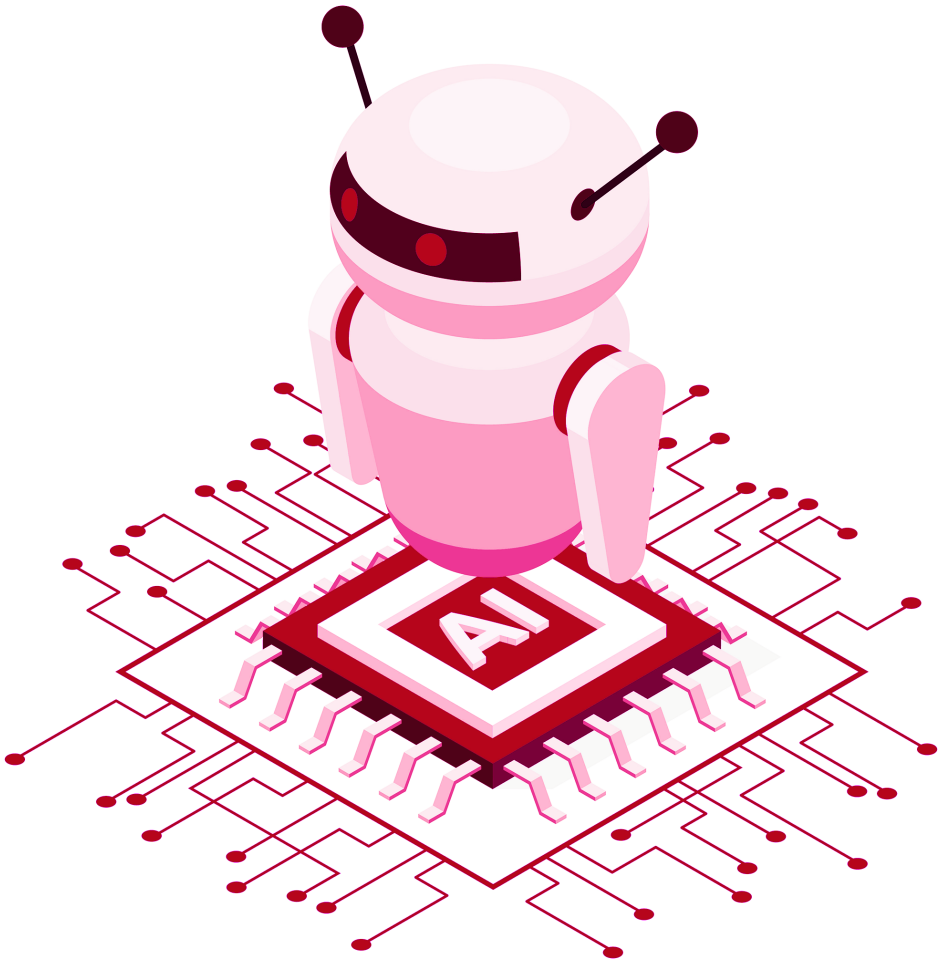

Deep Reinforcement Learning

Professor Mohammad Hossein Rohban

Homework 1:

Introduction to RL

By:

[Danial Parnian]

[401110307]

RIML

Deep RL Course

Spring 2025

Contents

Grading

The grading will be based on the following criteria, with a total of 100 points:

Task	Points
Task 1: Solving Predefined Environments	45
Task 2: Creating Custom Environments	45
Clarity and Quality of Code	5
Clarity and Quality of Report	5
Bonus 1: Writing a wrapper for a known env	10
Bonus 2: Implementing pygame env	20
Bonus 3: Writing your report in Latex	10

Notes:

- Include well-commented code and relevant plots in your notebook.
- Clearly present all comparisons and analyses in your report.
- Ensure reproducibility by specifying all dependencies and configurations.

1 Task 1: Solving Predefined Environments [45-points]

I used PPO and DQN algorithms on Cartpole and FlappyBird environments. The learning curves in Figure 1 show the reward progress over episodes for both algorithms (with default setting for hyperparamaters). As you can see, PPO is more stable and achieves better result. A video of the PPO-model's performance is included with this file. The following plots in Figure 2 show the result of changing learning rate from

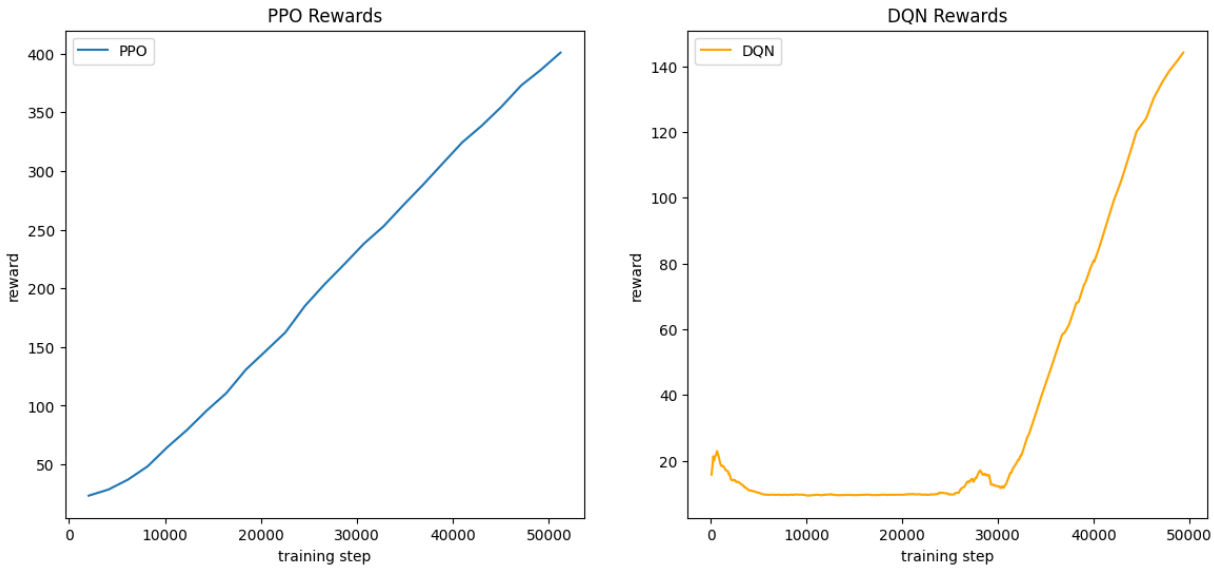

Figure 1: PPO and DQN learning curves for CartPole.

0.0003 to 0.001 which doesn't seem to have a meaningful effect for PPO but is helpful for DQN model. Also doubling the batch size seems to have a negative effect as figure 3 shows. After setting total steps

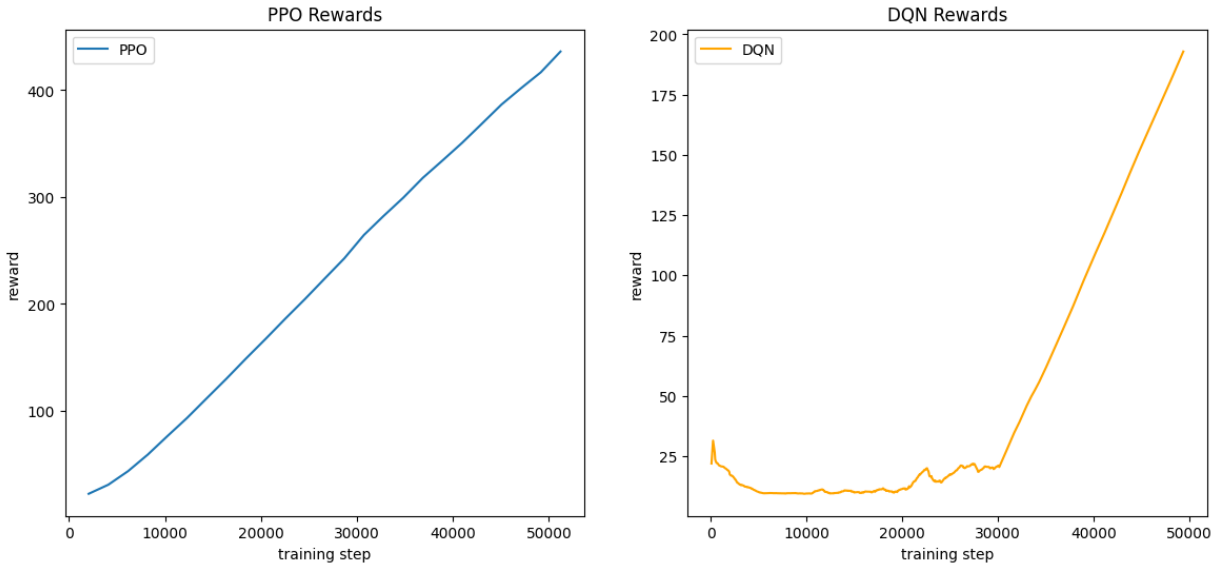

Figure 2: PPO and DQN learning curves for CartPole with lr=0.001.

to 150,000 you can see PPO has converged after about 80,000 episodes while DQN has not converged yet, which shows it's less sample efficient for cartpole. This aligns with expectations, as PPO generally provides better stability and efficiency in training compared to value-based methods like DQN, which can

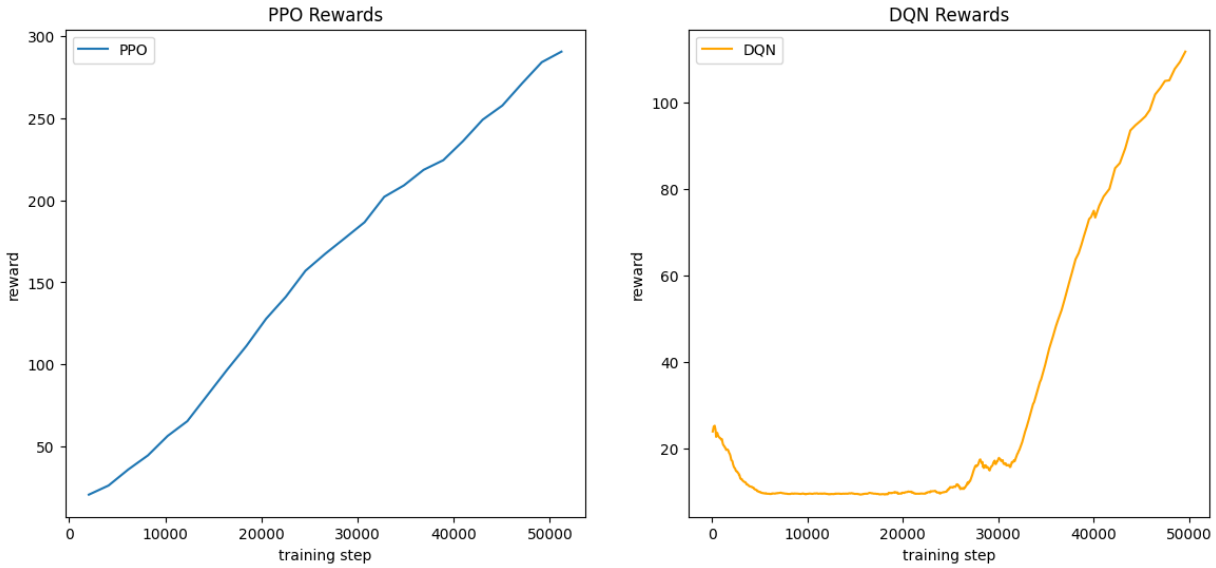

Figure 3: PPO and DQN learning curves for CartPole with batch size=128.

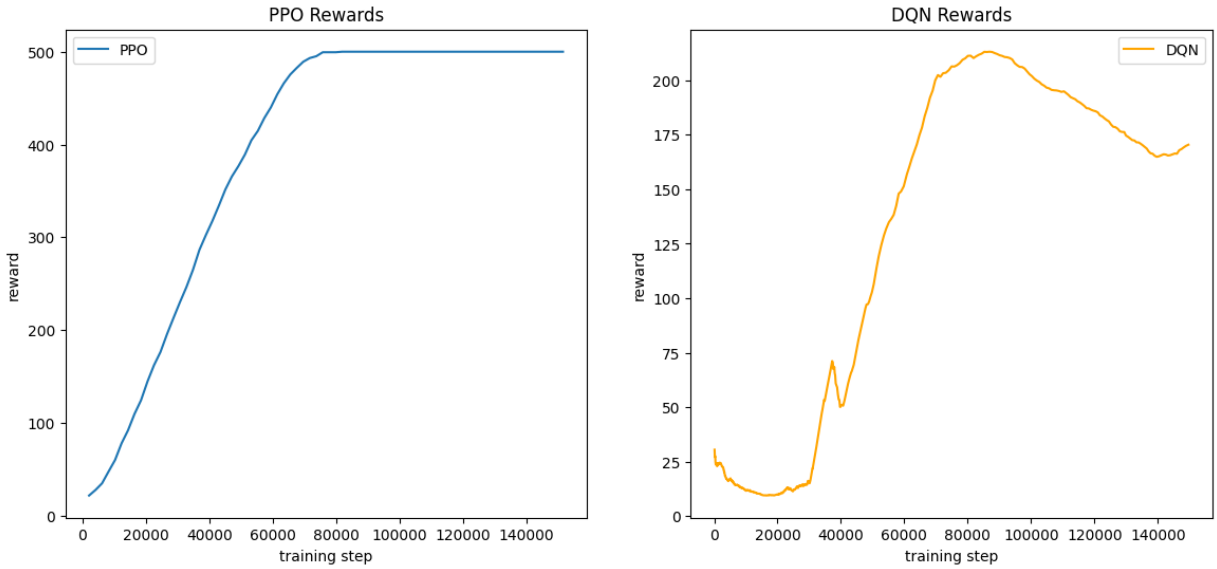

Figure 4: PPO and DQN learning curves for CartPole with 150,000 total time steps.

struggle with function approximation and exploration. I also experimented with the buffer size parameter for DQN. Reducing the buffer size did not degrade performance and even improved it. I re-trained the model with buffer size = 10,000. As shown in Figure 5, the mean reward peaked at step 100,000 but then started to decline, highlighting the instability of DQN.

As another example I picked Flappy Bird environment, where PPO performed a lot better than DQN due to more complex environment. You can see the resulting plots in Figures 6 & 7 & 8. Also a video is included.

In the end, I implemented a custom reward wrapper which doubles the original rewards, and re-trained with default setting on cartpole. You can see the graphs are exactly the same and only the reward values are doubled.

I have also included TensorBoard logs, one for each part. Due to their large size, I couldn't upload the notebook containing them, but the related files are provided.

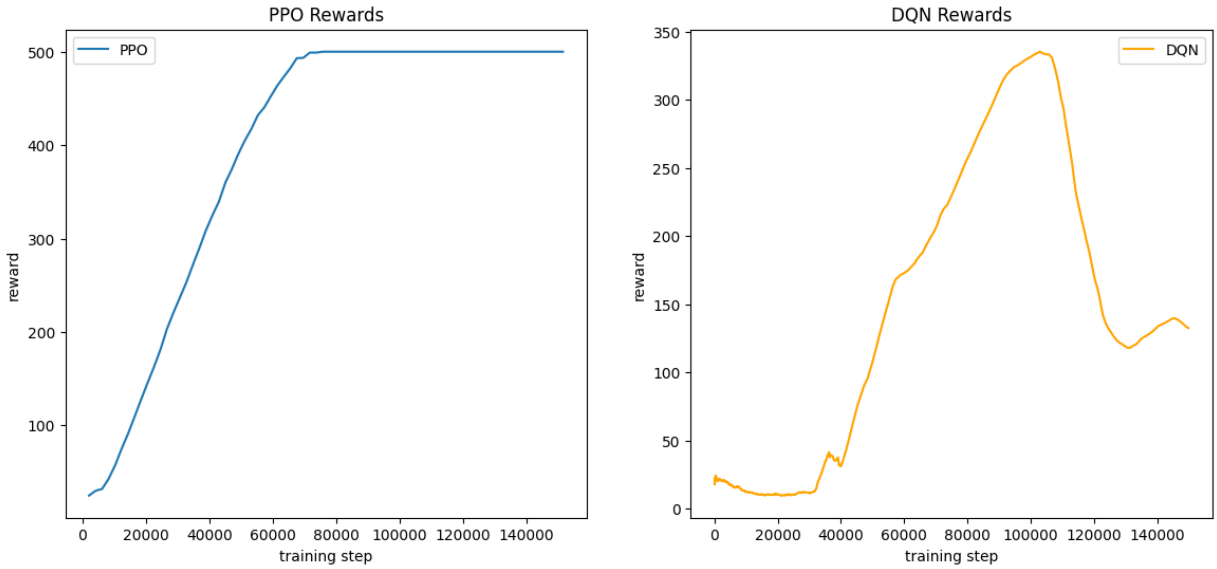

Figure 5: PPO and DQN learning curves for CartPole with 150,000 total time steps and buffer size=10000 for the DQN model.

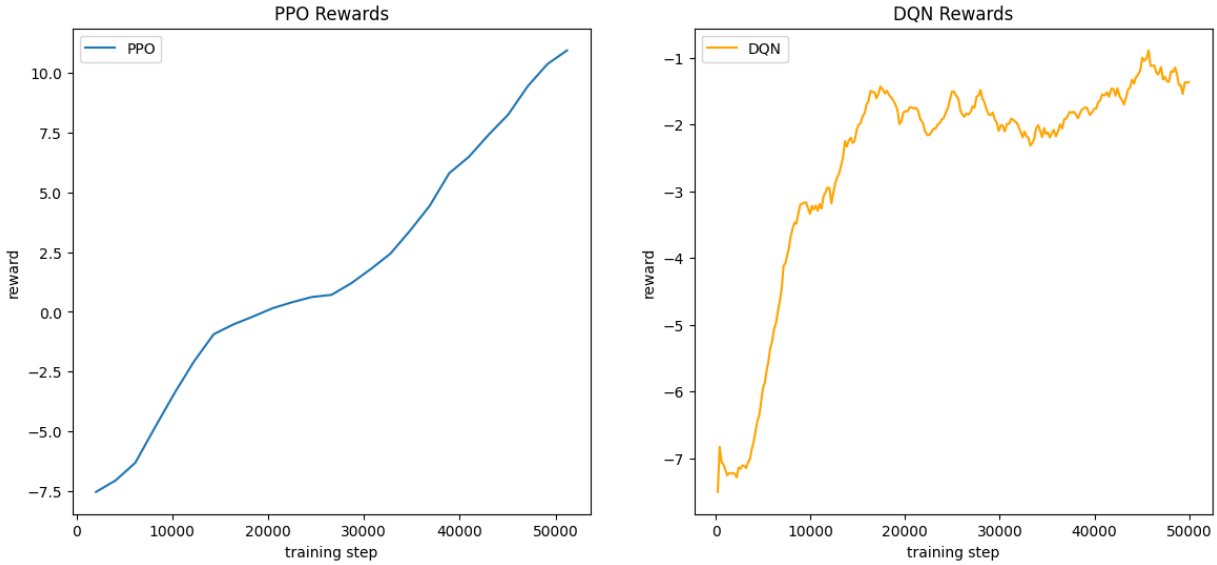

Figure 6: PPO and DQN learning curves for FlappyBird.

Conclusion: In Task 1, PPO and DQN were tested on CartPole and FlappyBird. PPO showed more stable learning curves and faster convergence, while DQN exhibited higher variance and slower learning. PPO was more sample efficient, requiring fewer episodes for convergence. Hyperparameter tuning, such as learning rate and buffer size adjustments, significantly impacted performance, with smaller buffer sizes improving DQN initially but leading to instability. These results highlight PPO's robustness and sample efficiency compared to DQN in these environments [7].

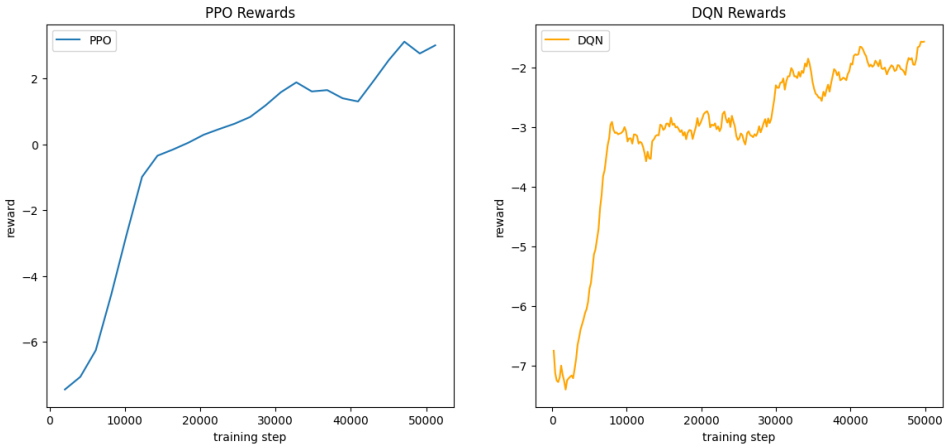

Figure 7: PPO and DQN learning curves for FlappyBird with lr=0.001 which hurts the performance.

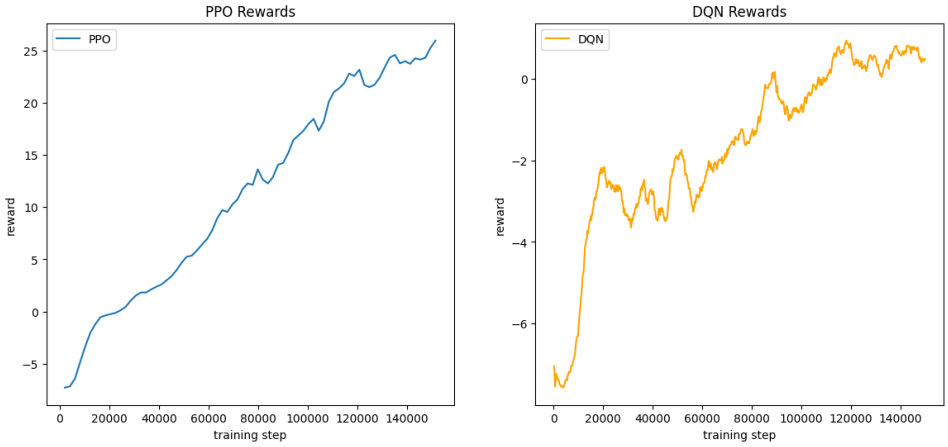

Figure 8: PPO and DQN learning curves for FlappyBird with 150,000 total time steps. PPO performs better.

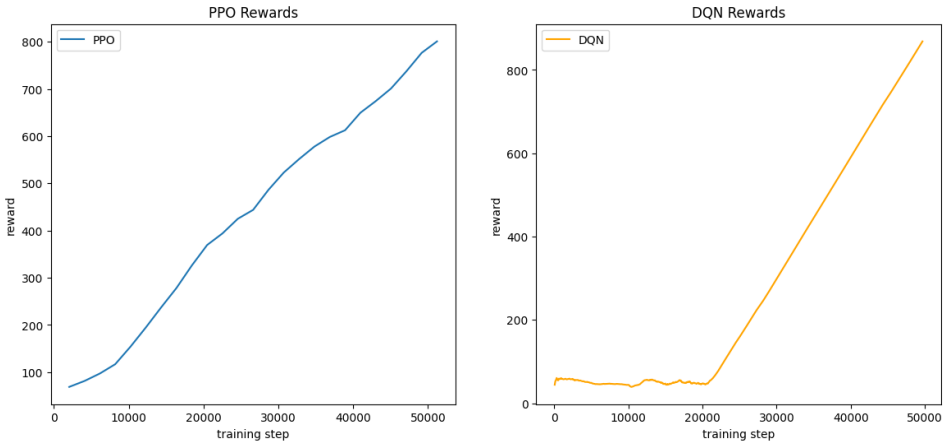

Figure 9: PPO and DQN learning curves for Cartpole with custom reward wrapper which doubles the reward.

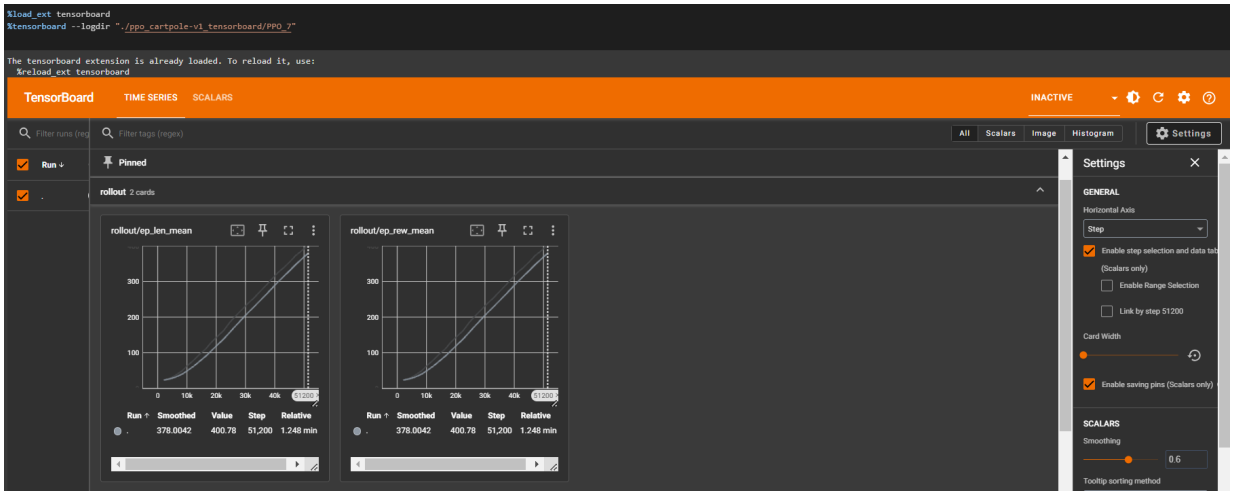

Figure 10: Tensorboard of PPO algorithm for CartPole.

Figure 11: Tensorboard of DQN algorithm for CartPole.

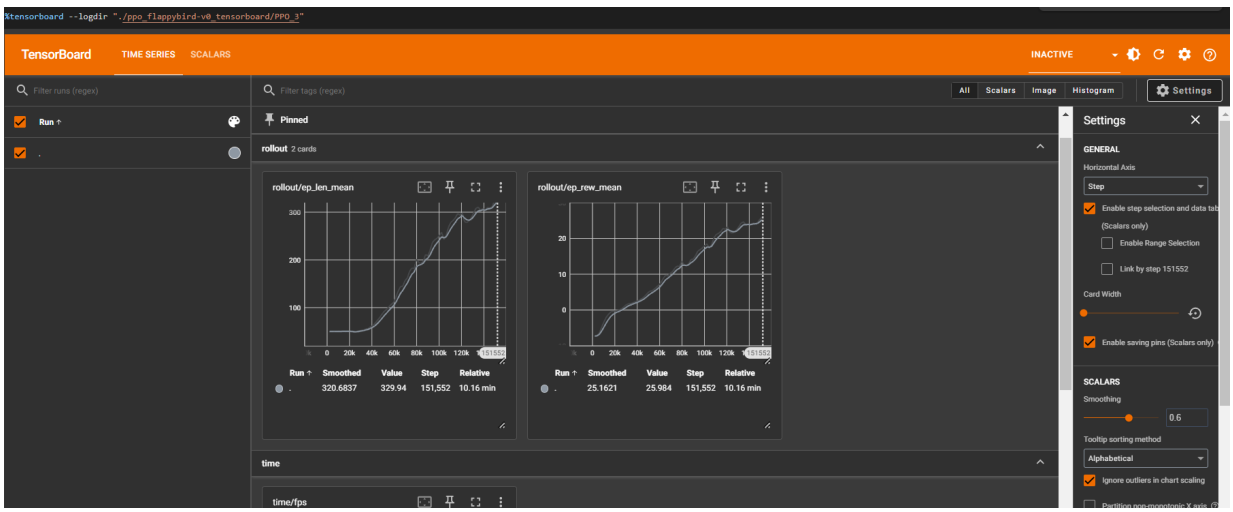

Figure 12: Tensorboard of PPO algorithm for FlappyBird.

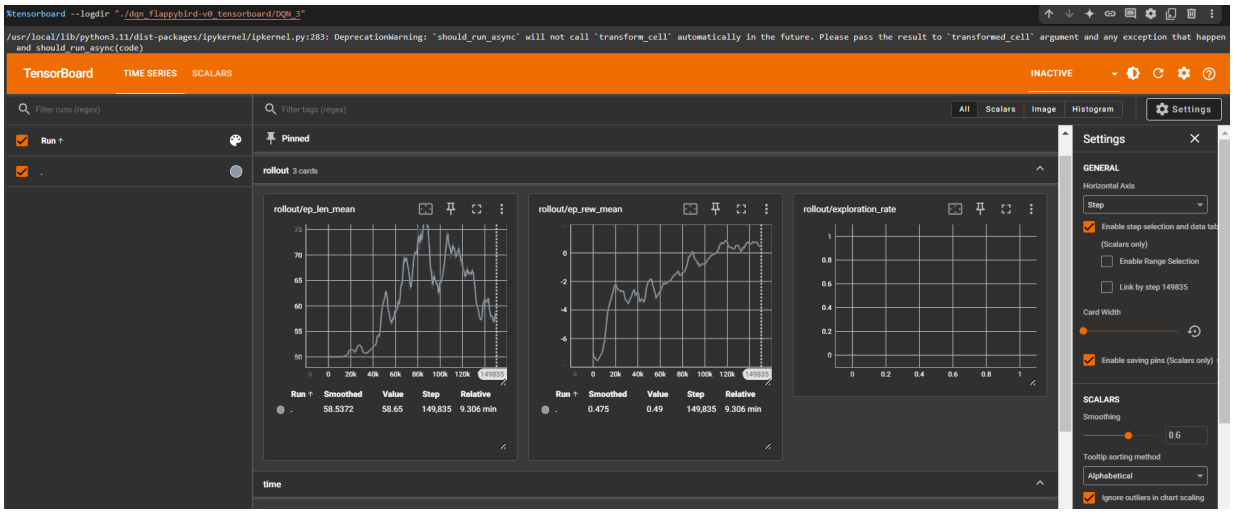

Figure 13: Tensorboard of DQN algorithm for FlappyBird.

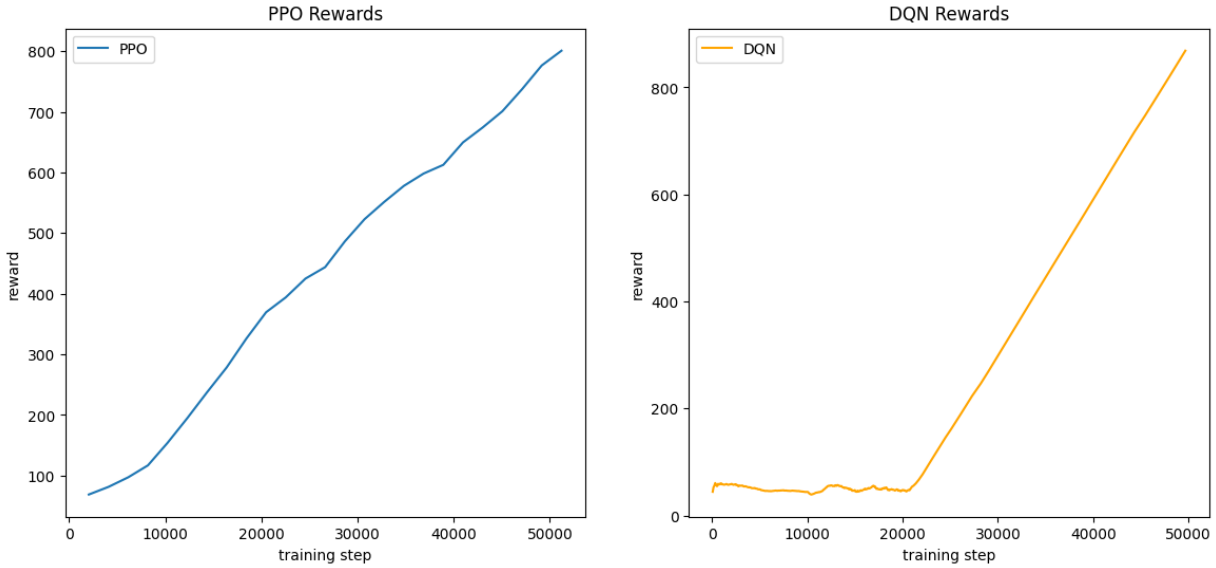

Figure 14: PPO and DQN learning curves for Cartpole with custom reward wrapper which doubles the reward.

2 Task 2: Creating Custom Environments [45-points]

For this part, I implemented a deterministic 4×4 MDP where the agent starts at (0,0) and the goal is (3,3). Cells (1,1) and (2,2) are unreachable. I then trained PPO and DQN on this custom environment. I also tried A2C, which was very sample-efficient and converged much faster than the other methods due to the simplicity of the environment. The reward-per-episode plots and TensorBoard logs for these models are shown in the upcoming figures. Additionally, I tested a learning rate of 0.001, which slightly improved convergence speed. Note that DQN remains unstable and is not a good choice for this problem, PPO works but may be overkill, and A2C is both efficient and fast.

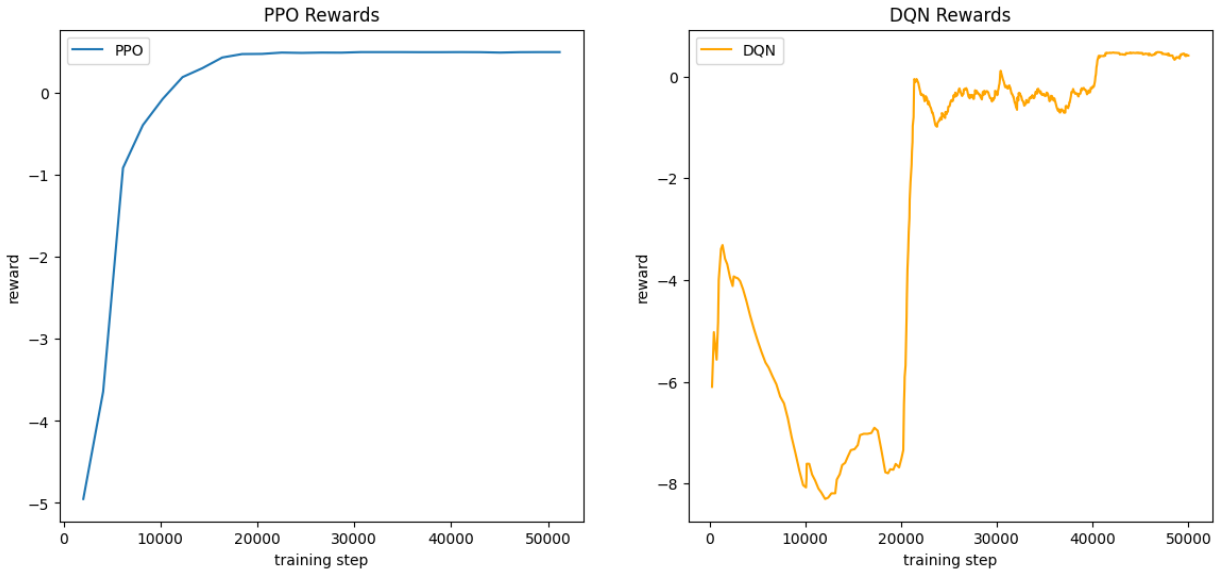

Figure 15: PPO and DQN learning curves for custom environment.

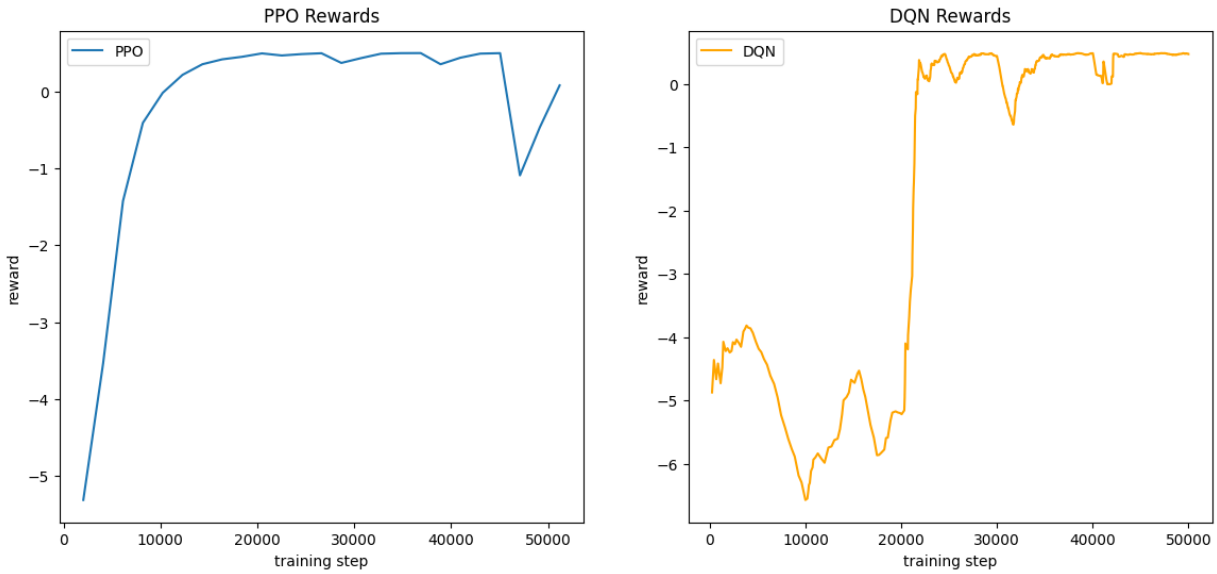

Figure 16: PPO and DQN learning curves for custom environment with lr=0.001.

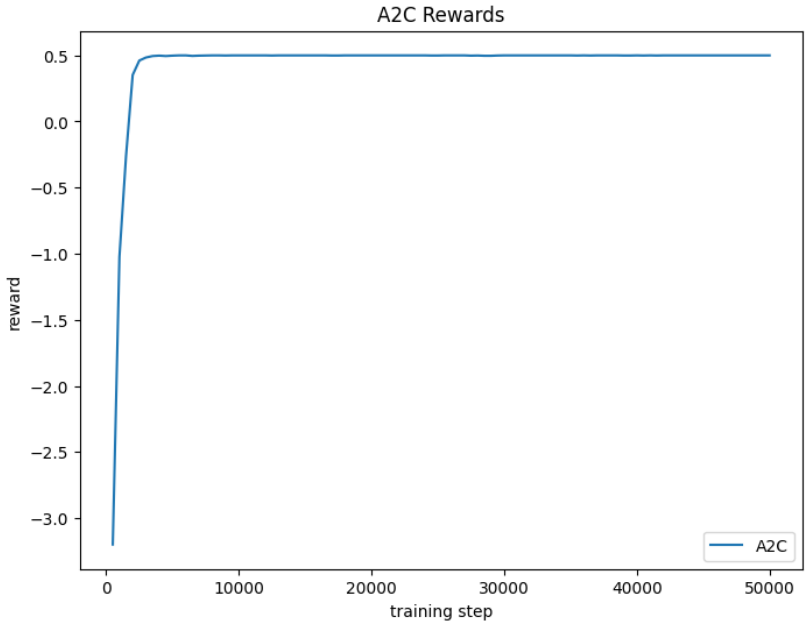

Figure 17: A2C learning curve for custom environment which is the most efficient method.

Figure 18: Tensorboard of PPO algorithm for custom environment.

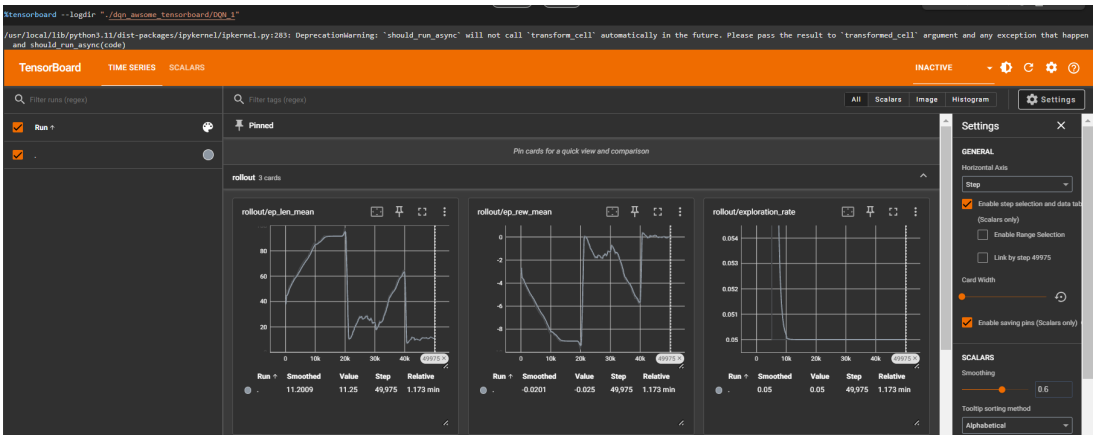

Figure 19: Tensorboard of DQN algorithm for custom environment.

3 Task 3: Pygame for RL environment [20-points]

For this section, I implemented a simple game where you have to jump over bananas! The environment state includes the distance to the next obstacle and a boolean indicating whether you are jumping or not. The available actions are to jump or stay on the ground. You receive a positive reward for each frame you survive, an additional reward for successfully jumping over bananas, and a penalty for unnecessary jumps and dying.

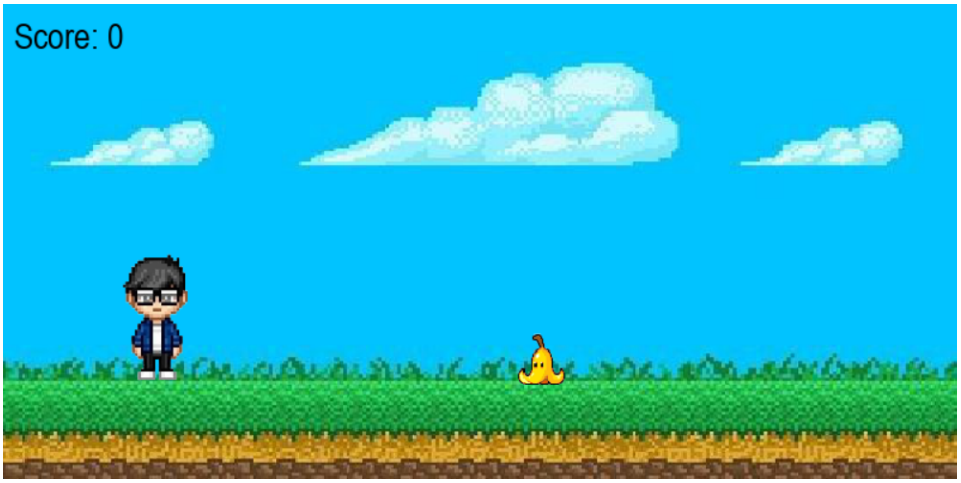

Figure 20: A picture of the custom game. you have to jump over bananas to survive.

I trained a PPO model on my game for 400,000 episodes, which seems to be the best option for this game. The reward-per-episode plot is included, along with a video of the model playing the game, showing that it can successfully jump over the first obstacle. However, several hours of training might be needed to reach higher levels!

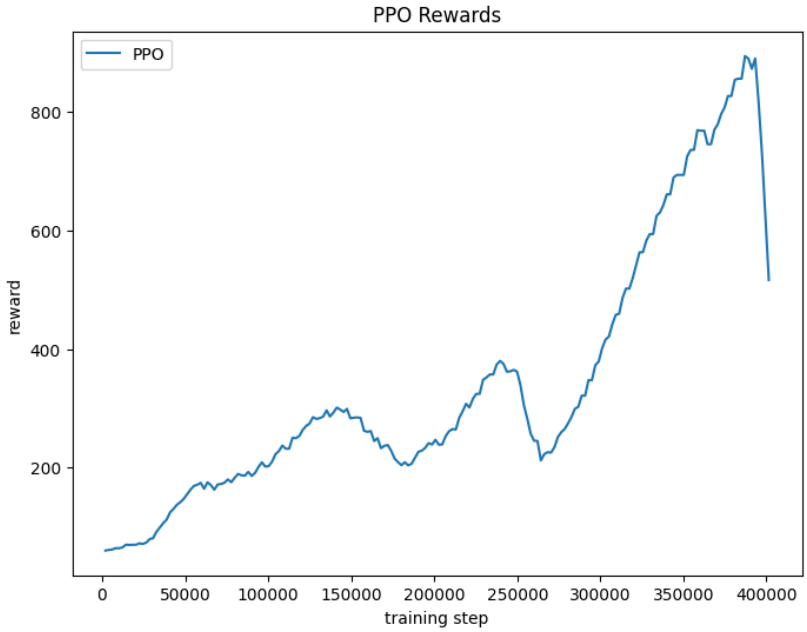

Figure 21: PPO reward curve on my game.

References

[1] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*, 2nd Edition, 2020. Available online: http://incompleteideas.net/book/the-book-2nd.html

[2] A. Raffin et al., "Stable Baselines3: Reliable Reinforcement Learning Implementations," GitHub Repository, 2020. Available: https://github.com/DLR-RM/stable-baselines3.

[3] Gymnasium Documentation. Available: https://gymnasium.farama.org/.

[4] Pygame Documentation. Available: https://www.pygame.org/docs/.

[5] CS 285: Deep Reinforcement Learning, UC Berkeley, Pieter Abbeel. Course material available: http://rail.eecs.berkeley.edu/deeprlcourse/.

[6] Cover image designed by freepik

[7] Conclusion paragraph written by ChatGPT