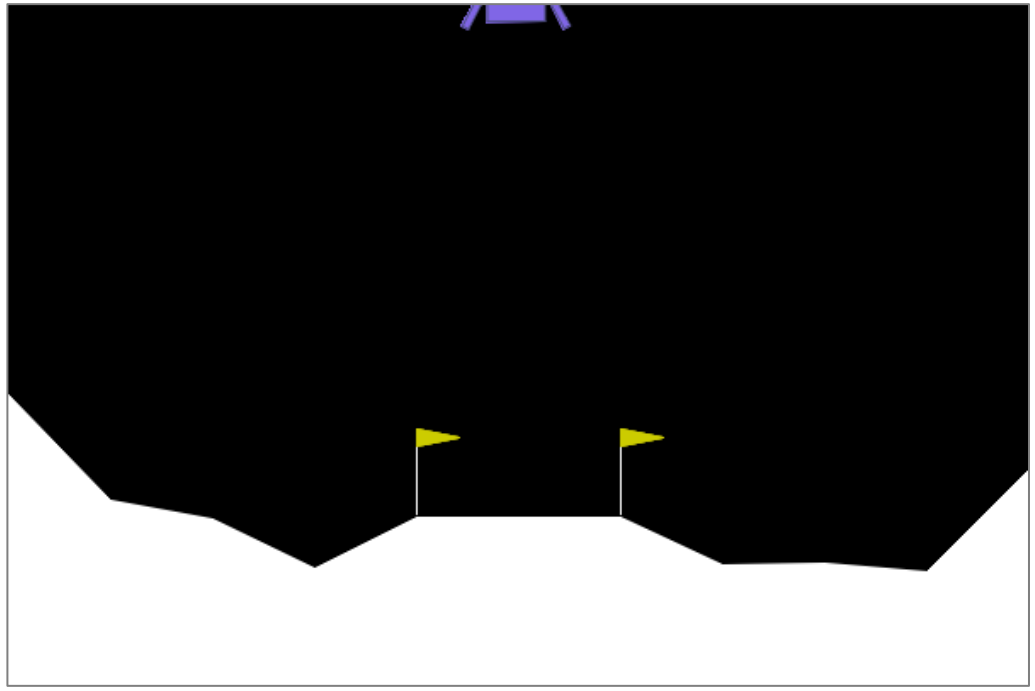
Reinforcement Learning

Final Assignment - Thomas Vroom - 2025

Introduction - Environment

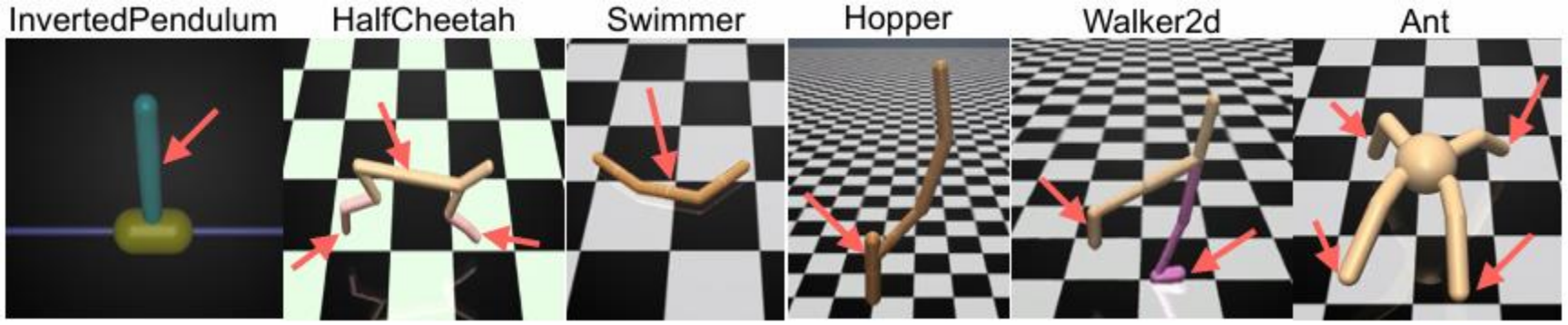
- LunarLander-v3 environment
  - More challenging than the now deprecated v2
- Challenging control task
- State space: 8-dimensional vector
- Action space: 4 actions
  - (Noop, Fire main engine, Fire left engine, Fire right engine)



Source: lunar_lander.gif (600×400)

Introduction - RARL

- Robust Adversarial Reinforcement Learning (RARL)
- Train an adversary agent that perturbs the environment in a way to maximally hinder the protagonist agent (i.e. turn the environment into a zero-sum game)
- Protagonist learns a more robust policy!
- Curriculum learning: environment becomes harder over time



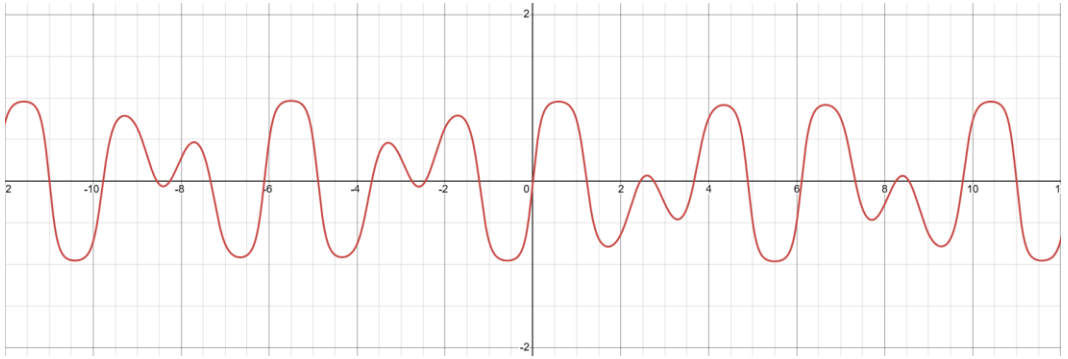
Source: [1]

Introduction - RARL

- How could you perturb the lunar lander environment?

- The lunar lander environment has an optional parameter: *enable_wind*

$$\tanh\left(\sin\left(2kx\right)+\sin\left(\pi kx\right)\right)$$



- Further scaled by the *wind_power* and *turbulence_power* parameters

Research Questions

1. Can a DRL agent successfully land a lander on the moon?

2. What difference does adding wind during training make on the “robustness” of the final policy?

3. What difference does RARL make on the “robustness” of the final policy?

DRL – Double Deep Q-Networks (DDQN)

- 2 Networks:
  - Policy Network: 8 → 128 → 256 → 4
  - Target Network: same as policy network, used to predict Q-values of next state
- Weights of policy network are periodically copied to the target network
- Replay buffer remembers visited state-action pairs and their rewards
- Epsilon-greedy action selection with epsilon decay
- Based on [5]

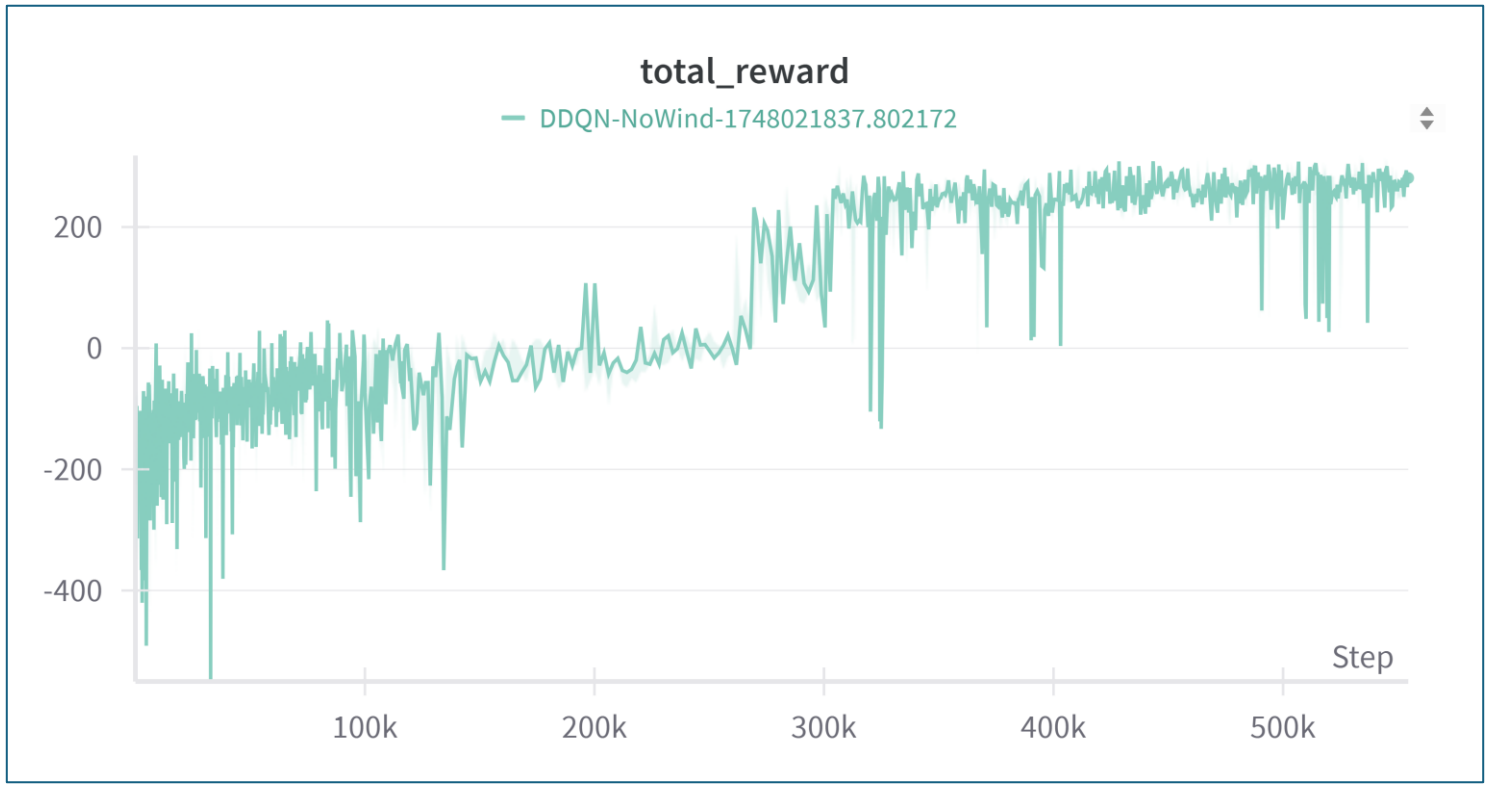
DRL – Double Deep Q-Networks (DDQN)

- Hyperparameters:

Parameter	Value
Total number of episodes	2000
Size of the replay buffer	100 000
Batch size	64
Steps to replace the target network	500
Discount factor	0.99
Learning rate	0.0001
Weight decay	0.01
Minimum epsilon	0.01
Epsilon decay per step	0.000005

Source: https://medium.com/@coldstart_coder/dqn-algorithm-training-an-ai-to-land-on-the-moon-1a1307748ed9

DRL – Double Deep Q-Networks (DDQN)



- Reward of 200 counts as a “valid solution”
- During training, all environments are limited to a max. step count of 1000. This is to try applying some time pressure to the agent, hopefully preventing it from getting stuck in the “hovering but not crashing” local optima!

DRL – Proximal Policy Optimization (PPO)

- 2 Networks:
  - Actor (policy): 8 → 128 → 128 → 4
  - Critic (value function): 8 → 128 → 128 → 1

- Policy gradient method that tries to:
  - Take small steps in policy-space (~on policy)
  - Take large steps in parameter space (optimize learning)

- I used the implementation found in the CleanRL repository
  - Based on the paper: The 37 Implementation Details of Proximal Policy Optimization

DRL – Proximal Policy Optimization (PPO)

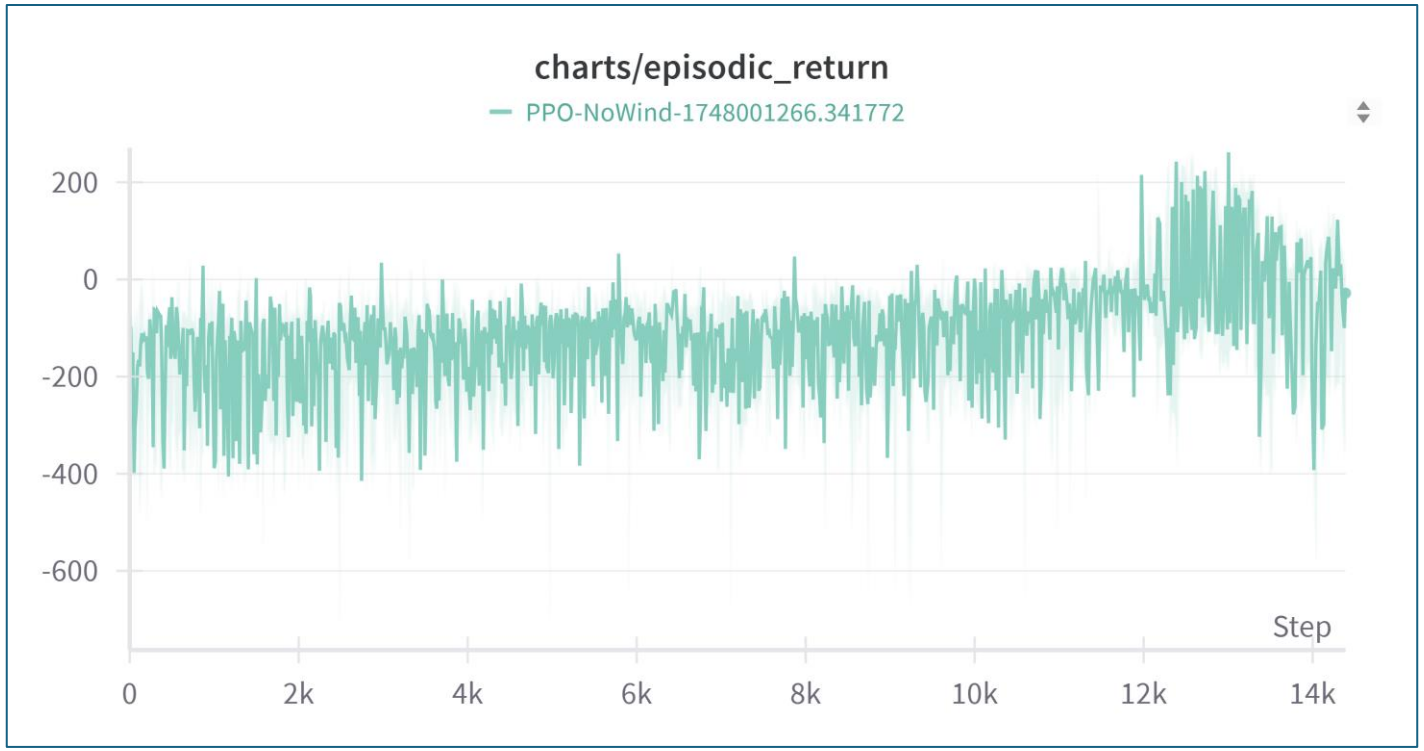
- Initial Hyperparameters:
  - The actor and critic also have hidden layers of size [64, 64] at this point

Parameter	Value
Parallel environments	8
Total number of steps	2 000 000
Steps per rollout	2048
Number of minibatches	4
Policy update epochs	4
Discount factor	0.99
Learning rate	0.0002
Decay learning rate	False

Parameter	Value
GAE lambda	0.95
Surrogate clipping coef.	0.2
Advantage norm.	True
Clip loss value function	True
Entropy coefficient	0.01
Value function coef.	0.5
Max. gradient norm.	0.5
Target KL threshold	null

DRL – Proximal Policy Optimization (PPO)

- Results:

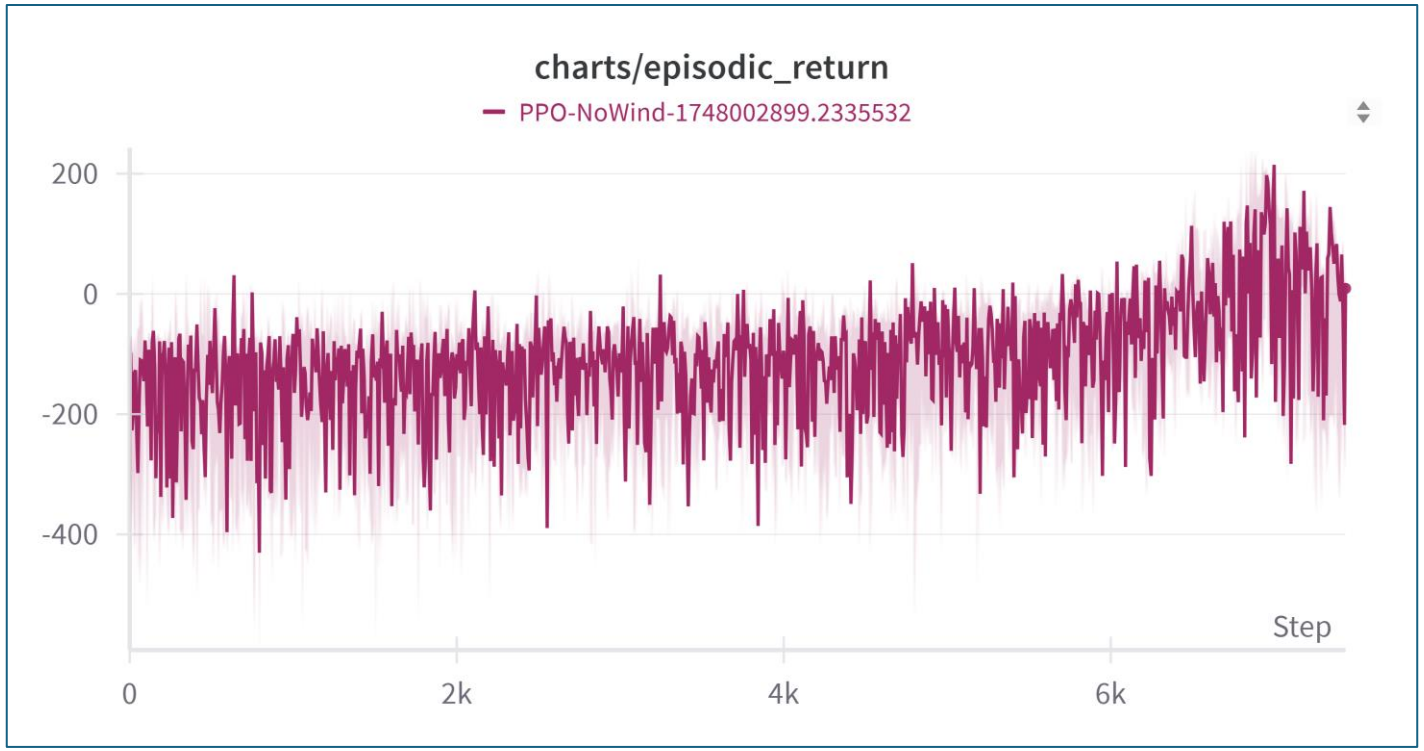


- No real signs of learning, falls short of the 200 threshold…

First change: actor-critic hidden layers from [64, 64] → [128, 128]

DRL – Proximal Policy Optimization (PPO)

- Results:



- Shows signs of learning, but high variance and still falls short…

DRL – Proximal Policy Optimization (PPO)

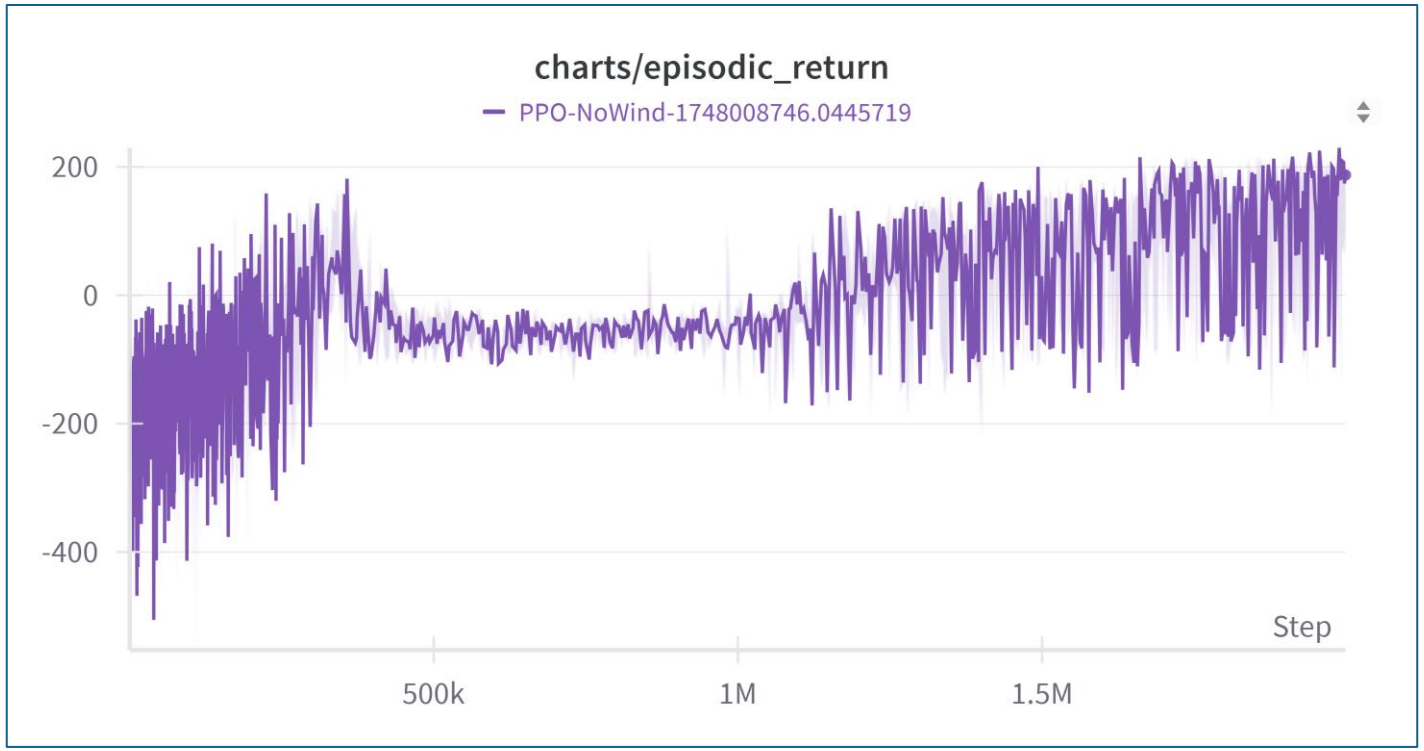
- Since I don’t have time for a full hyperparameter search, I ask ChatGPT to give me suggestions on the most important ones:

Parameter	Value
Parallel environments	8
Total number of steps	2 000 000
Steps per rollout	2048
Number of minibatches	**4 → 16**
Policy update epochs	**4 → 8**
Discount factor	0.99
Learning rate	**0.0002 → 0.0003**
Decay learning rate	**False → True**

Parameter	Value
GAE lambda	**0.95 → 0.9**
Surrogate clipping coef.	0.2
Advantage norm.	True
Clip loss value function	True
Entropy coefficient	0.01
Value function coef.	0.5
Max. gradient norm.	0.5
Target KL threshold	null

DRL – Proximal Policy Optimization (PPO)

- Results:



- Touches the threshold of 200, but still high variance and seems to get stuck…

DRL – Proximal Policy Optimization (PPO)

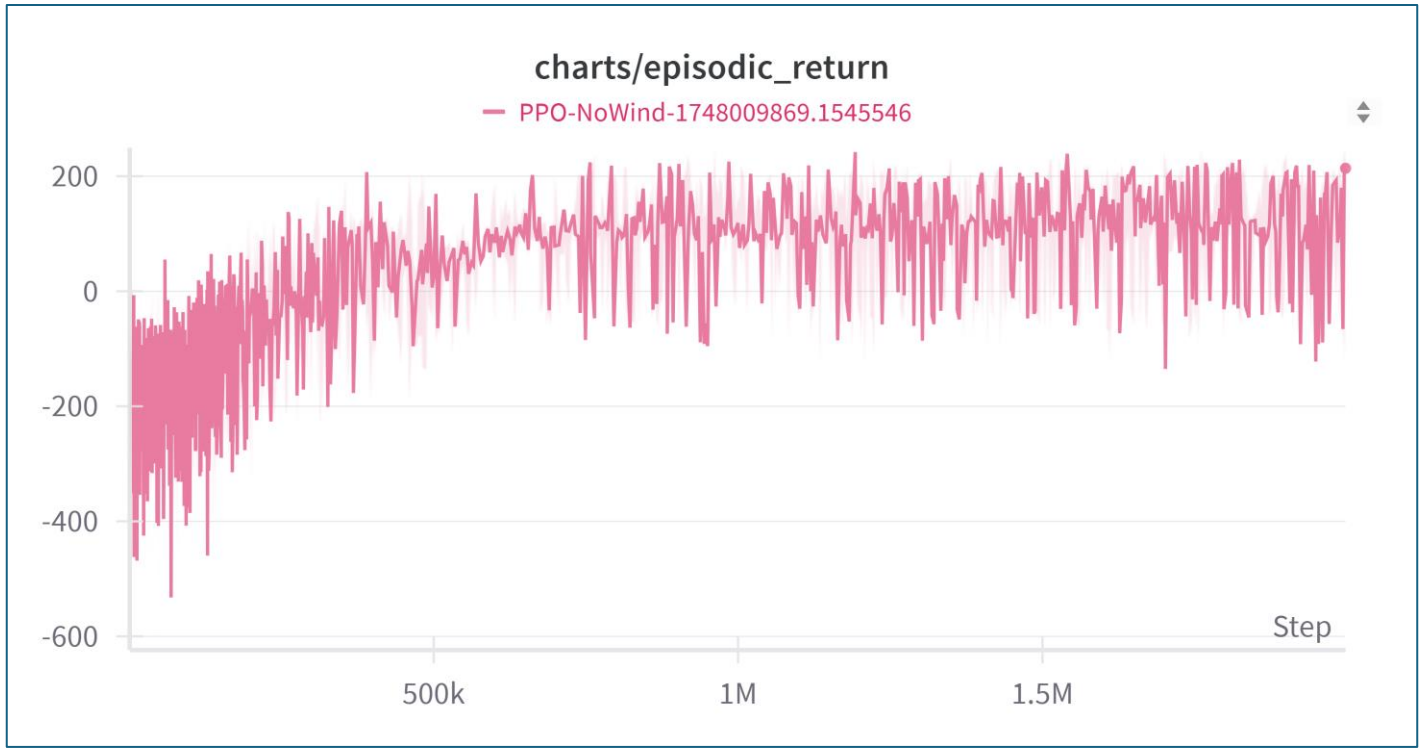
- More back and forth with ChatGPT:

Parameter	Value
Parallel environments	8
Total number of steps	2 000 000
Steps per rollout	2048
Number of minibatches	16
Policy update epochs	8
Discount factor	0.99
Learning rate	0.0003
Decay learning rate	True

Parameter	Value
GAE lambda	0.9
Surrogate clipping coef.	0.2
Advantage norm.	True
Clip loss value function	**True → False**
Entropy coefficient	**0.01 → 0.02**
Value function coef.	0.5
Max. gradient norm.	0.5
Target KL threshold	**Null → 0.01**

DRL – Proximal Policy Optimization (PPO)

- Results:



- Smoother learning curve, still only touches the 200 threshold…

DRL – Proximal Policy Optimization (PPO)

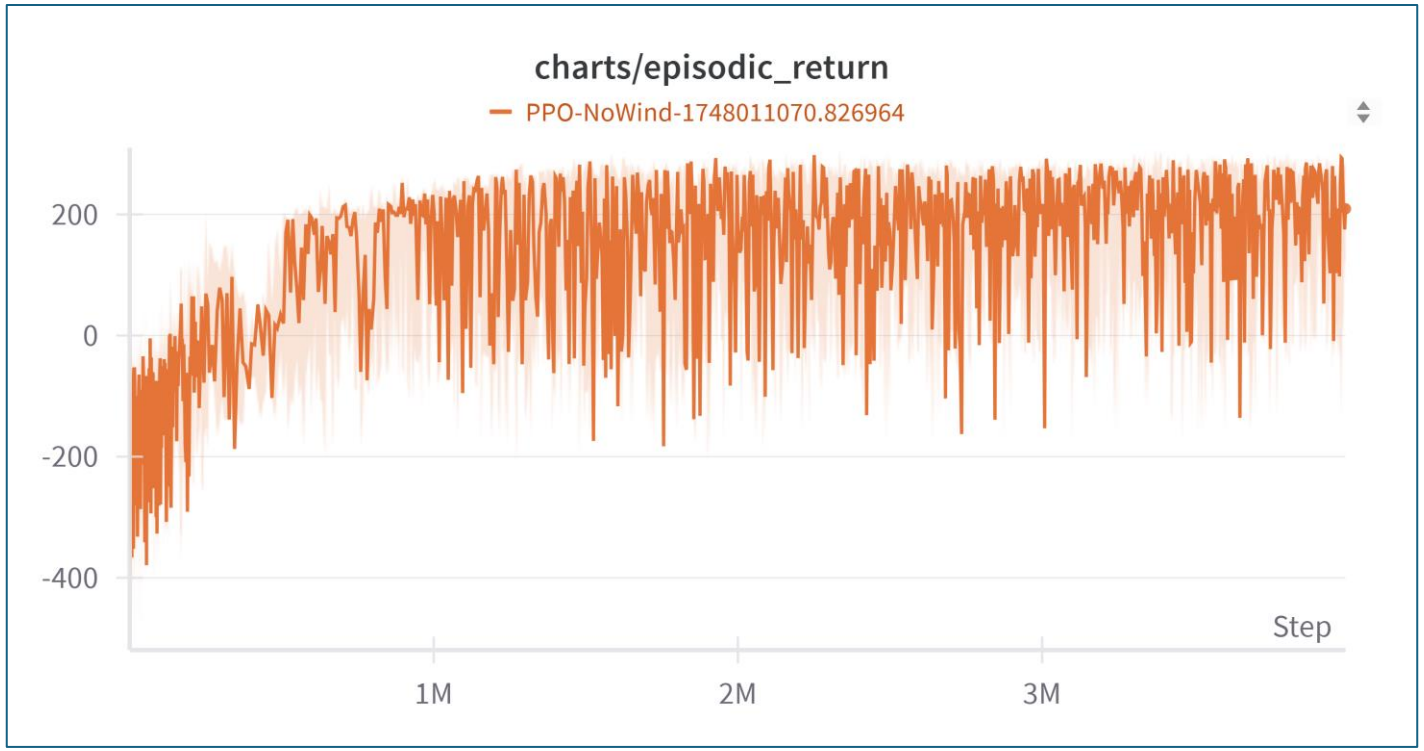
- Final Hyperparameters:

Parameter	Value
Parallel environments	8
Total number of steps	**2 000 000 → 4 000 000**
Steps per rollout	2048
Number of minibatches	16
Policy update epochs	8
Discount factor	0.99
Learning rate	0.0003
Decay learning rate	True

Parameter	Value
GAE lambda	**0.9 → 0.95**
Surrogate clipping coef.	0.2
Advantage norm.	True
Clip loss value function	False
Entropy coefficient	**0.02 → 0.01**
Value function coef.	0.5
Max. gradient norm.	0.5
Target KL threshold	0.01

DRL – Proximal Policy Optimization (PPO)

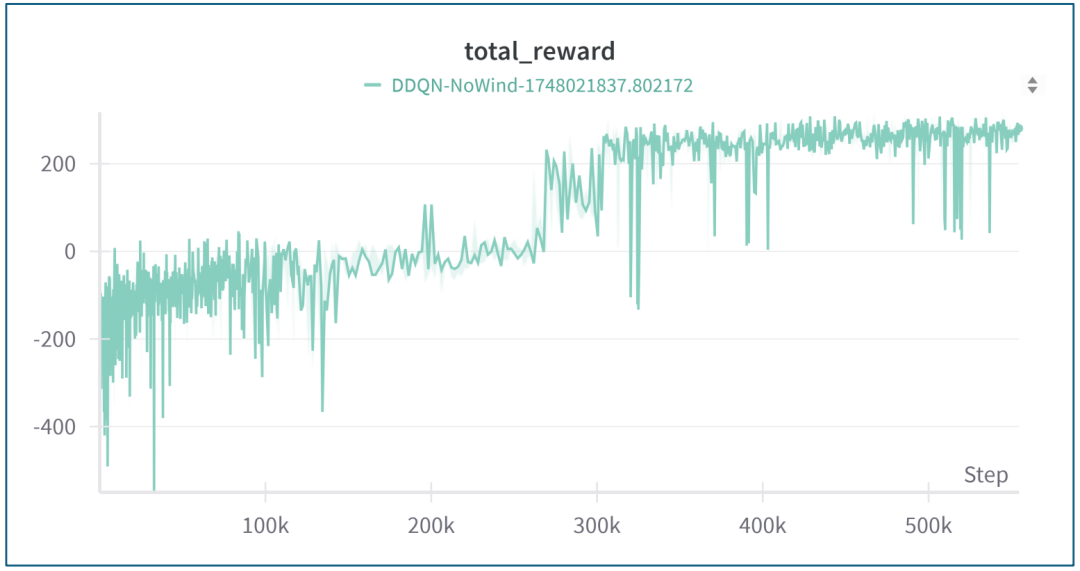
- Results:



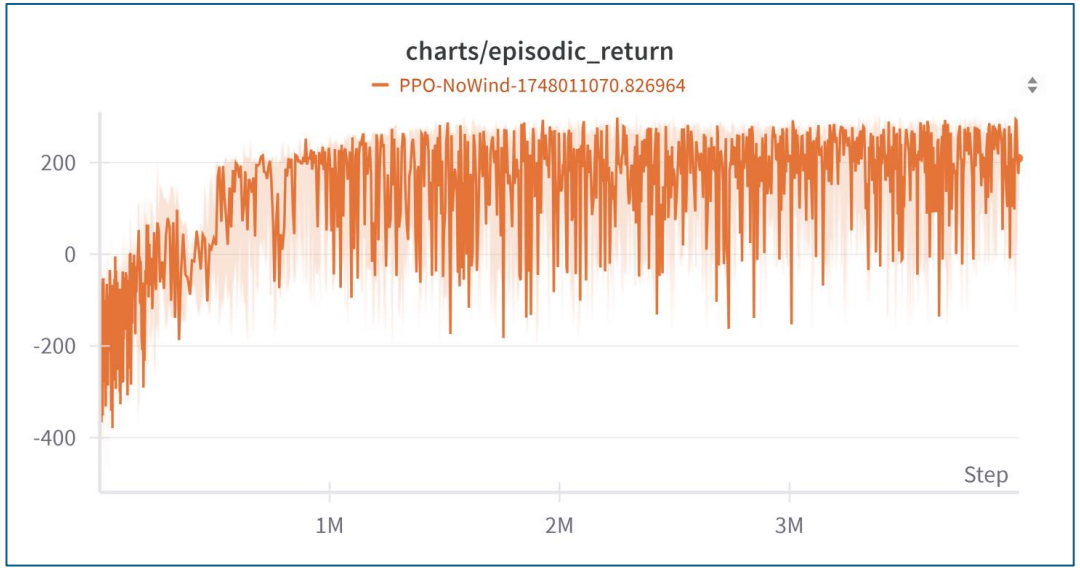
- Smooth learning curve, passes the 200 threshold!

Results – Wind / No Wind

- No wind during training:



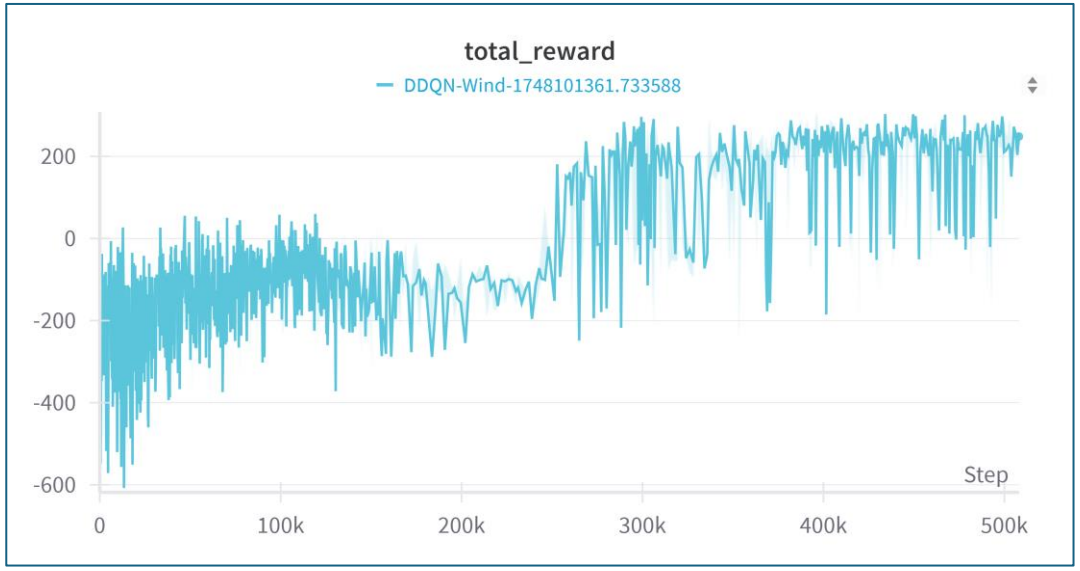
DDQN



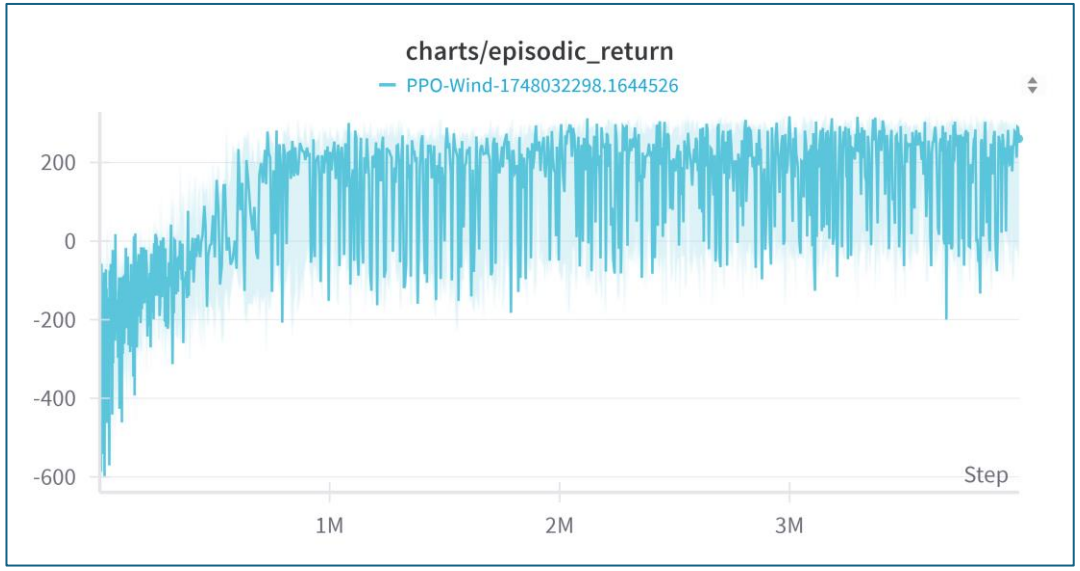
PPO

Results – Wind / No Wind

- Default wind (*wind_power*: 15, *wind_turbulence*: 1.5) during training:



DDQN



PPO

RARL – Modifying Lunar Lander

- Action Space: Discrete(4) → MultiDiscrete(4, 4)

- **Protagonist:** same agent as in the regular lunar lander environment

- **Adversary:** controls the wind experienced by the protagonist:
  1. -1 Linear wind velocity, -1 Angular wind velocity
  2. -1 Linear wind velocity, +1 Angular wind velocity
  3. +1 Linear wind velocity, -1 Angular wind velocity
  4. +1 Linear wind velocity, +1 Angular wind velocity

(Scaled by the *wind_power* and *turbulence_power* parameters)

- $R_{\text{adversary}} = -R_{\text{protagonist}}$

RARL – Modifying Algorithms

- DDQN:
  - Two policy networks (one for each agent)
  - Two target networks (one for each agent)
  - Two replay buffers (one for each agent)
  - Two epsilon values (one for each agent)

- Separate trackers for when to replace the target network

- While training one agent using epsilon-greedy, make the other follow a greedy policy

Algorithm 1 RARL (proposed algorithm)

Input: Environment $\mathcal{E}$; Stochastic policies μ and ν
Initialize: Learnable parameters θ_0^μ for μ and θ_0^ν for ν
for i=1,2,..N_{iter} **do**
  $\theta_i^\mu \leftarrow \theta_{i-1}^\mu$
  for j=1,2,..N_μ **do**
    $\{(s_t^i, a_t^{1i}, a_t^{2i}, r_t^{1i}, r_t^{2i})\} \leftarrow \text{roll}(\mathcal{E}, \mu_{\theta_i^\mu}, \nu_{\theta_{i-1}^\nu}, N_{\text{traj}})$
    $\theta_i^\mu \leftarrow \text{policyOptimizer}(\{(s_t^i, a_t^{1i}, r_t^{1i})\}, \mu, \theta_i^\mu)$
  end for
  $\theta_i^\nu \leftarrow \theta_{i-1}^\nu$
  for j=1,2,..N_ν **do**
    $\{(s_t^i, a_t^{1i}, a_t^{2i}, r_t^{1i}, r_t^{2i})\} \leftarrow \text{roll}(\mathcal{E}, \mu_{\theta_i^\mu}, \nu_{\theta_i^\nu}, N_{\text{traj}})$
    $\theta_i^\nu \leftarrow \text{policyOptimizer}(\{(s_t^i, a_t^{2i}, r_t^{2i})\}, \nu, \theta_i^\nu)$
  end for
end for
Return: $\theta_{N_{\text{iter}}}^\mu, \theta_{N_{\text{iter}}}^\nu$

Source: [1]

RARL – Modifying Algorithms

- PPO:
  - Two actor-critics (one for each agent)
  - Two learning rate (one for each agent)
  - Two action logs (one for each agent)
- Separate trackers for learning rate decay
- Since PPO uses steps instead of episodes, we have to make sure we cut off in steps divisible by the batch size

Algorithm 1 RARL (proposed algorithm)

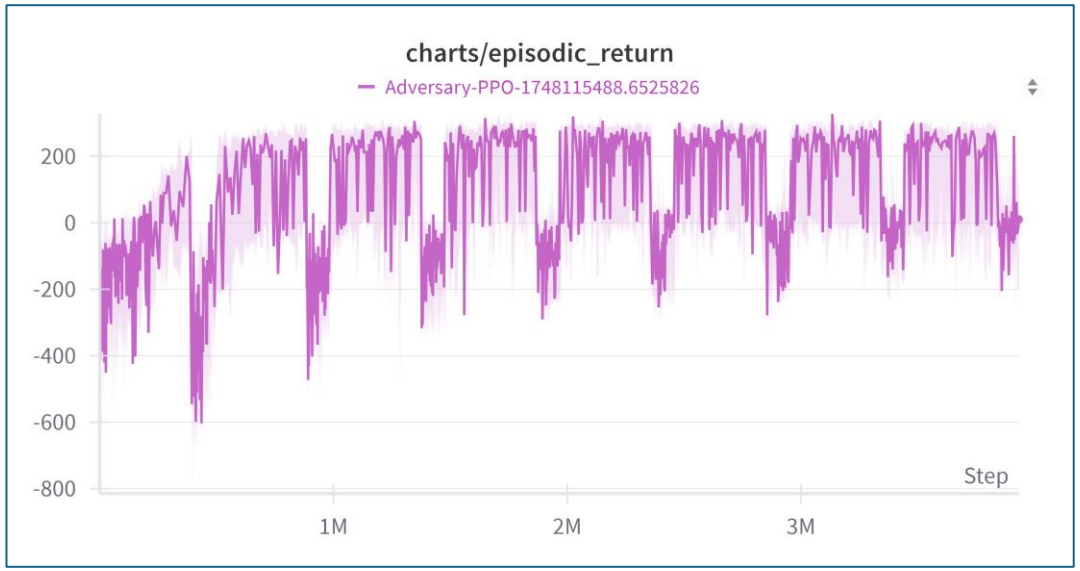
Input: Environment $\mathcal{E}$; Stochastic policies μ and ν
Initialize: Learnable parameters θ_0^μ for μ and θ_0^ν for ν
for i=1,2,..N_{iter} **do**
  $\theta_i^\mu \leftarrow \theta_{i-1}^\mu$
  for j=1,2,..N_μ **do**
    $\{(s_t^i, a_t^{1i}, a_t^{2i}, r_t^{1i}, r_t^{2i})\} \leftarrow \text{roll}(\mathcal{E}, \mu_{\theta_i^\mu}, \nu_{\theta_{i-1}^\nu}, N_{\text{traj}})$
    $\theta_i^\mu \leftarrow \text{policyOptimizer}(\{(s_t^i, a_t^{1i}, r_t^{1i})\}, \mu, \theta_i^\mu)$
  end for
  $\theta_i^\nu \leftarrow \theta_{i-1}^\nu$
  for j=1,2,..N_ν **do**
    $\{(s_t^i, a_t^{1i}, a_t^{2i}, r_t^{1i}, r_t^{2i})\} \leftarrow \text{roll}(\mathcal{E}, \mu_{\theta_i^\mu}, \nu_{\theta_i^\nu}, N_{\text{traj}})$
    $\theta_i^\nu \leftarrow \text{policyOptimizer}(\{(s_t^i, a_t^{2i}, r_t^{2i})\}, \nu, \theta_i^\nu)$
  end for
end for
Return: $\theta_{N_{\text{iter}}}^\mu, \theta_{N_{\text{iter}}}^\nu$

Source: [1]

RARL – Results

Adversarial-DDQN:

- Training cycles: 4
- Protagonist episodes: 600
- Adversary episodes: 200
- Wind power: 10
- Wind turbulence: 1

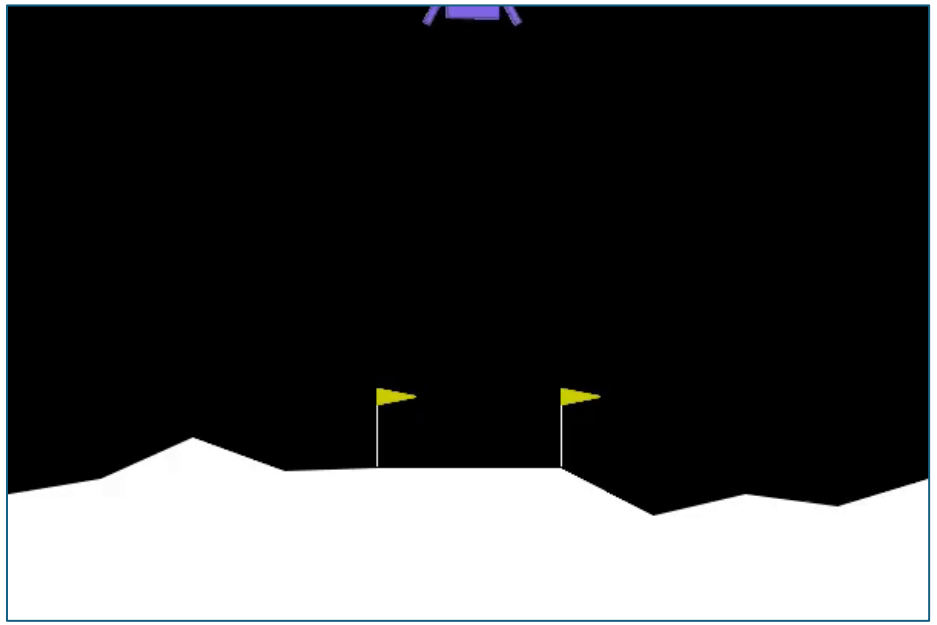


Adversarial-PPO:

- Training cycles: 8
- Protagonist steps: 400 000
- Adversary steps: 100 000
- Wind power: 15
- Wind turbulence: 1.5

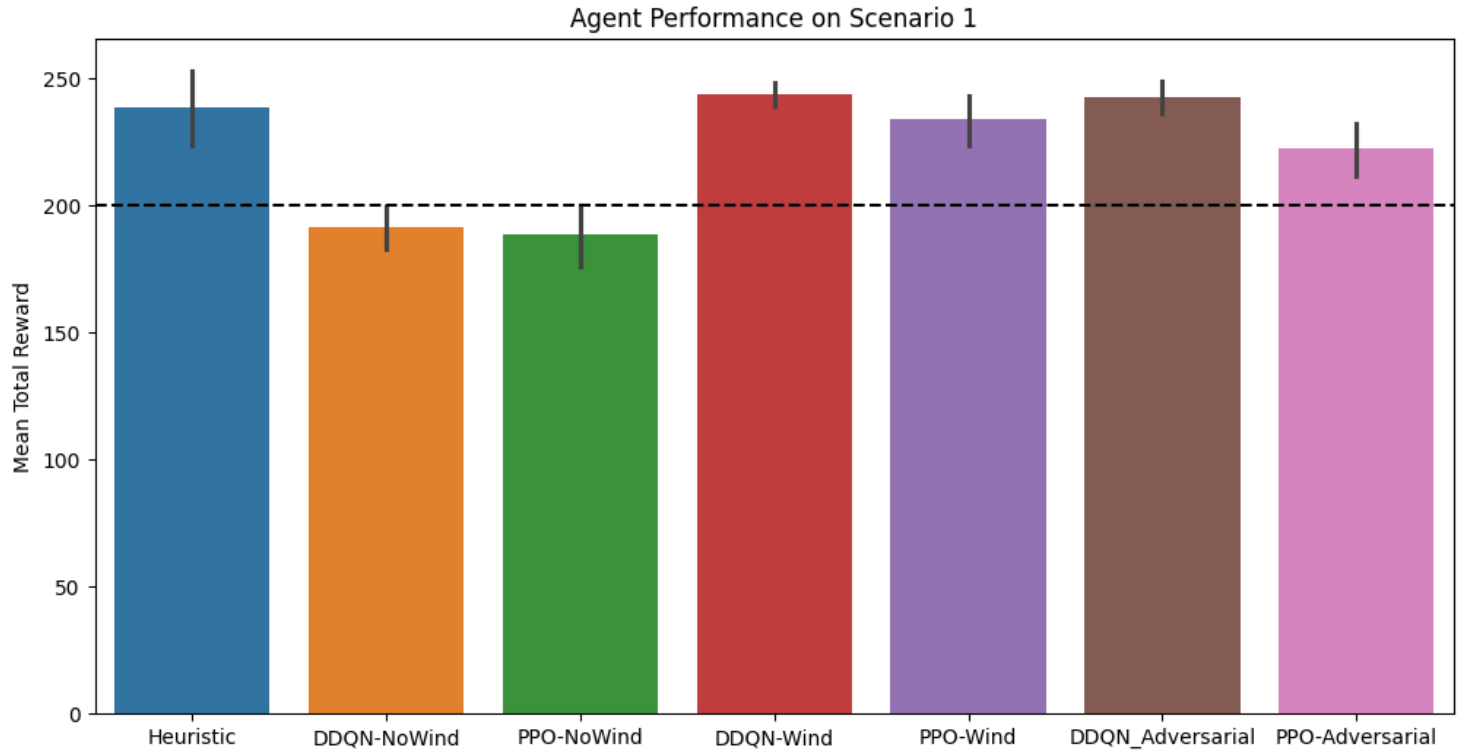
Experiments

- To test the “robustness” of the learned policies, I compared the policies on a series of environments and measured the mean total rewards
- All experiments are averaged over 200 seeds
- Considered policies:
  - Heuristic baseline (based on [7])
  - DDQN-NoWind
  - PPO-NoWind
  - DDQN-Wind
  - PPO-Wind
  - DDQN-Adversarial
  - PPO-Adversarial



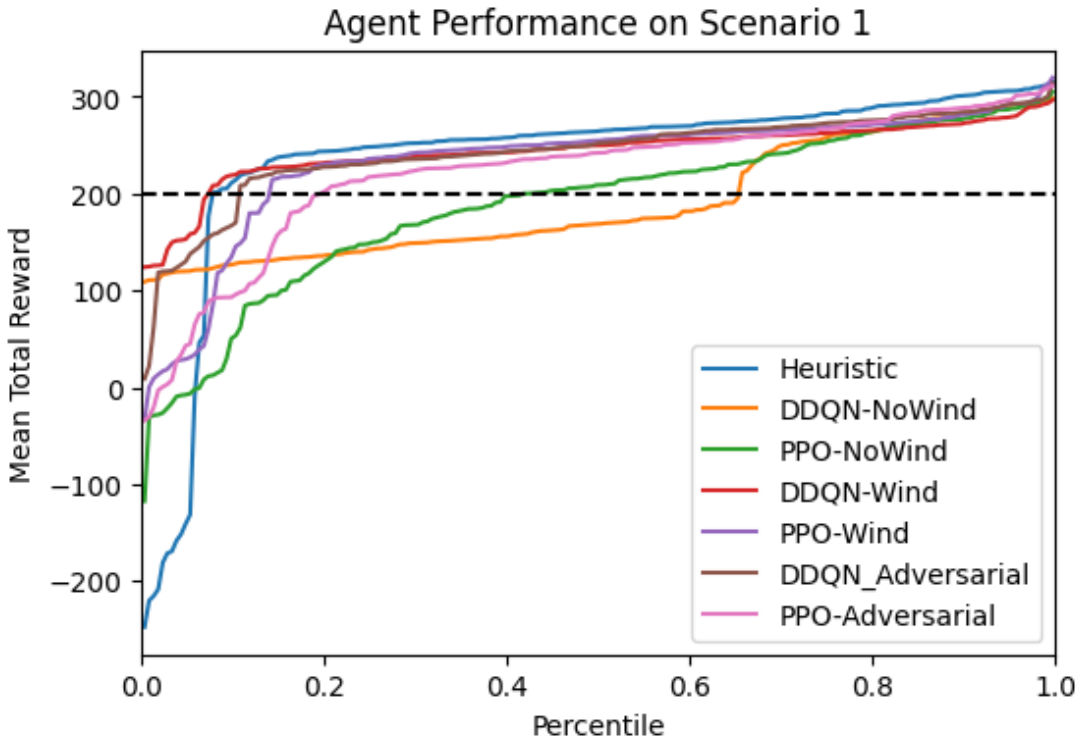
Experiments – Scenario 1

- Environments with no wind:



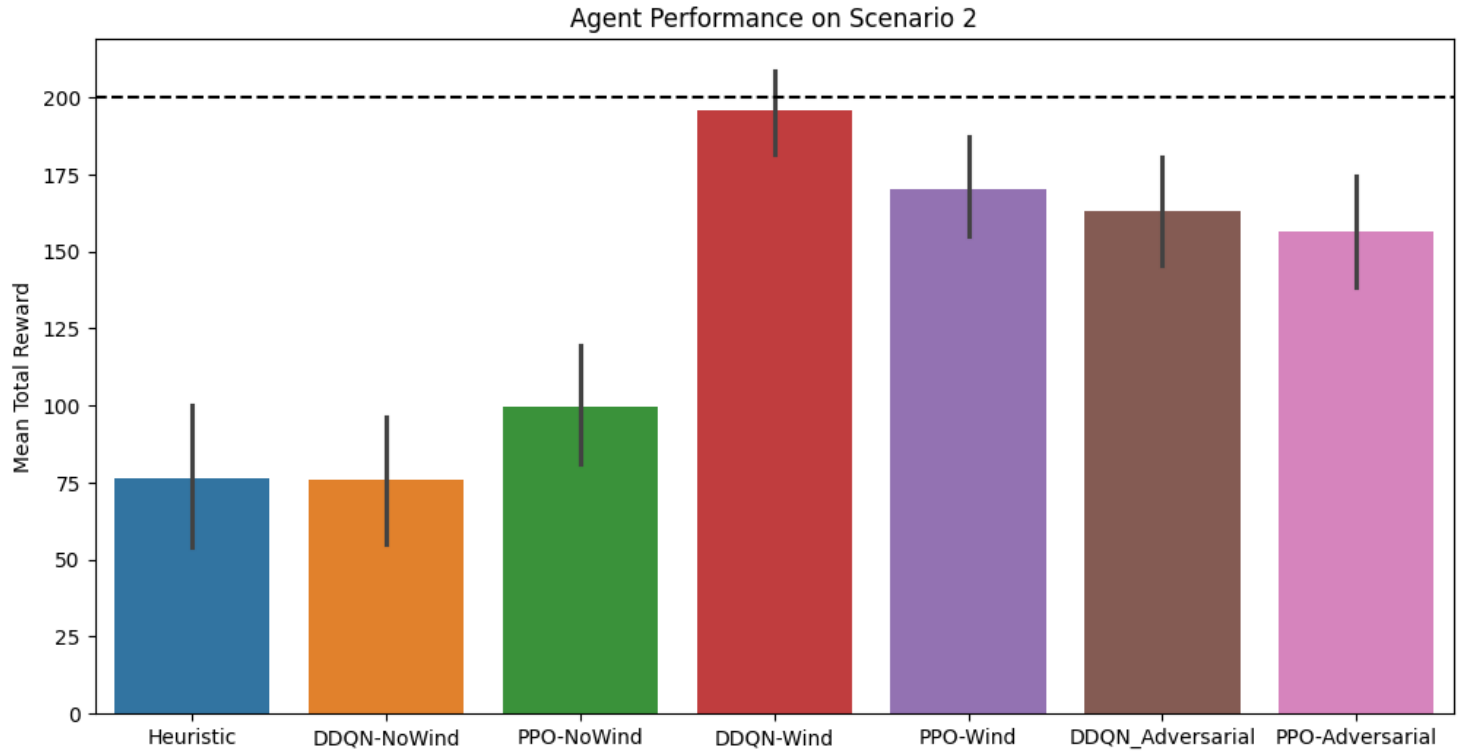
Experiments – Scenario 1

- Environments with no wind:



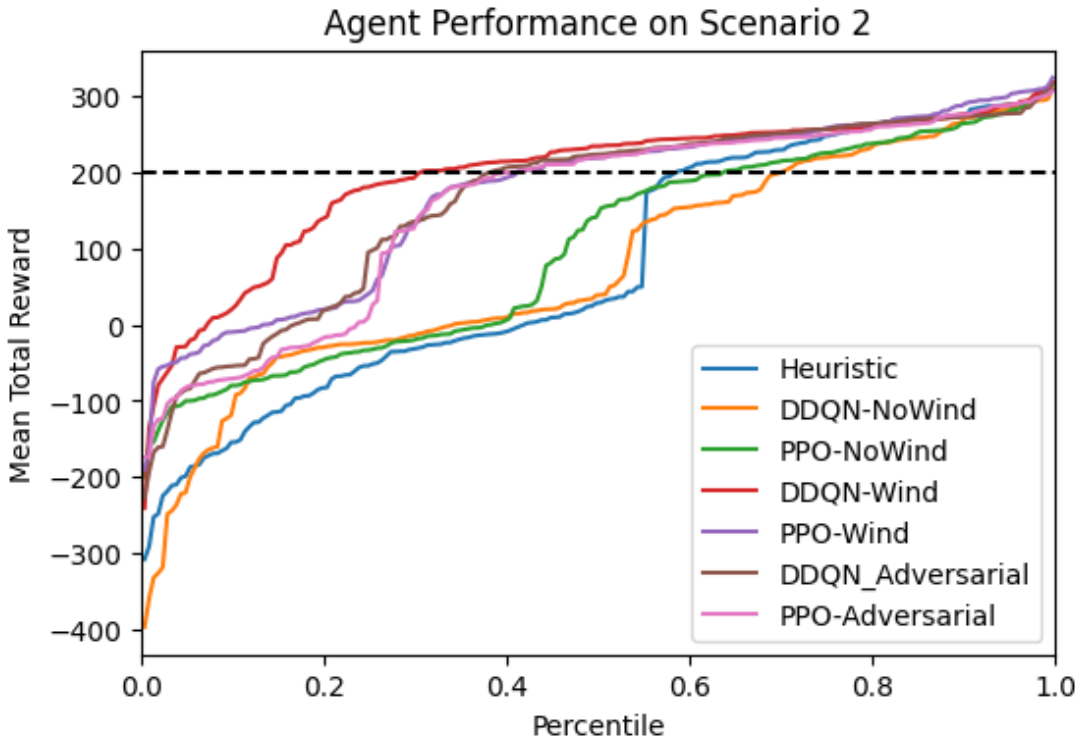
Experiments – Scenario 2

- Environments with default wind:



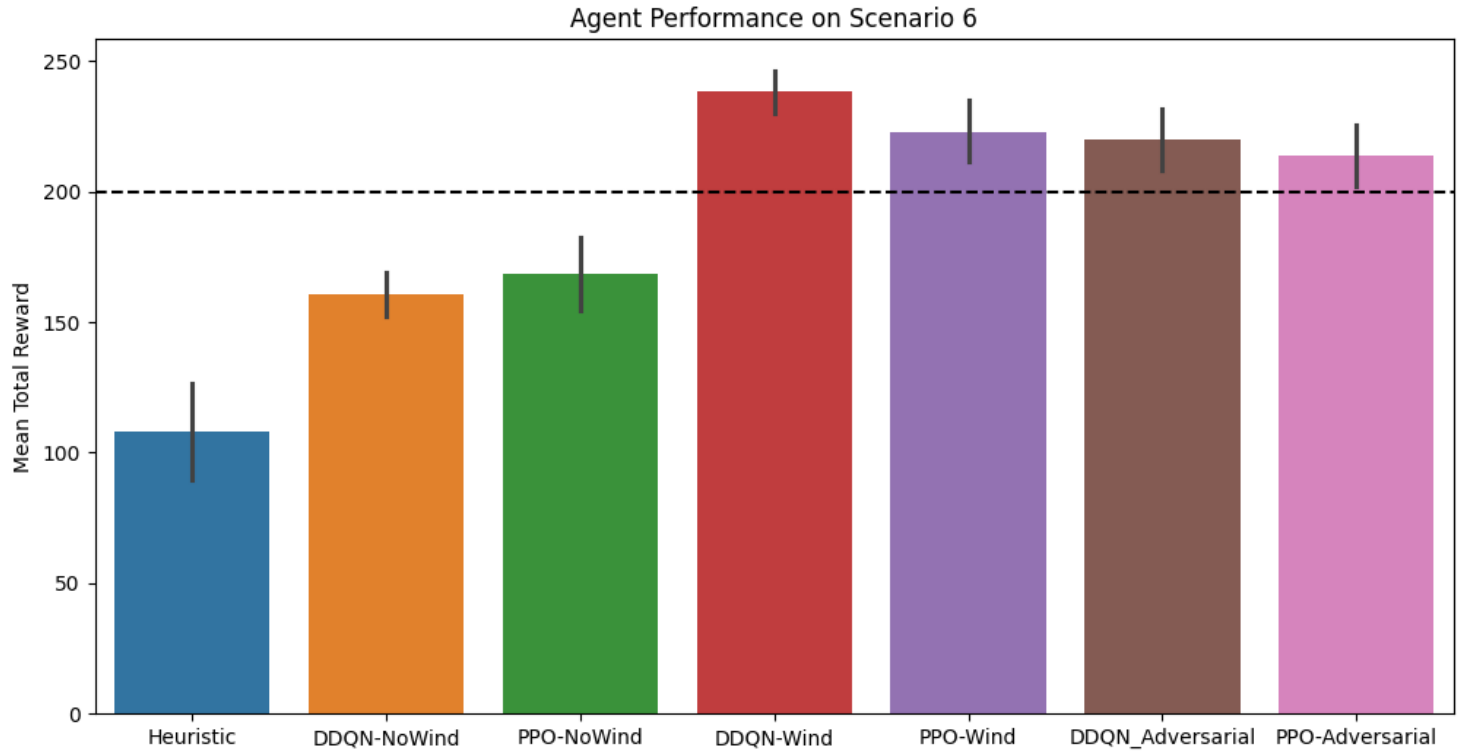
Experiments – Scenario 2

- Environments with default wind:



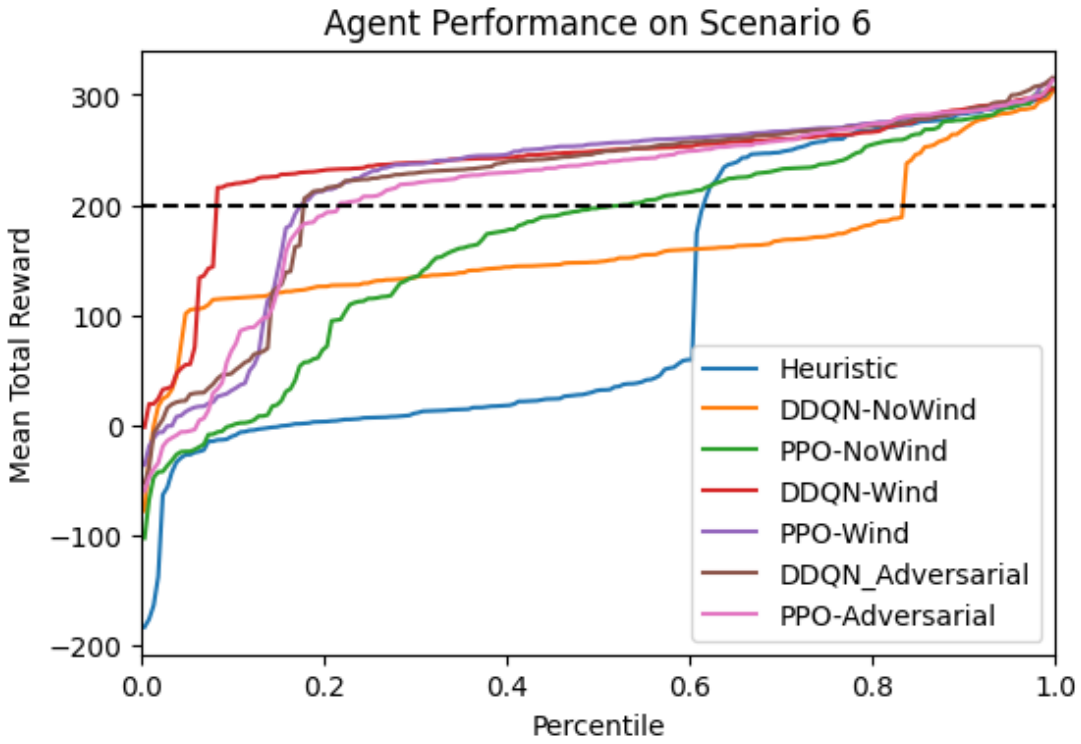
Experiments – Scenario 6

- Environments with increased gravity (g=-12):



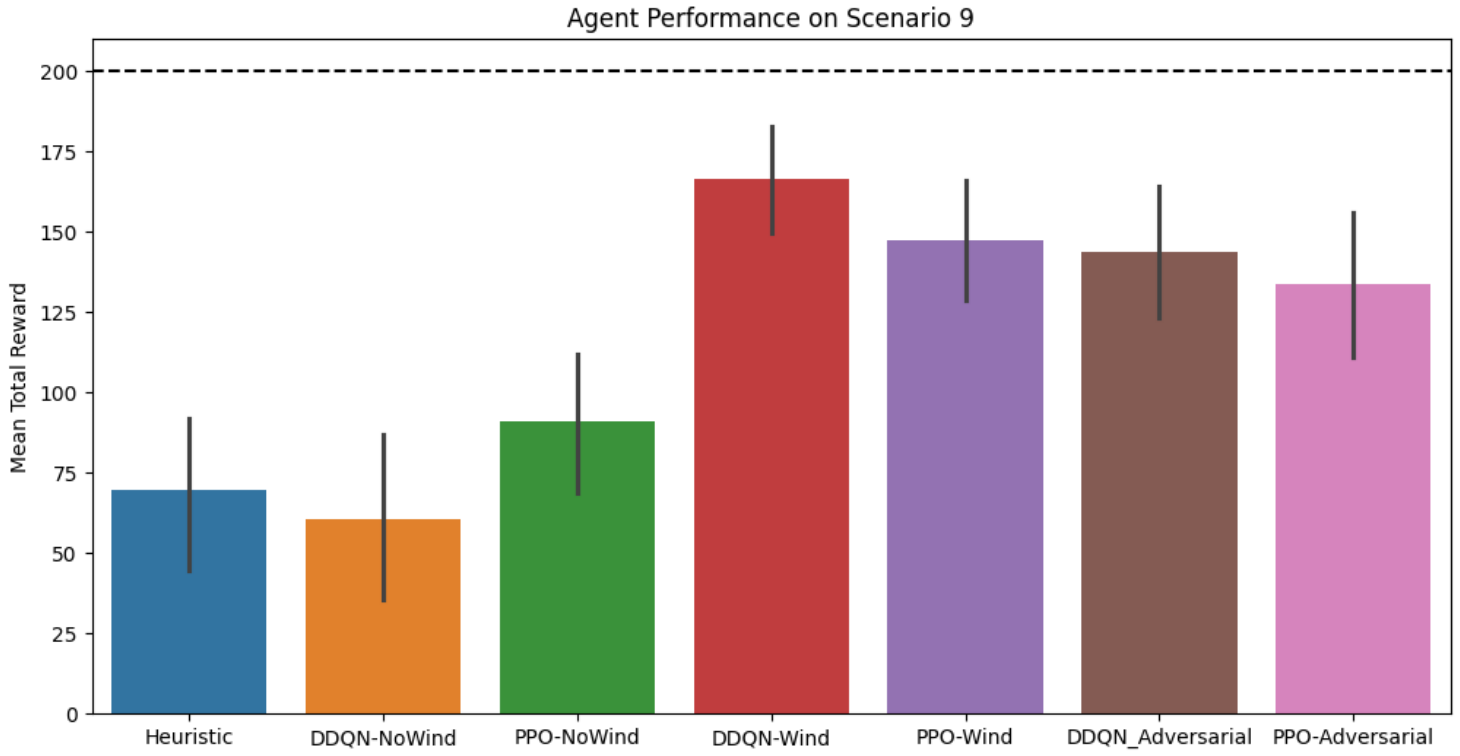
Experiments – Scenario 6

- Environments with increased gravity (*g*=-12):



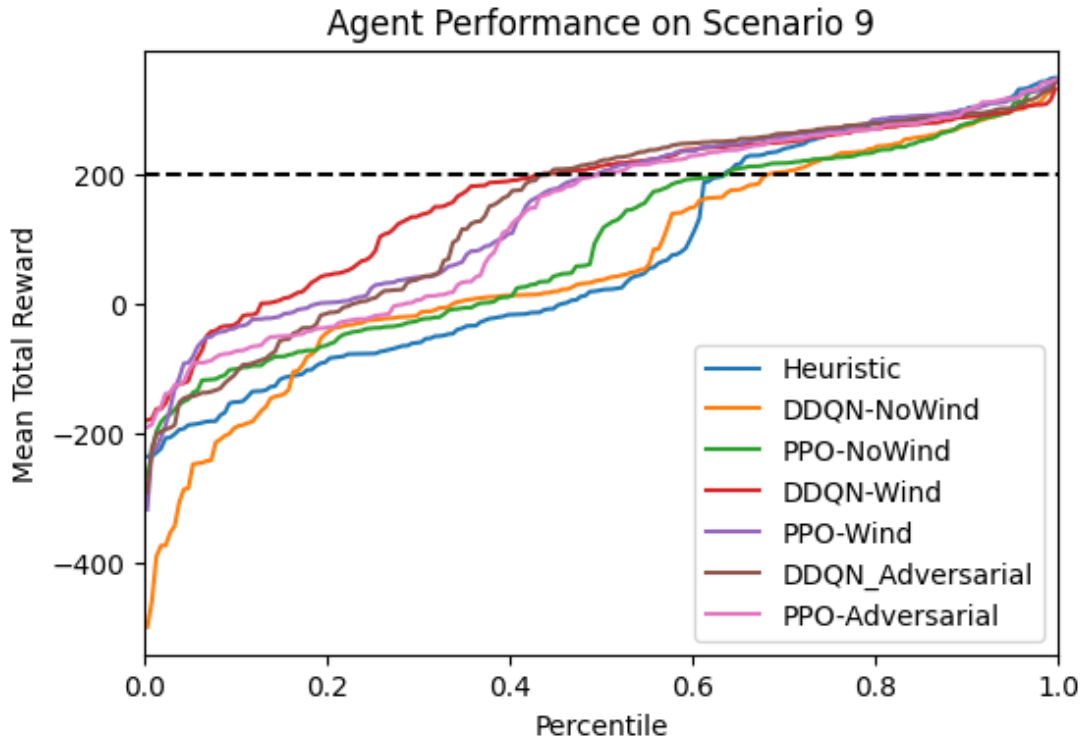
Experiments – Scenario 9

- Environments with default wind + higher initial velocity + engine noise:



Experiments – Scenario 9

- Environments with default wind + higher initial velocity + engine noise:



Conclusion

- DRL agents can successfully “solve” the lunar lander environment
- DDQN is better suited for the lunar lander environment than PPO
- Including wind during the training greatly increases the robustness of the learned policy
- RARL also greatly increases the robustness of the learned policy, but more hyperparameter tuning is likely needed to bring out its full potential

References

1. Pinto, L., Davidson, J., Sukthankar, R., & Gupta, A. (2017, July). Robust adversarial reinforcement learning. In *International conference on machine learning* (pp. 2817-2826). PMLR.
2. Huang, S., Dossa, R. F. J., Ye, C., Braga, J., Chakraborty, D., Mehta, K., & AraÃšjo, J. G. (2022). Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. *Journal of Machine Learning Research*, *23*(274), 1-18.
3. Korkmaz, E. (2023, June). Adversarial robust deep reinforcement learning requires redefining robustness. In *Proceedings of the AAAI Conference on Artificial Intelligence* (Vol. 37, No. 7, pp. 8369-8377).
4. Korkmaz, E., & Brown-Cohen, J. (2023, July). Detecting adversarial directions in deep reinforcement learning to make robust decisions. In *International Conference on Machine Learning* (pp. 17534-17543). PMLR.
5. https://medium.com/@coldstart_coder/dqn-algorithm-training-an-ai-to-land-on-the-moon-1a1307748ed9
6. https://github.com/EzgiKorkmaz/adversarial-reinforcement-learning
7. https://gymnasium.farama.org/environments/box2d/lunar_lander/