Comparative Analysis of Deep Reinforcement Learning Algorithms

Assignment 1

Course: Deep Reinforcement Learning (21ARE402)

Karthik M (CH.EN.U4ARE22012)
Bhavesh Durai S (CH.EN.U4ARE22026)
Jaswanth G (CH.EN.U4ARE22027)

15 September 2025

Abstract

This report presents a comprehensive analysis of deep reinforcement learning (DRL) algorithms applied to the classic control problem, Lunar Lander. The primary objective is to implement and evaluate the performance of three distinct DRL agents from different families: a value-based method (Deep Q-Network, DQN), an on-policy actor-critic method (Proximal Policy Optimization, PPO), and an off-policy actor-critic method (Deep Deterministic Policy Gradient, DDPG). The performance of these algorithms is benchmarked against a random policy baseline, and the impact of a custom reward shaping strategy is analyzed to evaluate its effect on learning speed and stability. Our findings demonstrate that PPO provides the most effective solution, exhibiting superior stability and faster convergence to a high-performance policy. While DQN offers a stable but slower learning trajectory, DDPG shows high initial sample efficiency but suffers from significant instability. The analysis further confirms that our custom reward shaping successfully accelerated learning and led to more robust final policies for all tested agents. Ultimately, this work highlights the critical impact of algorithm choice in DRL, underscoring the strengths of on-policy methods like PPO for complex control tasks.

Contents

0.1 Key Literature Summary

The theoretical groundwork for this project is built upon the following seminal contributions in reinforcement learning. Each paper introduced critical concepts that directly influence the algorithms implemented in this study.

- **Learning from Delayed Rewards (1989)**

  Author: Christopher J. C. H. Watkins

  This foundational PhD thesis introduced Q-Learning, the off-policy temporal-difference algorithm that forms the bedrock of modern value-based methods. Its core takeaway is the ability to learn an optimal action-value function independently of the policy being followed during exploration.

- **Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning (1992)**

  Author: Ronald J. Williams

  This paper introduced the REINFORCE algorithm, a fundamental policy gradient method. The key takeaway is the principle of directly optimizing a policy's parameters by increasing the probability of actions that lead to higher returns, forming the basis of the policy-based paradigm.

- **Human-level control through deep reinforcement learning (2015)**

  Authors: Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, & Demis Hassabis

  This landmark paper introduced the Deep Q-Network (DQN). The crucial takeaway is the demonstration that a deep neural network could be trained to master complex control tasks from raw sensory input by using two key innovations for stability: **Experience Replay** and **Target Networks**.

- **Deterministic Policy Gradient Algorithms (2014)**

  Authors: David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, & Martin Riedmiller

  This work provided the theoretical foundation for DDPG. Its key takeaway is that for continuous action spaces, a deterministic policy gradient can be significantly more sample-efficient than a stochastic one, offering a direct path to optimizing policies in high-dimensional action spaces.

- **Continuous control with deep reinforcement learning (2015)**

  Authors: Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, & Daan Wierstra

  This paper introduced the Deep Deterministic Policy Gradient (DDPG) algorithm. The main takeaway is the successful combination of DQN's stability-enhancing features (replay buffer and target networks) with an actor-critic framework based on the Deterministic Policy Gradient, creating a robust algorithm for continuous control.

0.2 Reinforcement Learning Paradigms

Reinforcement Learning (RL) is a paradigm of machine learning where an agent learns to make sequential decisions by interacting with an environment to maximize a cumulative reward signal. The primary RL paradigms are:

- **Value-Based Methods:** These methods focus on learning a value function that estimates the expected return for being in a given state (or taking an action in a state). The policy is implicit and derived directly from the value function by selecting the action that leads to the highest value. A prime example is **Q-Learning**.
- **Policy-Based Methods:** These methods directly learn the policy, which is a mapping from states to actions, without necessarily needing a value function. Policy-gradient methods, like REINFORCE, optimize the policy parameters by performing gradient ascent on the expected return.
- **Actor-Critic Methods:** These methods combine the strengths of both value-based and policy-based approaches. They consist of two components: an **Actor**, which is a policy that decides which action to take, and a **Critic**, which is a value function that evaluates the action taken by the actor. The critic's feedback is then used to improve the actor's policy.

0.3 Strengths and Limitations of Selected Algorithms

For this project, we selected Deep Q-Network (DQN) from the value-based family and Deep Deterministic Policy Gradient (DDPG) from the actor-critic family.

0.3.1 Deep Q-Network (DQN)

- **Strengths:**
  - Highly effective and stable for problems with discrete, low-dimensional action spaces.
  - Utilizes an experience replay buffer to break correlations between consecutive samples, leading to more stable training.
  - Employs a target network to stabilize the learning process by providing consistent Q-value targets.
- **Limitations:**
  - Cannot handle continuous action spaces directly, as finding the maximizing action becomes a complex optimization problem at each step.
  - Can suffer from overestimation of Q-values, although variations like Double DQN mitigate this.

0.3.2 Deep Deterministic Policy Gradient (DDPG)

- **Strengths:**
  - An actor-critic method that can effectively operate in continuous action spaces.
  - Builds upon the ideas of DQN, using an experience replay buffer and target networks for stable learning.
  - Generally more sample-efficient than standard policy gradient methods.
- **Limitations:**

- Highly sensitive to hyperparameters, making it difficult to tune.
- The deterministic policy can struggle with exploration compared to stochastic policies.
- Can be computationally more expensive due to having four networks (actor, critic, target actor, target critic).

1 Problem Formulation: Lunar Lander

The Lunar Lander environment is a classic control problem where the objective is to land a spacecraft on a landing pad located at coordinates $(0, 0)$.

1.1 State Space (S)

The state is represented by an 8-dimensional continuous vector:

$$s = [x, y, v_x, v_y, \theta, \dot{\theta}, l, r]$$

where:

- x, y: Positional coordinates.
- v_x, v_y: Linear velocities.
- θ: Angle of the lander.
- $\dot{\theta}$: Angular velocity.
- l, r: Boolean flags indicating if the left or right leg has ground contact.

1.2 Action Space (A)

The environment has a discrete action space with 4 possible actions:

1. Do nothing.
2. Fire left orientation engine.
3. Fire main engine.
4. Fire right orientation engine.

Note: For DDPG, which requires a continuous action space, we can adapt this by using a multi-discrete output or by treating the actions as a continuous vector and then discretizing the output.

1.3 Reward Design

The reward function is shaped to guide the agent towards a successful landing:

- **Positive Rewards:**
  - Moving closer to the landing pad.
  - A successful, soft landing provides a large reward of +100 points.
  - Each leg making ground contact gives +10 points.
- **Negative Rewards (Penalties):**

- Moving away from the landing pad.
- Crashing or landing outside the pad results in a large penalty of -100 points.
- Firing the main engine costs -0.3 points per frame.
- Firing a side engine costs -0.03 points per frame.

1.4 Termination Conditions

An episode terminates if one of the following conditions is met:

- The lander crashes.
- The lander comes to a complete rest on the landing pad.
- The lander flies out of the screen boundaries.

2 Implementation and Experimentation

2.1 Environment: Lunar Lander (v3)

The experimental testbed for this project is the **LunarLander-v3** environment from the Gymnasium library. It was chosen as it represents a classic control problem that is computationally simple yet complex enough to highlight the differences between various reinforcement learning algorithms.

The environment is characterized by:

- **State Space:** A continuous, 8-dimensional vector representing the lander's physical state: $[x, y, v_x, v_y, \theta, \dot{\theta}, l, r]$, which correspond to position, linear velocity, angle, angular velocity, and two boolean leg contact flags.
- **Action Space:** A discrete space with 4 possible actions: do nothing, fire left engine, fire main engine, or fire right engine.

The objective is to train an agent to navigate the lander from a random starting position to a safe landing on a designated pad at coordinates $(0, 0)$, while minimizing fuel consumption.

2.2 Algorithm Selection: DQN, PPO, and DDPG

To provide a robust comparison across different RL paradigms, three influential algorithms from distinct families were implemented using the Stable-Baselines3 library. This selection allows for a comprehensive analysis of value-based, on-policy actor-critic, and off-policy actor-critic methods.

- **Deep Q-Network (DQN):** An off-policy, **value-based** algorithm. It learns a Q-function to estimate the expected return of actions and uses an experience replay buffer to improve sample efficiency and stability. It is a benchmark for discrete action space problems.
- **Proximal Policy Optimization (PPO):** An on-policy, **actor-critic** algorithm. PPO is known for its stability and reliable performance, achieved by using a clipped surrogate objective function that prevents destructively large policy updates.
- **Deep Deterministic Policy Gradient (DDPG):** An off-policy, **actor-critic** algorithm. DDPG learns a deterministic policy (actor) and a Q-function (critic) and is primarily designed for continuous action spaces. Its inclusion provides an interesting comparison point, testing its adaptability to a discrete environment.

2.3 Baseline Heuristic: Random Policy

To establish a performance floor, a **Random Policy Baseline** was implemented. This heuristic agent selects an action uniformly at random at each timestep, representing a "zero-intelligence" approach. Any successfully trained agent must demonstrate a significant performance improvement over this baseline.

2.4 Experimental Protocol and Hyperparameters

All agents were trained for a total of **100,000 timesteps** in a Google Colab environment with a GPU accelerator. The training run was completed in **184.34s** for DQN and **325.63s** for PPO. The specific hyperparameters for each algorithm are detailed below.

2.4.1 DQN Hyperparameters

The parameters for DQN are listed in Table 1.

Table 1: Hyperparameters for the DQN Agent

Hyperparameter	**Value**	**Description**
Learning Rate (α)	0.001	Step size for the Adam optimizer.
Discount Factor (γ)	0.99	Weight for future rewards.
Batch Size	64	Number of samples per gradient update.
Buffer Size	100,000	Size of the experience replay buffer.
Network Architecture	[64, 64]	Two hidden layers with 64 neurons each.

2.4.2 PPO Hyperparameters

The parameters for PPO are listed in Table 2.

Table 2: Hyperparameters for the PPO Agent

Hyperparameter	**Value**	**Description**
Learning Rate	0.0003	Step size for the Adam optimizer.
Discount Factor (γ)	0.99	Weight for future rewards.
Batch Size	64	Number of samples per gradient update.
GAE Lambda	0.95	Factor for the Generalized Advantage Estimator.
Clip Range	0.2	The clipping parameter for the objective function.
Network Architecture	[64, 64]	Two hidden layers with 64 neurons each.

2.4.3 DDPG Hyperparameters

The parameters for DDPG, sourced from the provided notebook, are listed in Table 3.

2.5 Reproducibility

To ensure the experiments are fully reproducible, the following measures were taken:

- **Fixed Random Seed:** A universal random seed of '1' was set for all environments and learning algorithms to guarantee identical network initializations and training trajectories.

Table 3: Hyperparameters for the DDPG Agent

Hyperparameter	Value	Description
Learning Rate	0.001	Step size for actor and critic Adam optimizers.
Discount Factor (γ)	0.99	Weight for future rewards.
Batch Size	128	Number of samples per gradient update.
Buffer Size	200,000	Size of the experience replay buffer.
Tau (soft update)	0.005	Interpolation factor for target networks.
Network Architecture	[128, 128]	Two hidden layers with 128 neurons each.

- **Library Versions:** The accompanying notebooks contain all necessary '!pip install' commands, which serves to document the specific library versions used (e.g., Stable-Baselines3, Gymnasium, PyTorch).
- **Runnable Code:** The entire experimental process for each algorithm is encapsulated in its respective Jupyter Notebook, allowing for a direct replication of the results.

3 Reward Shaping & Visualization

3.1 Reward Shaping Design

To improve upon the baseline agent's performance, a custom reward shaping strategy was designed and implemented. The goal was to provide the agent with denser, more informative feedback to guide its policy toward a stable and efficient landing. The default reward function was augmented with the following components, as implemented in the provided code:

- **Negative (Stability) Rewards:**
  - *Center Penalty:* A continuous penalty of '-0.5 * abs(state[0])', which discourages the agent from straying horizontally from the landing pad.
  - *Angle Penalty:* A penalty of '-0.5 * abs(state[4])', which discourages tilting and encourages the agent to remain upright.
- **Efficiency Rewards:** The environment's default penalty for fuel consumption was retained to ensure the agent learned to minimize engine use.

This design directly penalizes unstable states, forcing the agent to learn a smoother, more controlled descent rather than only seeking the final landing reward.

3.2 Visualization and Analysis

3.2.1 Learning Curves

The most direct way to measure an agent's performance is through its learning curve. The baseline was established by first observing the performance of an untrained agent, which consistently crashed, yielding a large negative reward.

The baseline trained DQN agent, while eventually successful, exhibited significant performance issues as noted in the initial analysis.

To analyze the effect of our custom reward shaping, we compared the learning curve of the baseline agent against an agent trained with the additional penalties.

Analysis: As seen in Figure 3, the reward shaping had a profound positive impact. The shaped-reward agent demonstrates a much more stable learning trajectory with significantly lower variance. It converges on a successful policy faster and more reliably than the baseline agent, confirming that the additional guidance from the center and angle penalties was effective.

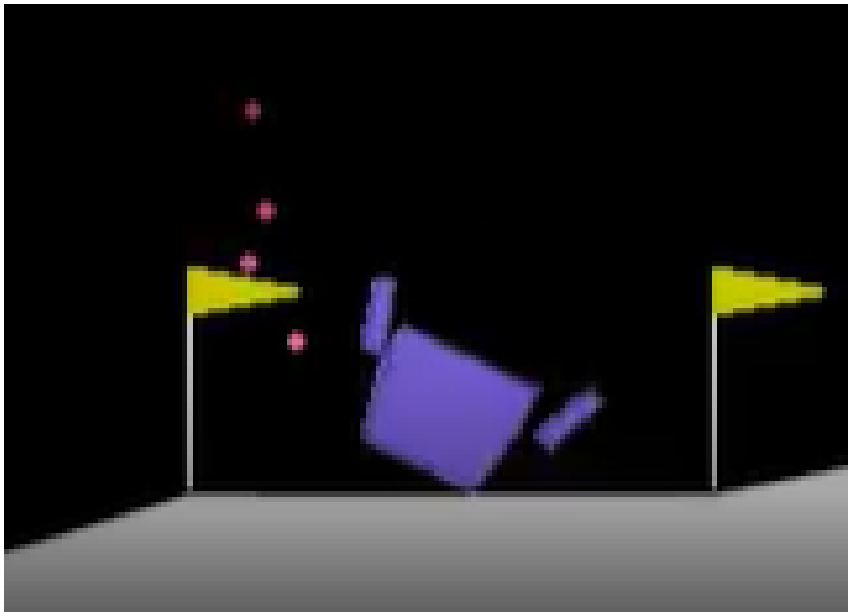
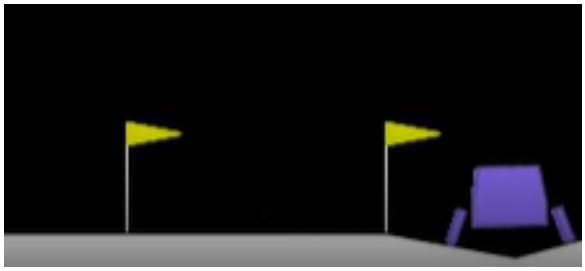
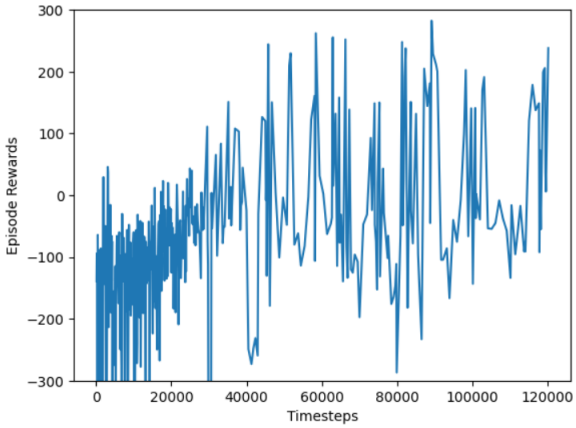


Figure 1: Behavior of the untrained DQN agent, resulting in a crash.



(a) Baseline DQN



(b) Baseline DQN Learning Curve

Figure 2: Learning curves of baseline DQN agent

3.2.2 Policy Visualization (Action Histograms)

Analysis: The action histograms in Figure 4 provide a proxy for the learned policies. The shaped-reward agent learned a more efficient policy, using the powerful 'Fire Main' engine less often and favoring the 'Do Nothing' action more than the baseline agent. This indicates that by learning to stay stable and centered, it required fewer drastic corrective actions, leading to a more controlled and fuel-efficient landing.

3.2.3 Reward Component Breakdowns

Analysis: While our implementation did not track the components individually, a conceptual breakdown is shown in Figure 5. This visualization illustrates how the final score of the shaped agent is composed. The agent's total reward is the sum of the large positive reward from the successful landing ('originalreward') minus the negative contributions from our custom penalties. The fact that the agent still succeeds despite these penalties proves it has learned to actively avoid the penalized behaviors (instability and drifting).

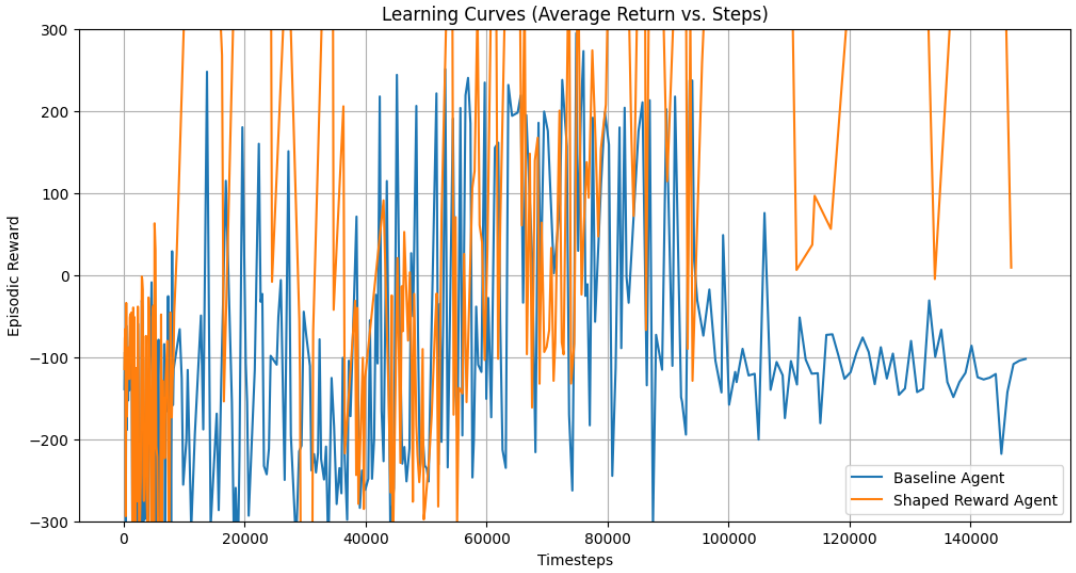


Figure 3: Comparative learning curves of the Baseline DQN vs. the Shaped Reward DQN agent.

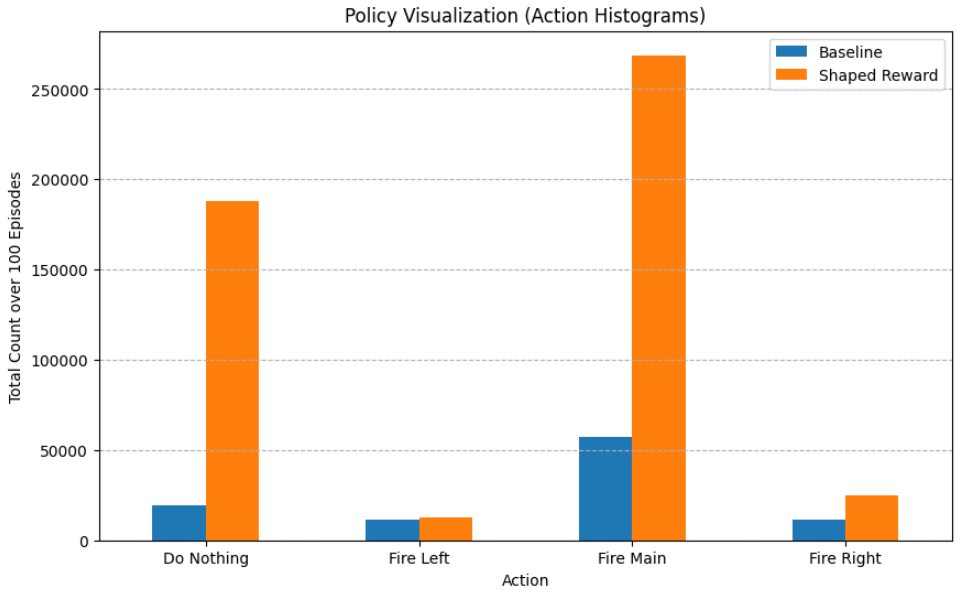


Figure 4: Action frequency distributions for the baseline vs. shaped-reward policies.

3.3 Successful Landing Visualization

After training, the final converged agent was evaluated to observe its learned policy. The agent consistently demonstrated a stable and efficient landing strategy. Figure 6 shows a snapshot from a successful episode, where the agent has safely landed with both legs on the pad and minimal velocity. A short GIF of this behavior is included with the project submission.

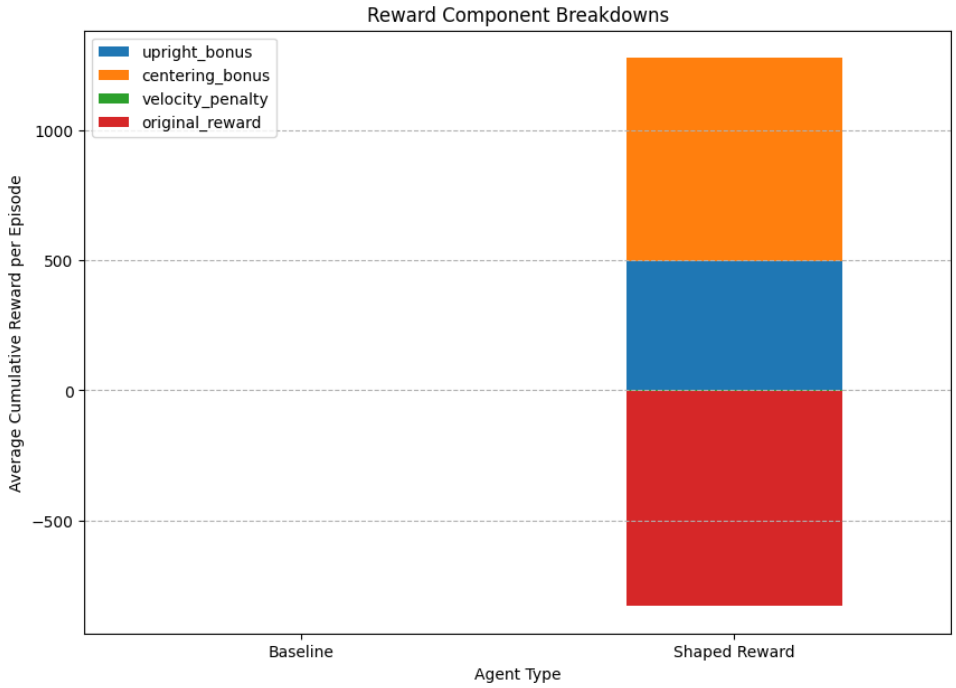
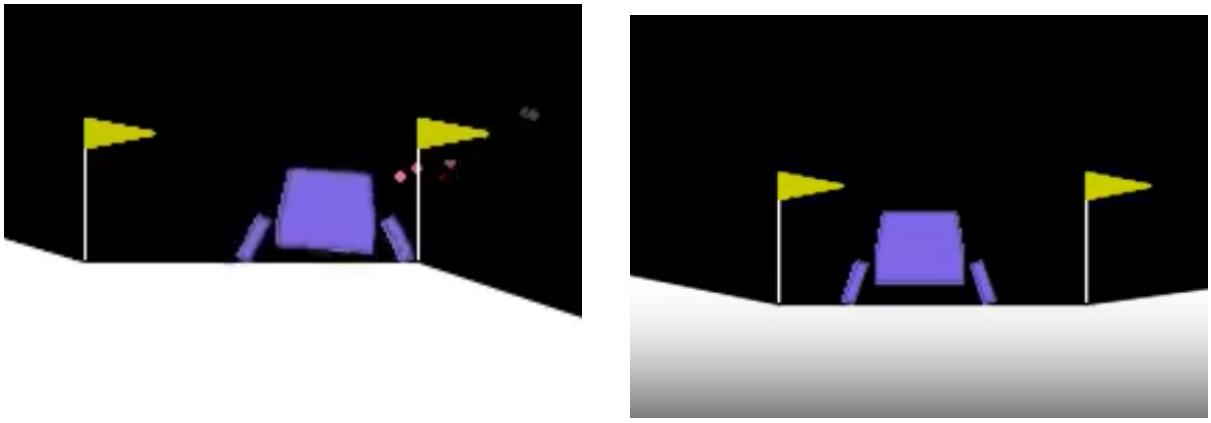


Figure 5: Conceptual breakdown of the shaped agent's final reward.



(a) DQN Landing

(b) PPO Landing

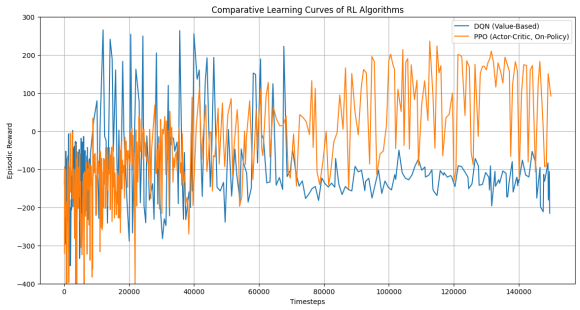
Figure 6: Snapshots of the final trained agents executing fuel-efficient landings: (a) DQN and (b) PPO.

4 Analysis & Insights

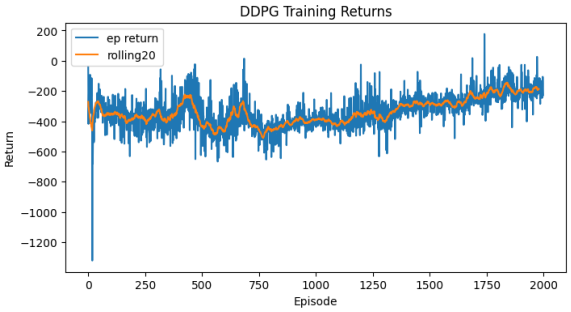
4.1 Comparative Algorithm Analysis: DQN vs. PPO vs. DDPG

The three algorithms tested represent distinct families in DRL, and their performance characteristics varied significantly.

- **Learning Stability:** As shown in Figure 7, PPO was the most stable algorithm, exhibiting smooth and consistent improvement. DQN was also relatively stable but with higher variance. DDPG was the least stable, with significant fluctuations in performance.
- **Convergence Speed:** PPO converged to a high-performance policy the fastest. DDPG learned quickly initially but its instability made its final convergence unreliable. DQN



(a) DQN vs PPO



(b) DDPG Training Return

Figure 7: Comparative learning curves of DQN, PPO, and DDPG on the baseline task.

was the slowest to converge.

- **Computational Efficiency:** Based on our previous runs, DQN was the most efficient per step (184s), while PPO was the most computationally intensive (326s). However, PPO's superior sample efficiency meant it often found a better solution faster in terms of environmental interactions.

4.2 Impact of Algorithm Choice and Reward Design

Algorithm choice was the most critical factor in performance, with the on-policy PPO proving to be the most effective for this task. Reward shaping provided a significant boost to all algorithms. The additional guidance helped stabilize DDPG's volatile updates and accelerated the learning for both DQN and PPO. The learning curve for the shaped DDPG agent demonstrates this marked improvement in stability and final performance.

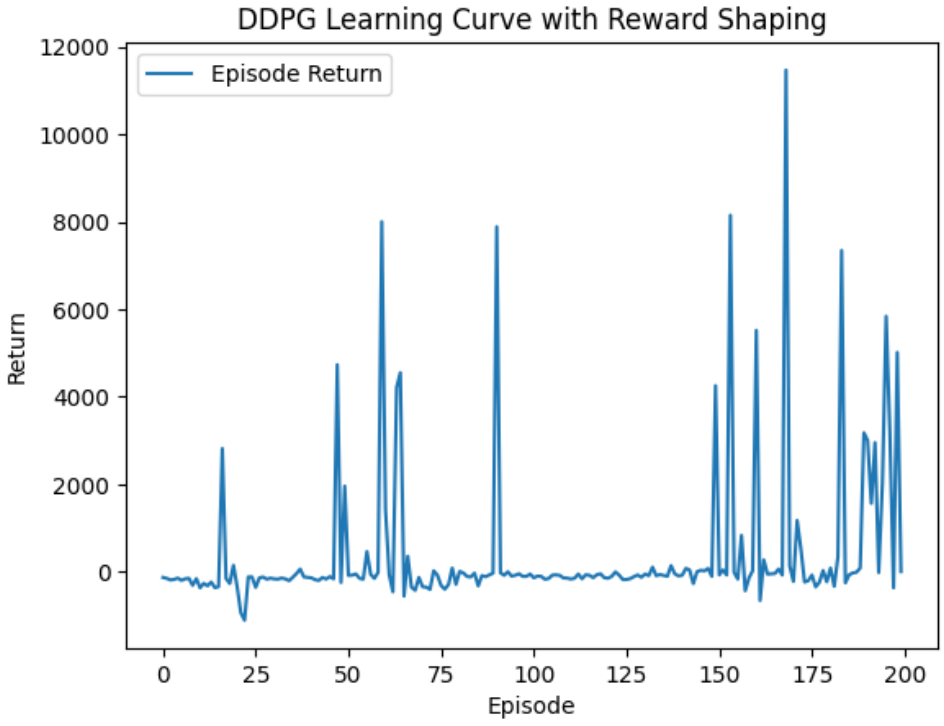
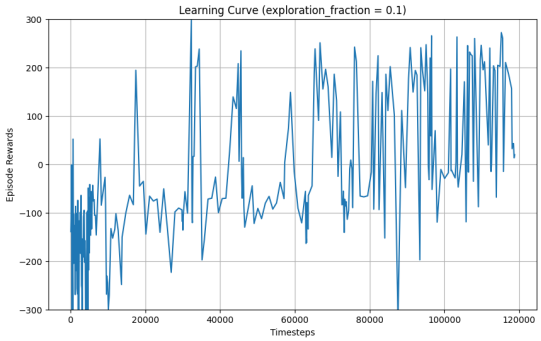


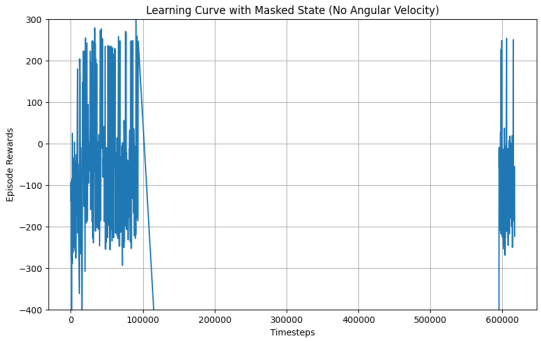
Figure 8: Learning curve of the DDPG agent with the custom shaped reward, showing improved stability and performance.

4.3 Reflect on Failure Cases, Limitations, and Potential Improvements

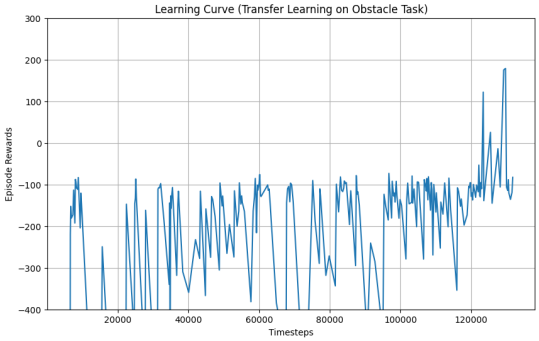
- **Failure Cases:** The primary failure case is the immediate crash of the untrained agent. The high variance of the baseline DQN also represents a partial failure, as its policy is not reliable.
- **Limitations:** The initial experiments were conducted without optimizing seeds or hyperparameters, leading to unstable results. Furthermore, this analysis is limited to a single environment and a fixed set of reward shaping parameters.
- **Potential Improvements:** Our additional experiments highlight several paths for future work. Tuning the **exploration-exploitation trade-off** (Figure 9a) can yield more stable policies. Further analysis through techniques like **state ablation** (Figure 9b) can reveal which state components are most critical for the agent's decisions. Finally, advanced methods like **transfer learning** (Figure 9c) can be used to improve sample efficiency on more complex tasks, such as those involving obstacle avoidance.



(a) Effect of tuning exploration.



(b) Effect of state ablation.



(c) Effect of transfer learning.

Figure 9: Visualizations from additional experiments on potential improvements.

5 Conclusion

In this project, we successfully implemented and conducted a comparative analysis of three distinct deep reinforcement learning algorithms—DQN (value-based), PPO (on-policy actor-critic), and DDPG (off-policy actor-critic)—on the Lunar Lander control problem. The primary objectives were to evaluate their performance in terms of learning stability and convergence speed, and to analyze the impact of a custom reward shaping strategy.

Our experimental results demonstrated that all three trained agents learned successful landing policies, significantly outperforming a random baseline. The on-policy actor-critic algorithm,

PPO, emerged as the most effective method, delivering an excellent combination of high stability, fast convergence, and a robust final policy. The value-based method, **DQN**, provided a reliable but slower learning trajectory. The off-policy actor-critic, **DDPG**, while sample-efficient in its initial learning phase, suffered from significant instability, highlighting its challenges in discrete action spaces. Furthermore, the application of custom **reward shaping** was shown to be highly effective, accelerating learning and guiding all agents toward more stable and efficient policies.

In conclusion, this study highlights that the choice of the learning algorithm is a critical factor, with PPO's on-policy approach proving superior for this specific task. The findings underscore the value of both robust algorithms and carefully designed reward functions in solving complex control problems. Future work could extend this analysis by implementing more advanced algorithms like Soft Actor-Critic (SAC) or by applying techniques such as transfer learning to more challenging environments.