Assignment-3

Kunal Kumar Sahoo (2025AIZ8459)
AIL7022

November 28, 2025

Codebase: `https://github.com/Kunal-Kumar-Sahoo/AIL7022-Reinforcement-Learning/tree/main/Assignment-4`

1 DQN Overestimation

1. The `LunarLander-v3` environment has been trained using both Deep Q Network (DQN) and Double Deep Q Network (DDQN) having same network configuration 1 and hyperparameter settings 2 for fair comparisons. I have implemented Prioritized Experience Replay for storing experiences.

Table 1: DQN Network Architecture

	Layer Operation	Input Size	Output Size
Hidden Block 1			
1	Fully connected layer	8	256
2	Layer Normalization	256	256
3	Tanh activation	256	256
Hidden Block 2			
4	Fully connected layer	256	256
5	Layer Normalization	256	256
6	Tanh activation	256	256
Output Block			
7	Fully connected layer (Q-values)	256	4

Table 2: Hyperparameter Configuration for DQN and DDQN Training

Hyperparameter	**Value**
Batch Size	256
Discount Factor (γ)	0.99
Learning Rate	5×10^{-4}
Replay Buffer Capacity	150,000
Start Training After	20,000
Polyak Parameter	0.01
PER Priority Exponent (α)	0.6
PER IS Bias Start (β_0)	0.4
PER β Annealing Frames	200,000
PER TD Error Epsilon	1×10^{-6}
Boltzmann Temperature Start	5.0
Boltzmann Temperature End	0.1
Temperature Decay Frames	5,000,000
Training Episodes per Seed	20,000
Optimizer	AdamW
Gradient Clipping (norm)	1.0

2. For both networks, the best models have been saved as `models/dqn.pt` and `models/ddqn.pt`, respectively. Refer to Figure 1 for training rewards for both the algorithms. It was observed that training rewards of the DQN agent were higher than that of the DDQN agent.

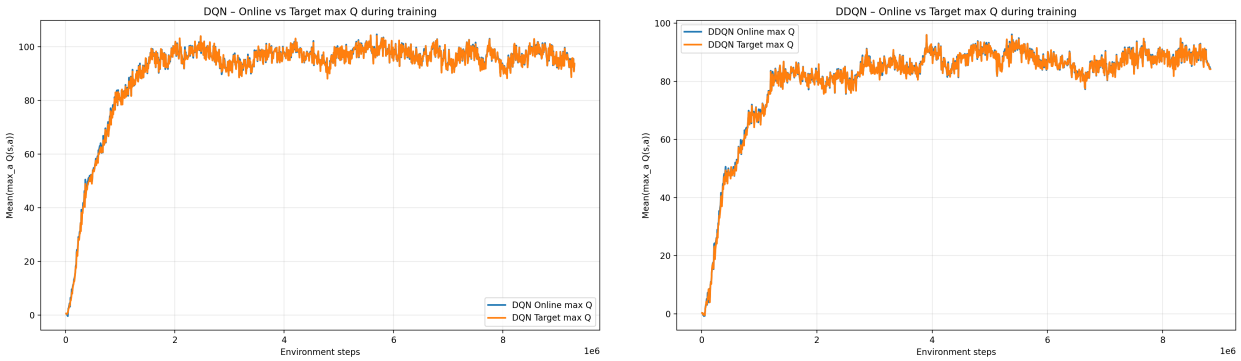


Figure 1: Training Rewards of DQN and DDQN respectively

3. Refer to Table 3 for evaluation results of DQN and DDQN agents across 100 episodes. Although training rewards for the DQN agent was higher, the evaluation result of the DDQN is higher with stricter deviation, which implies that DDQN learned a better policy.

Table 3: Evaluation results of the DQN and DDQN Agents

Architecture	Mean Reward	Std. Dev. Reward
DQN	101.34	192.41
DDQN	105.24	130.34

4. Refer to Figure 2 for the Q values predicted by both the agents during evaluation. It can be observed that DDQN predictions of Q values are very smooth for all the 4 actions as episodes evolve in comparison to DQN.

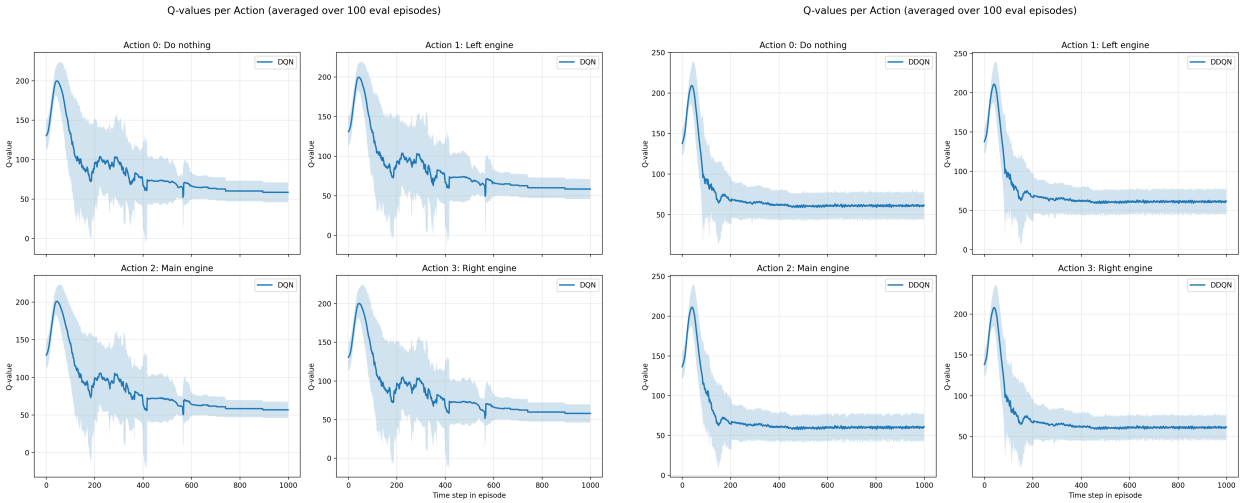


Figure 2: Q value predictions for DQN and DDQN respectively

2 Gradient Variance Analysis of REINFORCE Baselines

1. `InvertedPendulum-v5` environment was trained on the REINFORCE algorithm with a Gaussian policy network. The policy network is described in Table 4. The hyperparameters used for training are described in Table 5. The variants of baselines are described in Table 6

2. Refer to Figure 3 for visualizing the estimates of gradient estimates for different baselines of the REINFORCE algorithm. Across the four variants of REINFORCE, the gradients exhibit clear differences in both magnitude and variability. The vanilla version produces the largest and most volatile gradient magnitudes, reflecting its well-known high-variance nature. Introducing an average-reward baseline substantially reduces this variance, yielding smaller and more stable gradients. Reward-to-go further improves stability relative to the vanilla case, but its variance remains higher than that of the more principled baselines. The value function baseline provides the most consistent behaviour, achieving both the lowest mean gradient magnitude and the smallest

Table 4: Architecture of the policy and value networks

Network	Layer	Size	Activation
Policy	Input	4	–
Policy	Hidden layer 1	256 units	tanh
Policy	Hidden layer 2	256 units	tanh
Policy	Output (mean)	1	Linear
Policy	Log standard dev	1 (free param)	–
Value	Input	4	–
Value	Hidden layer 1	256 units	tanh
Value	Hidden layer 2	256 units	tanh
Value	Output	1	Linear

Table 5: Hyperparameter settings used in all experiments

Name	Symbol	Value
Discount factor	γ	0.99
Policy learning rate	α_π	3×10^{-4}
Value learning rate	α_V	10^{-3}
Maximum episodes	$N_{\text{epi}}^{\max}$	40000
Hidden layer sizes	–	$(256, 256)$
Value updates per batch	K_V	10
Collected trajectories	N_τ	500

standard deviation across all sample sizes. Overall, the results confirm that better baseline estimators lead to more stable and sample-efficient gradient estimates.

Table 6: Policy gradient variants, objective functions and implemented loss

Variant	**Advantage** A_t	**Policy objective** $J(\theta)$
None	$A_t = G_t$	$J(\theta) = \mathbb{E}\Big[\sum_t \log \pi_\theta(a_t \mid s_t) G_t\Big]$
Average reward baseline	$A_t = G_t - \bar{G}$	$J(\theta) = \mathbb{E}\Big[\sum_t \log \pi_\theta(a_t \mid s_t)\big(G_t - \bar{G}\big)\Big]$
Reward to go	$A_t = G_t$	$J(\theta) = \mathbb{E}\Big[\sum_t \log \pi_\theta(a_t \mid s_t) G_t\Big]$
Value function baseline	$A_t = G_t - V_\phi(s_t)$	$J(\theta) = \mathbb{E}\Big[\sum_t \log \pi_\theta(a_t \mid s_t)\big(G_t - V_\phi(s_t)\big)\Big]$

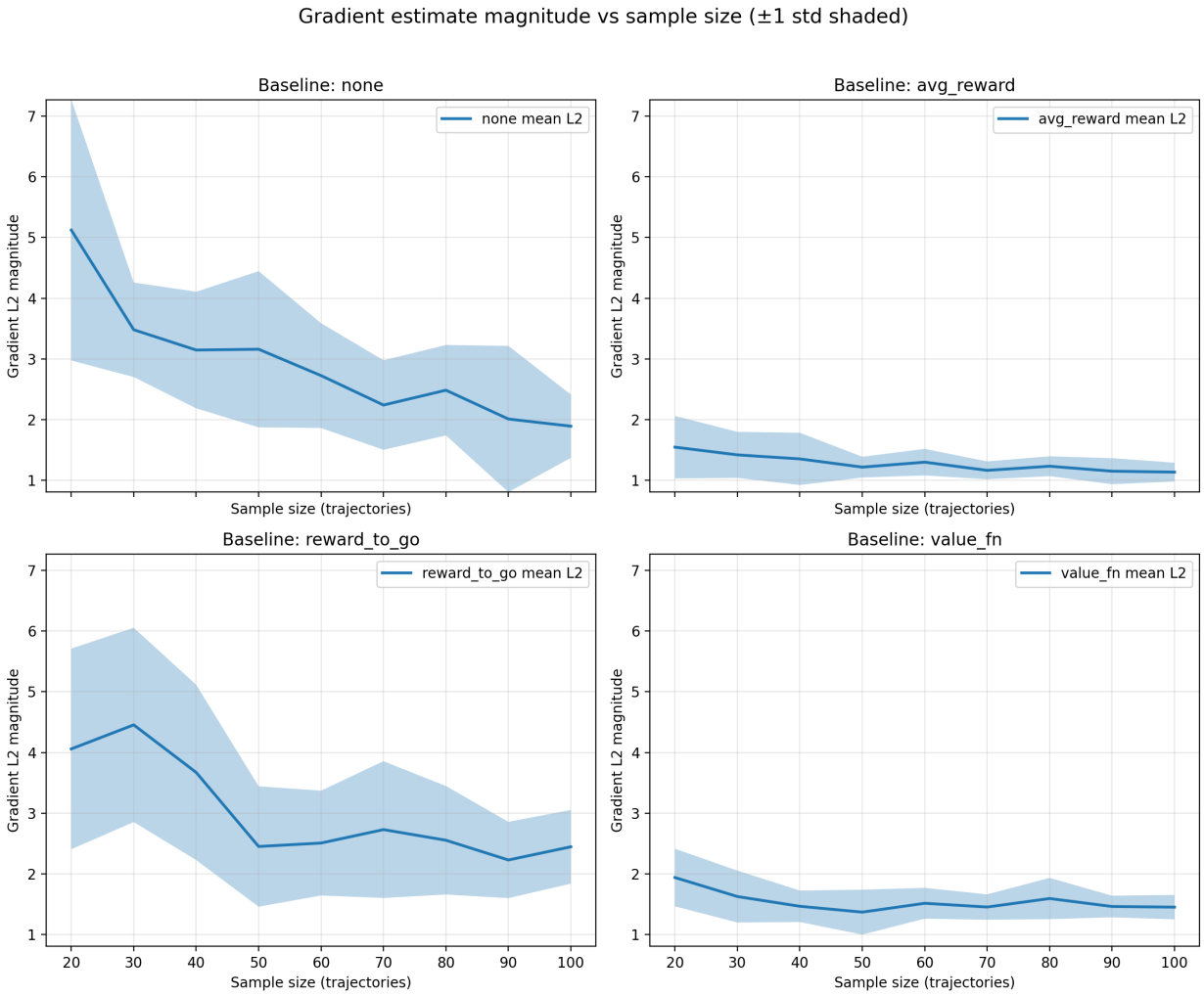


Figure 3: Gradient estimate for different baselines across different sample sizes

3 Stable Baselines

1. Refer to Table 7- 9 for implementation details.

Table 7: A2C hyperparameters for different MuJoCo environments

Hyperparameter	**InvertedPendulum-v5**	**Hopper-v5**	**HalfCheetah-v5**
Total timesteps	1×10^5	2×10^6	1×10^6
Learning rate	7×10^{-4}	7×10^{-4}	7×10^{-4}
n_{steps}	5	2048	2048
Discount factor γ	0.99	0.99	0.99
GAE λ	1.0	1.0	1.0
Value loss coefficient	0.5	0.5	0.5
Max grad norm	0.5	0.5	0.5

Table 8: PPO hyperparameters for different MuJoCo environments

Hyperparameter	**InvertedPendulum-v5**	**Hopper-v5**	**HalfCheetah-v5**
Total timesteps	1×10^5	2×10^6	1×10^6
Learning rate	3×10^{-4}	3×10^{-4}	3×10^{-4}
n_{steps}	2048	2048	2048
Batch size	64	64	64
Number of epochs	10	20	10
Discount factor γ	0.99	0.99	0.99
GAE λ	0.95	0.95	0.95
Clip range	0.2	0.2	0.2

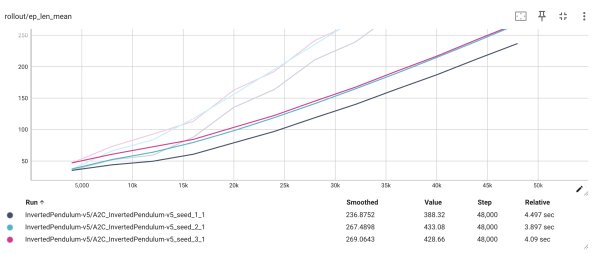
Table 9: Network configurations (policy and value networks) used with `MlpPolicy`

Environment	**Policy network hidden sizes**	**Value network hidden sizes**
InvertedPendulum-v5	$[64, 64]$	$[64, 64]$
Hopper-v5	$[512, 512]$	$[512, 512]$
HalfCheetah-v5	$[1024, 1024]$	$[1024, 1024]$

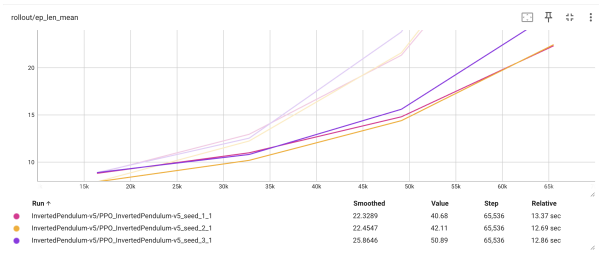
2. The mean episode lengths, mean episode reward, and policy gradient losses for A2C and PPO for different environments are presented below. Due to some issues (couldn't figure out) the TensorBoard logs for HalfCheetah PPO were not logged.:

3. Evaluation GIFs are as follows:

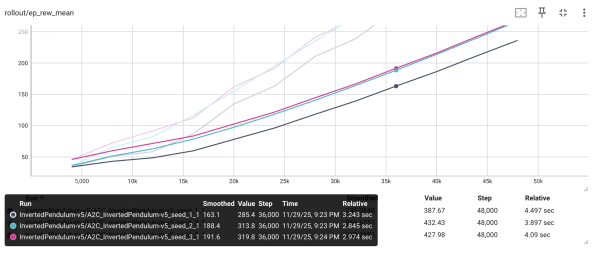
   - `InvertedPendulum-v5`: `https://drive.google.com/drive/folders/1vunDtmBly9QROzuZiVbxuIoK8LqbdBr3?usp=sharing`
   - `Hopper-v5`: `https://drive.google.com/drive/folders/1ByesvH6Spo_JkEoildcwY8maeeUVMgj?usp=sharing`
   - `HalfCheetah-v5`: `https://drive.google.com/drive/folders/1b99g0anxd9h6tPX1K6mNMVEWuxPVvbaE?usp=sharing`



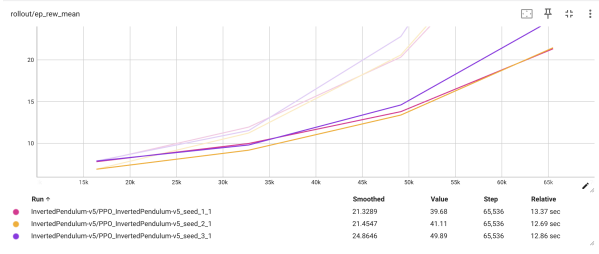
(a) Mean Episode Length A2C



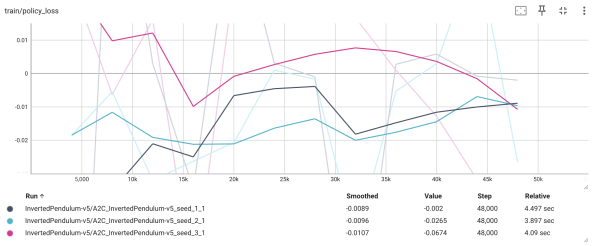
(b) Mean Episode Length PPO



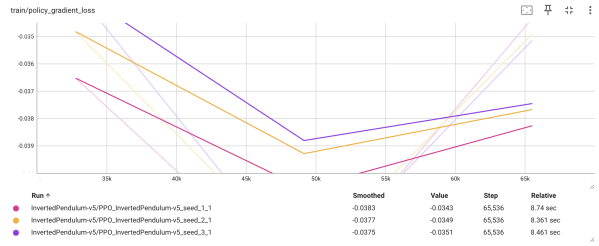
(c) Mean Episode Reward A2C



(d) Mean Episode Reward PPO



(e) Policy Gradient Loss A2C

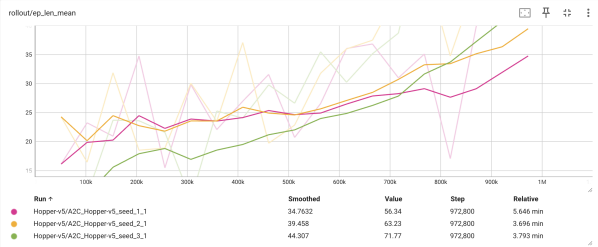


(f) Policy Gradient Loss PPO

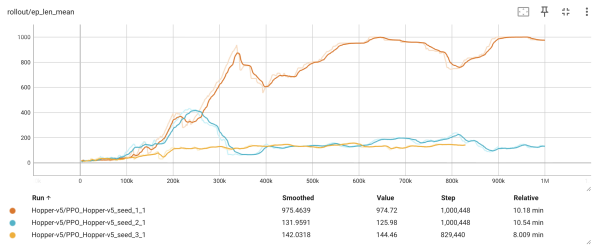
Figure 4: Training plots for `InvertedPendulum-v5`

4 Policy Gradients vs DQN

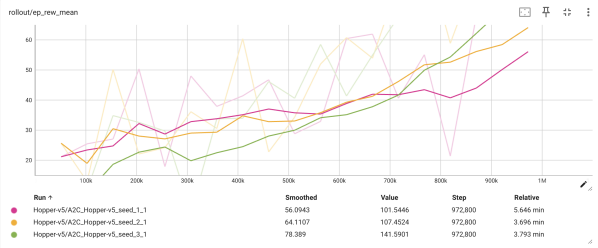
1. Refer to Table 10 for network configurations for both the agents DQN and A2C. The hyperparameters for training of both the agents have been presented in 11



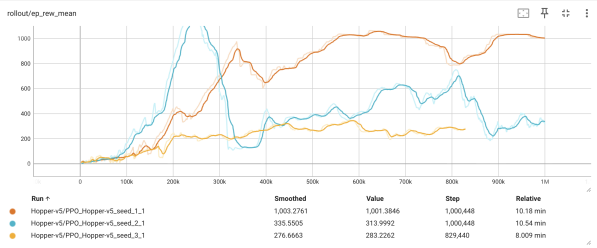
(a) Mean Episode Length A2C



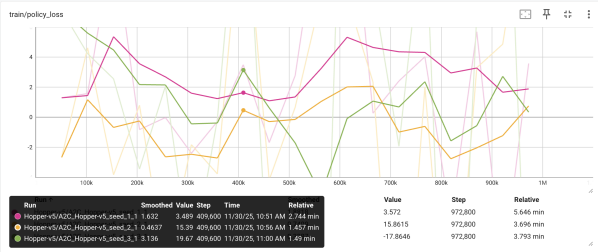
(b) Mean Episode Length PPO



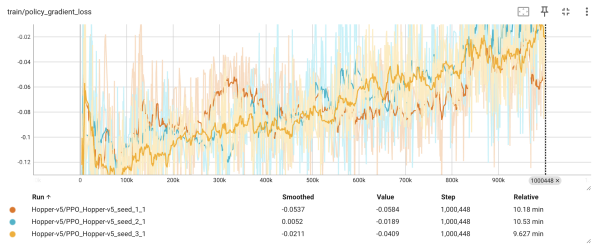
(c) Mean Episode Reward A2C



(d) Mean Episode Reward PPO



(e) Policy Gradient Loss A2C

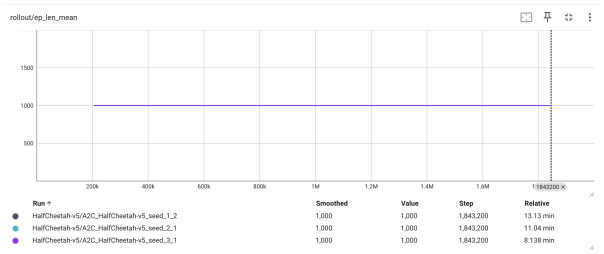


(f) Policy Gradient Loss PPO

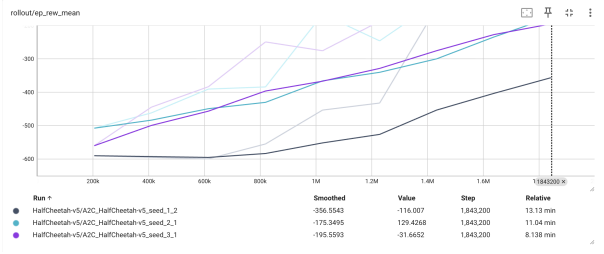
Figure 5: Training plots for `Hopper-v5`

Table 10: Network architectures for A2C (Actor–Critic) and DQN on `LunarLander-v3`

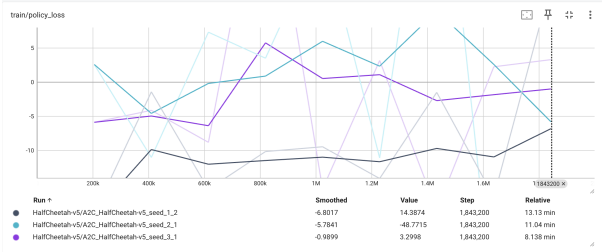
Algorithm	Network	Layer	Specification
A2C	Actor (policy $\pi_\theta(a \mid s)$)	Input	8
		FC1	Linear($8 \to 128$) + LayerNorm(128) + ReLU
		FC2	Linear($128 \to 128$) + LayerNorm(128) + ReLU
		Output	Linear($128 \to 4$) + Softmax over 4 discrete actions
A2C	Critic (value $V_\phi(s)$)	Input	8
		FC1	Linear($8 \to 128$) + LayerNorm(128) + ReLU
		FC2	Linear($128 \to 128$) + LayerNorm(128) + ReLU
		Output	Linear($128 \to 1$) (state value, no activation)
DQN	Q-network $Q_\psi(s, a)$	Input	8
		FC1	Linear($8 \to 128$) + LayerNorm(128) + ReLU
		FC2	Linear($128 \to 128$) + LayerNorm(128) + ReLU
		Output	Linear($128 \to 4$) (Q-value for each of 4 actions)



(a) Mean Episode Length A2C



(b) Mean Episode Reward A2C



(c) Policy Gradient Loss A2C

Figure 6: Training plots for `HalfCheetah-v5`

2. The training wall time for both agents is as follows:

   - **DQN**: 857 seconds
   - **A2C**: 272 seconds

   One interesting observation is that DQN took fewer number of episodes to finish training as compared to A2C, but since DQN was performing batch updates from an experience buffer as compared to online updates of A2C, the overall wall time of A2C was lesser.

3. The loss curves for both agents are presented in Figure 7. The actor loss begins with large oscillations and gradually moves toward zero, indicating increasing policy stability as the critic improves. The critic loss shows sharp spikes early in training but quickly converges to near-zero values, reflecting stable value estimation after the initial learning phase. In contrast, DQN loss starts very high and drops rapidly but continues to fluctuate throughout training due to its off-policy nature and bootstrapped Q-value updates. Overall, A2C losses stabilize more smoothly, while DQN maintains higher variance even after convergence.

Table 11: Hyperparameters used for A2C and DQN

Category	Hyperparameter	Value
General	Seed	42
	Max Episodes	1500
	Max Steps per Episode	1000
	Discount Factor γ	0.99
A2C	Actor Learning Rate	3×10^{-4}
	Critic Learning Rate	1×10^{-3}
	Hidden Layer Size	128
	Entropy Coefficient (start)	0.05
	Entropy Coefficient (end)	0.001
	Entropy Decay	0.995
	Gradient Clipping	1.0
DQN	Learning Rate	3×10^{-4}
	Hidden Layer Size	128
	Replay Buffer Size	50000
	Batch Size	64
	Epsilon (start)	1.0
	Epsilon (end)	0.05
	Epsilon Decay	0.995
	Soft Target Update τ	0.005
	Gradient Clipping	1.0

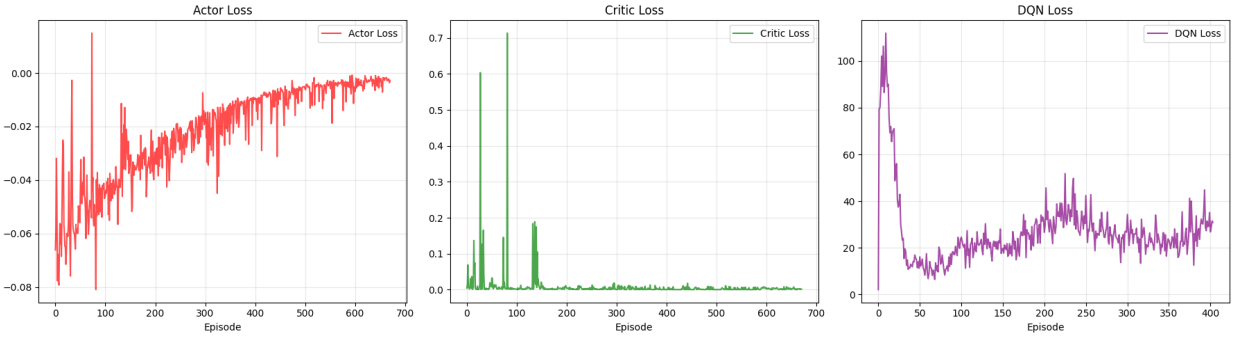


Figure 7: Training losses for DQN and A2C agent

4. The training reward curves are presented in Figure 8. The reward curves for both A2C and DQN show a clear upward trend, indicating overall learning progress from highly negative rewards toward positive returns. DQN improves more rapidly in the early stages and reaches higher rewards sooner, while A2C learns more gradually and exhibits greater variance later in training. Although both methods show improvement, neither curve increases strictly monotonically; both exhibit fluctuations and occasional drops due to stochastic exploration and the complexity of the task. Overall, DQN

displays faster and more stable early learning, whereas A2C shows slower but steady long-term improvement.

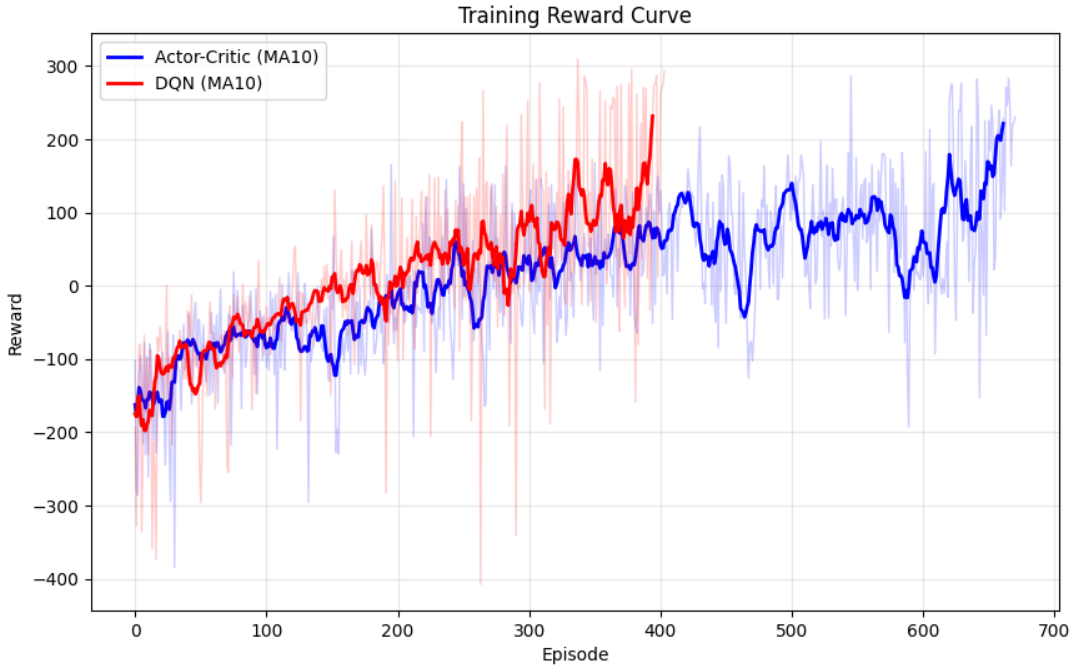


Figure 8: Training rewards for DQN and A2C agent

5. Refer to the following URLs to see the evaluation GIFs of both agents:
   - **DQN**: `https://drive.google.com/drive/folders/1onndZmchKJqa-n9_NTRcw6G-8ntmVHVB?usp=sharing`
   - **A2C**: `https://drive.google.com/drive/folders/1P2K1i6I4C5kkXE-Zhxs4JG-Hk9NJZYrs?usp=sharing`
6. The evaluation rewards for both agents are as follows. The plots are presented in Figure 9:
   - **DQN**: 180.48 ± 48.65
   - **A2C**: 192.32 ± 20.19

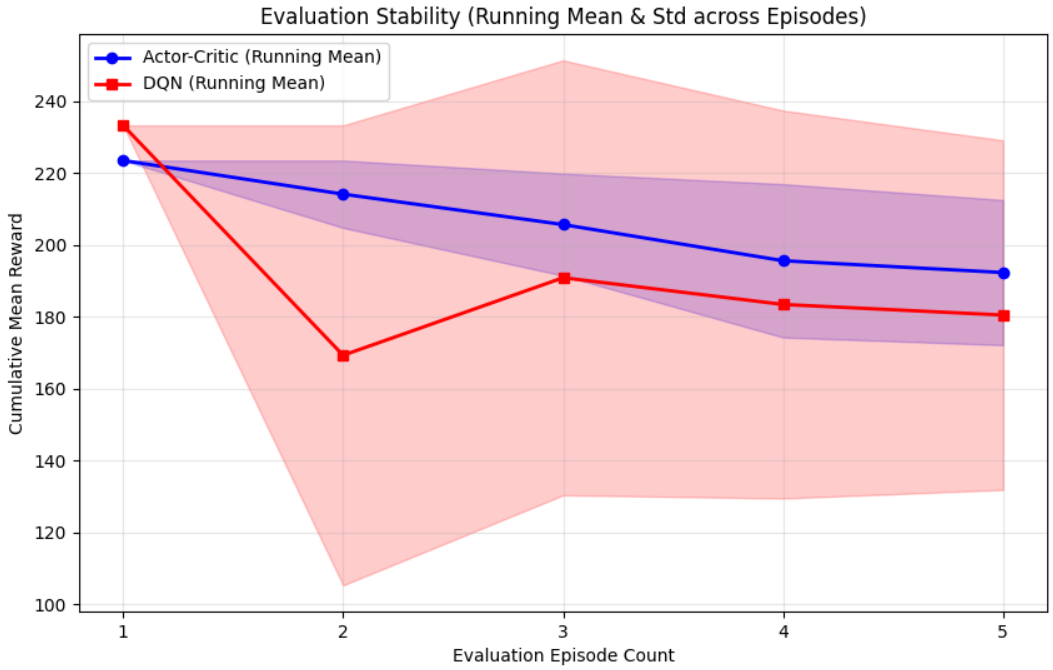


Figure 9: Evaluation rewards for DQN and A2C agent