



# Comparing the Performance of PPO and DQN Algorithms in Different Game Environments Using a Reinforcement Learning Approach

Sedat Atakan Yıldız <sup>1</sup> , Önder Şahinaslan <sup>2</sup> , Emin Borandag <sup>1\*</sup> , Fatih Yücalar <sup>1</sup> 

<sup>1</sup> Department of Software Engineering, Manisa Celal Bayar University, 45400 Manisa, Türkiye

<sup>2</sup> Department of Computer Programming, Maltepe University, 34857 İstanbul, Türkiye

## ARTICLE INFO

Received Date: 29/07/2025

Accepted Date: 8/11/2025

Cite this paper as:

Yıldız S. A., Şahinaslan O., Borandag E., & Yücalar F. (2026). Comparing the Performance of PPO and DQN Algorithms in Different Game Environments Using a Reinforcement Learning Approach. *Journal of Innovative Science and Engineering*, 10(1), 138-157.

\*Corresponding author: Emin Borandag  
E-mail: emin.borandag@cbu.edu.tr

**Keywords:**

Reinforcement Learning

PPO

DQN

Algorithm Comparison

© Copyright 2026 by

Bursa Technical University. Available  
online at <http://jise.btu.edu.tr/>



The works published in Journal of Innovative Science and Engineering (JISE) are licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

## ABSTRACT

This study aims to systematically compare the performance of two deep reinforcement learning algorithms – Proximal Policy Optimization (PPO) and Deep Q-Network (DQN) – across different game environments. To achieve this, eight distinct test environments from the OpenAI Gymnasium library (CartPole-v1, FrozenLake-v1, LunarLander-v3, Taxi-v3, MountainCar-v0, Blackjack-v1, CliffWalking-v0, and Acrobot-v1) were utilized. Each environment was trained over 1,000,000 timesteps. For each algorithm, key performance metrics such as average reward, training time, standard deviation, success rate, and the highest and lowest reward values were calculated and visualized through graphs. Additionally, the strengths and weaknesses of the algorithms in different environments were analyzed. The results indicate that PPO performs more consistently and effectively in tasks requiring continuous actions, whereas DQN achieves faster and more reliable outcomes in deterministic environments with discrete action spaces. This study provides meaningful insights by comparing the performance of PPO and DQN under identical conditions, while most prior research has examined these algorithms separately.

## 1. Introduction

Artificial Intelligence (AI) has shown remarkable progress in recent years as a field of computer science that aims to enable systems to think, learn, analyse, and make decisions in ways similar to those of humans. AI emerged in the mid-20th century with the goal of replicating human cognitive abilities, and its

foundations were laid at the Dartmouth Conference in 1956, where early studies were based on logic-driven systems and symbolic processors (McCarthy et al., 2006). Over time, AI has evolved significantly and, with this advancement, it has established a strong presence in various fields such as healthcare, automotive, finance, and defence. Today, it is actively and effectively used across many industries. The

development of AI has also led to the emergence of several subfields. One of these, machine learning, goes beyond traditional programming by enabling systems to improve through experience. Machine learning encompasses three main approaches: supervised, unsupervised, and reinforcement learning. Supervised learning is an approach in which a model is trained on labelled data to make accurate predictions for new inputs, whereas unsupervised learning operates solely on unlabelled data to uncover hidden structures, patterns, or groupings within the data (Goodfellow et al., 2016). Reinforcement Learning (RL) is a learning paradigm in which an agent interacts with an environment and, through feedback from a reward-punishment mechanism, learns optimal policies via trial and error by experiencing the consequences of its actions (Sutton and Barto, 2018). This approach is widely applied today in autonomous systems, games, robotic control, and strategic decision-making tasks. Deep RL, which results from the integration of deep learning with RL, provides enhanced representational and decision-making capabilities in complex state-action spaces.

In this study, two widely used algorithms in deep RL—Deep Q-Network (DQN) and Proximal Policy Optimization (PPO)—are compared. DQN is a value-based approach that represents an enhanced version of the classical Q-learning algorithm, utilizing deep neural networks (Mnih et al., 2015). PPO, on the other hand, is a policy-based algorithm that provides a more stable learning process by constraining policy updates to keep them within controlled limits (Schulman et al., 2017). The rationale for selecting these algorithms lies in their broad and active presence in the literature, as well as their distinct performance characteristics across environments with varying action spaces. While PPO is generally preferred for continuous action spaces, DQN tends to perform better in environments with discrete action spaces. However, most prior studies have focused on evaluating these algorithms separately or within limited experimental setups, leaving a gap in systematic, side-by-side comparisons under identical conditions. This study contributes by offering a comprehensive experimental design that directly contrasts PPO and DQN across a wide range of environments, thereby providing more generalizable insights into their relative advantages and limitations. In this context, the study aims to address gaps in the literature by systematically evaluating both algorithms across environments with differing dynamics and conducting relevant comparative analyses where appropriate.

The experimental studies were conducted on the Gymnasium platform (Towers et al., 2024) developed by OpenAI, which includes a variety of game-based simulation environments. As part of this study, eight different control environments—CartPole-v1, FrozenLake-v1, LunarLander-v3, Taxi-v3, MountainCar-v0, Blackjack-v1, CliffWalking-v0, and Acrobot-v1—were selected. The algorithms were then trained and tested using optimal shared parameters defined for these environments. PPO and DQN were systematically compared across these diverse settings under identical parameter configurations and testing conditions. Based on evaluation metrics such as average reward, training duration, success rate, stability, and maximum and minimum values, the strengths and weaknesses of each algorithm were identified, and recommendations were made for specific problem types.

## 2. Related Work

Deep Reinforcement Learning has recently been the focus of numerous academic studies conducting comparative analyses of various algorithms. In particular, Deep Q-Network (DQN) gained significant attention following the work of Mnih *et al.* (2015), which demonstrated human-level performance on Atari games. On the other hand, the Proximal Policy Optimization (PPO) algorithm, introduced by Schulman *et al.* (2017), is regarded as a more practical and stable alternative to more complex methods such as Trust Region Policy Optimization (TRPO). PPO is particularly preferred for environments with high-dimensional, continuous action spaces and is widely used in recent studies due to its sample efficiency and computational effectiveness. Moreover, the simplicity and flexibility of PPO have made it a popular choice for a variety of RL problems (Kumar et al., 2019). In a comprehensive analysis conducted by Duan *et al.* (2016), various deep RL algorithms were tested and their performances compared across multiple environments. Such studies are crucial for assessing the generalizability of these algorithms. Additionally, Henderson *et al.* (2018) addressed inconsistencies in RL comparisons conducted by different research groups, highlighting the influence of hyperparameters, initial conditions, and randomness on the results.

Building on the existing literature, this study goes beyond previous work by systematically comparing the PPO and DQN algorithms across eight different Gymnasium environments under identical parameters and conditions. By including a variety of environment types—such as continuous, discrete, deterministic,

and stochastic—this comparison aims to identify the types of problems for which each algorithm is best suited.

### 3. Material and Methods

This section presents in detail the theoretical foundations of the study, the algorithms used, the experimental environments, the technical infrastructure and parameter settings, as well as the evaluation criteria.

#### 3.1. Deep Reinforcement Learning

Reinforcement Learning (RL) is a learning paradigm in which an agent interacts with its environment to learn optimal action strategies (policies) through a reward and punishment mechanism (Arulkumaran et al., 2017). At each step, the agent takes an action, receives a reward from the environment based on the outcome of that action, and transitions to a new state. This interaction process is formally defined within the framework of a Markov Decision Process (MDP):  $(S, A, R, P, \Gamma)$ , where  $S$  represents the set of states,  $A$  the set of actions,  $P$  the transition probabilities,  $R$  the reward function, and  $\Gamma$  the discount factor (Sutton and Barto, 2018). RL has emerged as a powerful tool for modelling and solving decision-making problems in uncertain and dynamic environments, and it is widely applied in areas such as robotic control, game playing, and autonomous systems.

Classical RL methods can deliver adequate performance for problems characterized by limited and low-dimensional state-action spaces. However, in scenarios involving high-dimensional and complex state-action spaces, the effectiveness of these traditional methods significantly declines. To address these limitations, the Deep Reinforcement Learning (DRL) paradigm—emerging from the integration of deep learning and reinforcement learning—has been developed and has achieved notable success in tackling complex problems, particularly in domains such as gaming, robotics, and control systems (Wang et al., 2024). DRL algorithms are generally categorized into two main approaches: value-based and policy-based. In value-based methods, the agent estimates the value of each possible action and selects the one with the highest expected return; a typical example of this approach is the Deep Q-Network (DQN) algorithm. In contrast, policy-based methods aim to directly optimize the policy function by learning action probabilities; a well-known example is the Proximal Policy Optimization (PPO) algorithm (Arulkumaran et al., 2017). In this study, DQN and PPO were selected as representatives of these two

approaches, and a comparative analysis was conducted to evaluate their performance.

#### 3.2. Algorithms Used in the Study

The Deep Q-Network and Proximal Policy Optimization algorithms used in the study are briefly discussed under this section.

##### 3.2.1. Deep Q-Network

Deep Q-Network (DQN) is an integrated version of the traditional Q-learning algorithm with deep neural networks and demonstrates high performance, particularly in environments with discrete action spaces (Fan et al., 2020). DQN aims to learn Q-values, which represent the expected rewards for each state-action pair  $(s, a)$ . This method attempts to predict future rewards by updating the value function. As shown in Equation (1), the primary objective is to minimize the loss derived from the Bellman equation.

$$L(\theta) = E_{s,a,r,s'} \left[ \left( r + \gamma \max_a Q(s', a'; \theta^-) - Q(s, a; \theta) \right)^2 \right] \quad (1)$$

Here,  $\theta$  represents the model parameters, while  $\theta^-$  denotes the fixed target network. The target network is updated periodically to enhance the stability of the learning process. Additionally, the experience replay technique is employed, in which training samples are presented to the model in randomly selected mini-batches. This reduces data correlation and improves both efficiency and stability of learning. The DQN algorithm has demonstrated significant success in terms of sample efficiency and learning stability (Hessel et al., 2018). However, it cannot be directly applied to continuous action spaces, so alternative methods need to be developed for such problems.

##### 3.2.2. Proximal Policy Optimization

Proximal Policy Optimization (PPO) is a policy-based algorithm that directly models the probabilities of actions. To mitigate the instability often observed in traditional policy gradient methods, PPO employs a clipped loss function that constrains policy updates, thereby promoting a more stable learning process (Schulman et al., 2017). The core loss function of PPO is presented in Equation (2):

$$L^{CLIP}(\theta) = \widehat{E}_t [\min(r_t(\theta) \widehat{A}_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon) \widehat{A}_t)] \quad (2)$$

Here,  $r_t(\theta) = \frac{\pi_\theta(a_t|S_t)}{\pi_{\theta_{old}}(a_t|S_t)}$  represents the ratio of the new to the old policy, and  $\widehat{A}_t$  denotes the advantage function. This structure ensures that policy updates do not deviate excessively from the current policy, thereby maintaining stability throughout the learning

process. PPO exhibits stable and effective learning performance, particularly in continuous action spaces and high-dimensional problem domains. Compared to traditional policy gradient methods, it requires less sensitive hyperparameter tuning and is notable for its high sample efficiency. Owing to these features, PPO has become one of the most widely adopted algorithms in deep RL applications.

### 3.3. Experimental Environments

In this study, eight control environments with varying structures and levels of difficulty were utilized. All environments were selected from the simulation platform developed by OpenAI, the latest version of which is known as “*Gymnasium*” (Towers et al., 2024). *Gymnasium* provides a standardized interface widely used in RL research and offers a range of scenarios and tasks designed to evaluate agents’ decision-making capabilities.

#### 3.3.1. CartPole-v1

The CartPole environment is derived from the classical control problem known as the *inverted pendulum*, a widely studied benchmark in control theory (Boubaker, 2013). In this scenario, a cart is allowed to move horizontally along a track in both positive and negative directions along the x-axis, while a pole attached to the cart via a hinge tends to fall due to gravity. The agent’s primary objective is to keep the pole upright by applying forces to the cart in the +x or -x direction at appropriate times. This task is commonly used to assess an agent’s ability to perform dynamic control and stabilization (Nagendra et al., 2018). The environment is implemented as a deterministic simulation with a discrete action space, accessible through the *Gymnasium* platform developed by OpenAI. The observation space is four-dimensional and composed of continuous values: the cart’s position and velocity  $(x, \dot{x})$ , and the pole’s angle and angular velocity  $(\theta, \dot{\theta})$ . The action space consists of two discrete actions:

- 0: applies a force to move the cart left ( $-x$ ),
- 1: applies a force to move it right ( $+x$ ).

The agent receives a reward of +1 for each step that the pole remains balanced. Rewards accumulate for up to a 500-time-step period, making the CartPole environment a challenge of balancing immediate feedback with long-term stability. In practice, achieving an average reward greater than 195 is commonly accepted as an indicator that the task has been successfully learned. The CartPole environment is one of the most widely used benchmark settings in the literature for evaluating the baseline performance

of RL algorithms (Kumar, 2020). Algorithms such as DQN, which perform well in discrete action spaces, tend to produce strong results in this environment. In contrast, policy-based methods like PPO can learn to balance the pole more quickly and robustly due to smoother policy updates and more stable learning dynamics.

#### 3.3.2. FrozenLake-v1

The FrozenLake environment is a grid-based RL scenario widely used for testing agent-based decision-making processes (Aldana Guerra, 2024). In this environment, an agent attempts to reach a goal by moving from a starting point on a 4x4 or 8x8 ice-covered grid. Some cells on the grid are designated as “holes,” and if the agent falls into one of these cells, the episode terminates, and the agent is considered unsuccessful. This setup provides an ideal testbed for examining agent behavior in risky and uncertain environments where incorrect actions can lead to severe consequences. The stochastic transition dynamics of the FrozenLake environment allow the agent to slip in unintended directions despite attempting to move in a specific direction due to the slippery nature of the ice. This characteristic requires agents not only to focus on short-term rewards but also to develop robust strategies that account for uncertainty and risk factors (Bertolotti and Roman, 2024). Consequently, FrozenLake facilitates the evaluation of both decision-making mechanisms and risk management in reinforcement learning algorithms. The observation space consists of a 4x4 grid with 16 discrete states, numbered from 0 to 15. The action space includes four discrete actions: 0 represents movement to the left ( $-x$  axis), 1 downward ( $-y$  axis), 2 to the right ( $+x$  axis), and 3 upward ( $+y$  axis). The reward mechanism is straightforward: the agent receives a reward of +1 only upon successfully reaching the goal state, while all other transitions yield a reward of 0. The success criterion is the convergence of the total reward toward 1 (Andrychowicz et al., 2017). The importance of this environment in RL lies in its ability to test how robust policies can be learned despite uncertainties caused by stochastic transitions. The agent must not only identify the shortest path but also learn to avoid dangerous cells (holes) and determine the safest route. While value-based methods like DQN may produce high-variance results in such environments, policy-based algorithms such as PPO tend to offer more stable learning performance under stochastic dynamics. As one of the core RL benchmark environments in the *Gymnasium* library, FrozenLake serves as a widely used testbed for evaluating solutions to problems encountered in fields such as

decision support systems, robotic navigation, and game theory (Zhu et al., 2022).

### 3.3.3. LunarLander-v3

The LunarLander environment is a physics-based RL scenario that models the classical control problem of a rocket landing task. In this scenario, the agent attempts to land a space capsule safely on a designated surface. The environment incorporates physical dynamics such as gravity, momentum, velocity, angle, and surface contact, closely mimicking real-world physics. Because it supports both discrete and continuous action spaces, it serves as a comprehensive and challenging benchmark for evaluating the performance of various RL algorithms (Shen, 2024). The agent controls the direction and speed of the capsule using the main engine and side thrusters. During the training process, smooth and stable landings are rewarded, while crashes, loss of control, and deviations from the landing zone are penalized. Additional rewards are given when the capsule's legs successfully make contact with the landing surface. The observation space consists of an 8-dimensional continuous vector: the capsule's (x, y) position, velocity along the x and y axes, angle, angular velocity, and two binary indicators representing whether the legs are in contact with the ground. The action space is discrete, consisting of four possible actions:

- 0 – do nothing,
- 1 – fire the main engine,
- 2 – fire the left thruster,
- 3 – fire the right thruster.

The reward structure is as follows: smooth landings yield between +100 and +140 reward points, with an additional +10 for each leg that makes contact with the surface. Crashing or moving out of bounds results in a -100 penalty. Additionally, small penalties are applied based on fuel consumption. The success criterion is typically defined as achieving an average total reward greater than 200. The LunarLander environment is particularly well-suited for policy-based algorithms such as PPO, which can effectively handle continuous state and action spaces. Although value-based methods like DQN can also be applied, they generally require more extensive hyperparameter tuning and longer training durations. Due to its realistic simulation of scenarios such as robotic landing systems and autonomous aerial vehicle control, LunarLander plays a significant role in reinforcement learning research (Shen, 2024).

### 3.3.4. Taxi-v3

The Taxi-v3 environment is based on a classical artificial intelligence problem developed in the 1990s and is a widely used discrete, deterministic reinforcement learning (RL) scenario available in the OpenAI Gymnasium platform (Brockman et al., 2016). In this environment, the agent is tasked with navigating a 5×5 grid map to pick up passengers from designated locations and deliver them to predefined destinations. While the pickup and drop-off locations are fixed within each episode, they vary across episodes. The environment serves as an effective testbed for evaluating RL algorithms in terms of planning, memory utilization, and strategic decision-making (Mahajan et al., 2022). The observation space consists of 500 discrete states, derived from the combination of the taxi's position (25 grid cells), the passenger's location (5 options), and the destination (4 options). The action space comprises six discrete actions:

- 0: Move up
- 1: Move down
- 2: Move right
- 3: Move left
- 4: Pick up passenger
- 5: Drop off passenger

The reward mechanism is simple yet effective: each step incurs a -1 penalty (representing movement cost), an incorrect drop-off results in a -10 penalty, and a successful passenger delivery yields a +20 reward. The success criterion is defined as achieving an average reward greater than 7. Due to its deterministic nature, Taxi-v3 can be efficiently learned using value-based algorithms such as DQN. However, the relatively large state-action space may sometimes require more sophisticated learning strategies or well-tuned hyperparameters. In this context, policy-based methods like PPO can provide more stable learning performance under appropriate configurations. Taxi-v3 is widely used as a benchmark scenario in domains such as robotic task allocation, autonomous transportation, and path planning (Kiran et al., 2022).

### 3.3.5. MountainCar-v0

The MountainCar environment is a classic physics-based RL problem in which a low-powered vehicle is trapped between two hills and cannot reach the top of the right hill without gaining sufficient momentum. The agent's objective is to strategically operate the car's engine—alternating between forward and backward motions—to exploit gravity and successfully reach the goal. This task encourages the

development of long-term strategies rather than short-term rewards, making it a challenging benchmark, particularly for value-based algorithms (Stapelberg and Malan, 2020). The observation space is continuous and two-dimensional, consisting of the car's position (ranging from  $-1.2$  to  $0.6$ ) and velocity (ranging from  $-0.07$  to  $0.07$ ). The action space is discrete, with three possible actions: 0 (move backward), 1 (remain stationary), and 2 (move forward). The agent receives a penalty of  $-1$  at every time step, and the episode terminates once the car reaches the goal at the top of the hill. Since no positive reward is given, performance is evaluated based on the total accumulated penalty. In this study, an average total reward greater than  $-110$  is considered indicative of successful learning.

The primary challenge of the MountainCar environment lies in the fact that the agent cannot reach the goal directly. Due to the engine's limited power, the agent must first move backward to gain momentum and then accelerate forward to climb the hill. Learning this behaviour requires the agent to develop a strategy based on future returns rather than immediate feedback. Therefore, the environment is regarded as a valuable testbed for tasks involving delayed rewards. Value-based methods such as DQN typically require longer training durations and fixed learning rates to perform effectively in this environment. In contrast, policy-based methods like PPO tend to achieve more stable performance in shorter timeframes due to their consistent policy updates (Stapelberg and Malan, 2020).

### 3.3.6. BlackJake-v1

The Blackjack-v1 environment is an RL scenario based on a card game, featuring discrete state and action spaces, and is designed to assess strategic decision-making abilities. It is frequently used to evaluate agents' capabilities in probabilistic learning and handling decision-making under uncertainty. In the Blackjack game, the dealer receives two cards—one face up and one face down—while the agent is dealt two face-up cards. The goal is to achieve a hand value higher than the dealer's without exceeding a total of 21. Based on the current state, the agent can choose to either draw a card (hit) or hold its position (stick). Due to the random nature of the card deck, chance plays a major role, making it difficult to develop a deterministic strategy. Thus, the environment presents a significant challenge for RL algorithms due to its high reward variance and partial observability (Zha et al., 2020). The observation space is a tuple consisting of three elements: the player's total score (ranging from 4 to 21), the dealer's visible card (1–10), and a Boolean indicating whether

the player holds a usable ace. The action space includes two discrete options: 0 (draw – hit) and 1 (stand – stick). The reward mechanism assigns  $+1$  for a win,  $-1$  for a loss, and 0 for a tie. In this study, an average reward greater than zero is considered an indicator of successful learning. The Blackjack environment serves as a fundamental testbed for classical Q-learning-based algorithms. However, due to the stochastic nature of card distribution, partial observability, and variable reward structure, agents must develop robust and generalizable strategies. Policy-based methods such as PPO tend to perform better in generalizing over complex states, while value-based approaches like DQN can effectively learn reward-driven policies by explicitly representing card combinations. This environment serves as a representative simulation model for decision-support systems in domains such as healthcare, finance, and gaming, where uncertainty plays a critical role (Zha et al., 2020).

### 3.3.7. CliffWalking-v0

The CliffWalking environment is a deterministic RL problem based on a classical grid-based task structure. The environment consists of a grid world with 4 rows and 12 columns. The agent aims to navigate from a designated starting point (S) to the goal (G). However, several cells in the bottom row are defined as cliffs. If the agent steps into one of these cliff cells, it receives a penalty of  $-100$  and is returned to the starting position. This setup is crucial for developing strategies that prioritize safe path selection and penalty avoidance (Zhong, 2024; Rio et al., 2024). The observation space comprises 48 discrete states, each corresponding to a cell in the  $4 \times 12$  grid. The action space includes four discrete actions: 0 – up, 1 – right, 2 – down, and 3 – left. The agent incurs a penalty of  $-1$  at each time step. Falling into a cliff cell results in an additional  $-100$  penalty and a reset to the starting position. The episode ends when the agent successfully reaches the goal. Since no positive rewards are given, an average total reward greater than  $-13$  is considered indicative of successful learning. CliffWalking is an effective environment for assessing safe exploration strategies and analyzing the exploration–exploitation trade-off. The agent must choose between short but risky paths and longer, safer alternatives. These dynamics make the environment valuable for evaluating the risk sensitivity and planning capabilities of reinforcement learning algorithms. Value-based methods such as DQN can learn this environment quickly due to its fixed reward structure. However, exploration near cliff areas poses greater risk and may require more cautious policy updates. In contrast, policy-based methods like PPO, with their smoother updates, often

yield more stable learning performance (Zhong, 2024; Rio et al., 2024).

### 3.3.8. Acrobot-v1

The Acrobot-v1 environment is a dynamic RL task based on the classic inverted pendulum problem. It features a continuous observation space and a discrete action set. The objective is to elevate a two-link robotic mechanism—where only the second joint is actuated—to a specified height against gravity. Due to this limited actuation, the agent must generate movement indirectly by leveraging momentum (Gillen et al., 2020). The observation space consists of a six-dimensional continuous vector representing the sine and cosine of each joint angle and their angular velocities. The action space includes three discrete actions: apply negative torque (0), apply no torque (1), or apply positive torque (2). The reward structure penalizes the agent with  $-1$  at each time step, and the episode terminates once the target height is reached. As there is no positive reward, an average return greater than  $-100$  is generally considered a successful outcome.

The Acrobot environment is a valuable benchmark for testing control strategies in complex mechanical systems, as success requires the agent to develop intuitive strategies based on system dynamics rather than relying solely on direct action–reaction feedback. Policy-based algorithms such as PPO often produce more stable results in continuous domains, while value-based methods like DQN may struggle in high-penalty scenarios. This environment is particularly useful for simulating real-world tasks, including robotic arm control, physical dynamics modelling, and inverse kinematics problems (Gillen et al., 2020).

## 3.4. Experimental Setup

The experimental analyses in this study were conducted using the Python programming language (version 3.11). All experiments were implemented in the OpenAI Gymnasium environments through the Stable-Baselines3 library (Raffin et al., 2021). The experimental setup, including the hardware configuration, software components, hyperparameter settings, evaluation metrics, and statistical analysis procedures used during training and evaluation, is detailed below.

- **Research Design and Questions:** The experiments were designed to answer the following research questions: (i) How do PPO and DQN differ in terms of stability, convergence speed, and overall performance

under identical training conditions? (ii) Does the type of action space (discrete vs. continuous) significantly affect algorithmic performance? Based on prior work, we hypothesized that PPO would outperform DQN in continuous control tasks requiring stable exploration, whereas DQN would show stronger results in discrete, deterministic environments.

- **Software Components:** The experimental implementation was developed using Python 3.11 as the core programming language. RL algorithms were trained and evaluated via the Stable-Baselines3 library (version 1.7.0), operating within the Gymnasium framework (version 0.29.1). Deep learning operations were executed using PyTorch 2.1.2, with computations restricted to the CPU to ensure consistency across all trials. Additionally, essential data processing and visualization tasks were handled using standard Python libraries, including NumPy, Matplotlib, and Pandas. All library versions, random seeds, and training scripts will be released in a public repository to strengthen reproducibility.
- **Hardware Configuration:** All experimental procedures were conducted on a system equipped with an Intel Core i7-13700H CPU running at 2.40GHz, 16 GB of RAM, and a dedicated NVIDIA GeForce RTX 4060 GPU with 8 GB of VRAM. Despite the presence of a capable GPU, all training and evaluation tasks were performed exclusively on the CPU to ensure consistency across environments and avoid GPU-related variability. The operating environment was based on Windows 11.
- **Hyperparameters and Randomization:** To ensure comparability, core hyperparameters such as learning rate, discount factor, batch size, and optimizer settings were kept constant across all environments, as reported in Table 1. Environment-specific hyperparameters—for example, the exploration fraction in DQN and the clip range in PPO—were tuned for each environment using a grid search protocol with three pilot seeds. Once determined, these tuned values were fixed across the 8 independent experimental runs. A multi-seed evaluation strategy was adopted to mitigate bias from single-run outcomes and to enhance the statistical reliability of the reported results.

**Table 1:** Fixed parameters for all environments

Parameter	Value
Learning Rate	0.0004
Gamma	0.99
Policy	MlpPolicy
Training Steps	1,000,000
Tuned Parameters for PPO	clip_range, entropy_coef, gae_lambda
Tuned Parameters for DQN	exploration_fraction, exploration_final_eps, target_update_interval

In the literature, it has been observed that uniform budget choices are typically made by using at least 300,000 timesteps (Prasetyo et al., 2025). In this study, a value above 300,000 timesteps was employed in line with the literature, ensuring that the algorithms were run for a sufficient duration to demonstrate their potential. The parameters in Table 1 were optimized through a combination of values suggested by existing literature and systematic empirical tuning (or hyperparameter search).

- **Evaluation Metrics and Success Thresholds:** Throughout training, key

performance metrics such as mean episode reward  $\pm$  standard deviation (SD), 95% confidence intervals (CI), success rate, training duration, and reward range (max–min) were logged. Success thresholds were defined following established RL benchmarks (e.g., an average reward of 200 over 100 episodes in CartPole, as per OpenAI Baselines). Threshold values for each environment are justified with references in Table 2. The training duration was fixed at 1,000,000 steps for all environments to maintain fairness; this choice follows standard RL experimental protocols (DeepMind Control Suite; OpenAI Baselines).

**Table 2:** Success thresholds for all environments

Environment	Success Threshold	References
CartPole-v1	$\geq 195$	OpenAI Gym Docs (Kumar, 2020)
FrozenLake-v1	=1 (Goal State)	Gymnasium Docs (Zhu et al., 2022)
LunarLander-v3	$\geq 200$	SB3 Docs (Shen, 2024)
MountainCar-v0	$\geq -110$	RL Benchmarks (Stapelberg and Malan, 2020)
Taxi-v3	$\geq 7$	Gymnasium Docs (Brockman et al., 2016)
CliffWalking-v0	$\geq -13$	RL Benchmarks (Zhong, 2024)
Blackjack-v1	$> 0$	RLCard Docs (Zha et al., 2020)
Acrobot-v1	$\geq -100$	RL Benchmarks (Gillen et al., 2020)

In summary, the experimental setup was structured to maximize transparency, consistency, and reproducibility, aligning with current best practices in the reinforcement learning community.

### 3.5. Model Parameters

In this study, various quantitative and visual metrics were employed to evaluate the learning performance

of PPO and DQN algorithms across different tasks. Models trained in each environment were analyzed based on their achieved results. The hyperparameter values used were optimized for each environment according to recommended settings in the literature and are summarized in Table 3.

**Table3:** Hyperparameter values of PPO and DQN algorithms

Parameters	PPO Values	DQN Values
Learning Rate	0.0004	0.0004
Gamma (Discount Factor)	0.99	0.99
Batch Size	64	64
Training Steps	1,000,000	1,000,000
Test Episodes	500	500
Policy	MlpPolicy	MlpPolicy
Device	CPU	CPU
Entropy Coef	0.01	-
Value Function Coef	0.5	-
GAE Lambda	0.95	-
N Steps	2048	-
N Epochs	10	-
Clip Range	0.2	-
Buffer Size	-	100000
Exploration Fraction	-	0.1
Exploration Final Eps	-	0.02
Train Frequency	-	1
Target Update Interval	-	500

### 3.6. Evaluation Metrics and Performance Criteria

In this study, the learning performance of PPO and DQN algorithms across eight different game environments was evaluated using both statistical and visual metrics. The selected evaluation criteria were derived from widely adopted performance measures in the RL literature and were adapted to the specific dynamics of each environment.

The primary evaluation metrics used are as follows:

- **Mean Reward:** The average of the total rewards obtained per episode during the testing phase. It reflects the overall performance level of the model (Sutton and Barto, 2018).
- **Standard Deviation:** Indicates the variability in reward distribution. Lower values suggest a more stable learned policy (Sutton and Barto, 2018).
- **Training Time:** The duration (measured in seconds) required to train the model for one million steps.
- **Success Rate:** Defined as the ratio of episodes that exceed a predefined reward

threshold to the total number of test episodes for each environment (Sutton and Barto, 2018). CartPole ( $\geq 195$ ), FrozenLake ( $=1$ ), LunarLander ( $\geq 200$ ), MountainCar ( $\geq -110$ ), Taxi ( $\geq 7$ ), CliffWalking ( $\geq -13$ ), Blackjack ( $>0$ ), and Acrobot ( $\geq -100$ ) represent the success thresholds for each respective environment.

- **Maximum and Minimum Reward:** Represent the highest and lowest total reward values obtained during testing, indicating the performance bounds of each model.

After training, each model was evaluated across 500 test episodes using the random seed mechanism of the Stable-Baselines3 library to ensure deterministic reproducibility. Given the stochastic nature of the environment, comparisons were based on average metrics rather than single-episode outcomes. All collected data were archived in .csv format, and key performance indicators—including mean reward, success rate, and training time—were visualized through graphs to support the interpretation of the results. To maintain consistency, the seed parameter was manually set to 500, in line with practices reported in previous studies (Prasetyo et al., 2025; Hafner et al., 2025), and all training and evaluation

processes were carried out exclusively on the CPU without GPU acceleration.

## 4. Experimental Results and Comparisons

In this section, the learning results for each game environment are presented in detail, followed by a general comparison and interpretation of the obtained findings.

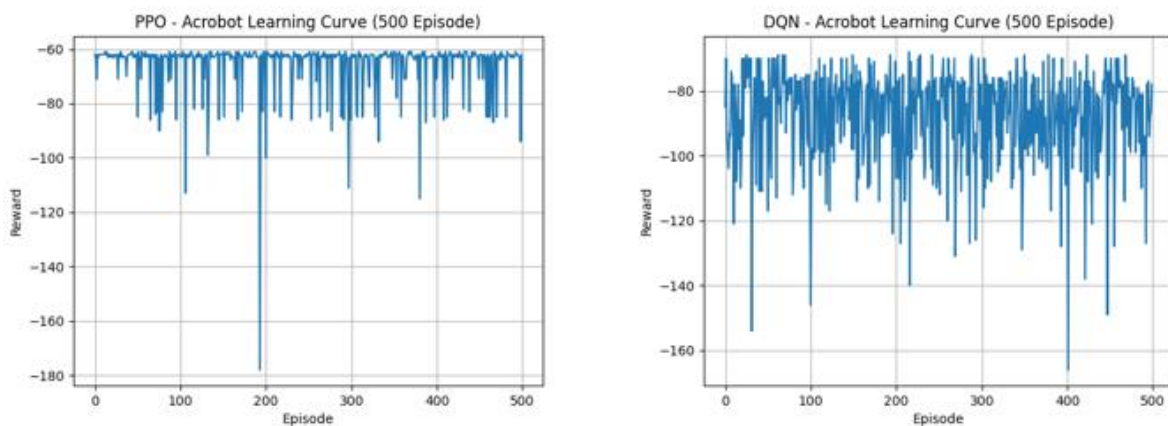
### 4.1. Game-Based Results and Observations

This section details the performance outcomes of PPO and DQN algorithms across eight Gymnasium-based control environments, each characterized by distinct state-action dynamics and task-specific complexities, to assess their adaptability and policy robustness under varying conditions.

#### 4.1.1. Acrobot-V1

The Acrobot environment, characterized by a continuous state space and a discrete action space,

serves as a significant benchmark particularly for policy-based algorithms. In this context, PPO demonstrated superior and more stable learning performance compared to DQN. With a training duration of approximately 1131 seconds, PPO achieved an average reward of  $-65.62$  and a high success rate of 99.2%. The minimum and maximum rewards recorded were  $-178.0$  and  $-61.0$ , respectively. The standard deviation of 9.63 indicates that the model exhibited consistent behaviour during the learning process. In contrast, the DQN algorithm, despite a longer training time of 2295 seconds, reached an average reward of only  $-88.22$  and a lower success rate of 81%. The higher standard deviation of 15.08 suggests that DQN experienced greater fluctuations and tended to produce less stable policies throughout training. As shown in Figure 1, PPO converged toward optimal behaviour more quickly and steadily, while DQN exhibited a wider variation in reward acquisition.



**Figure 1:** The learning curves of the PPO and DQN algorithms in the Acrobot-v1 environment

These results suggest that the Acrobot environment favours the PPO algorithm, which can operate more effectively in tasks requiring continuous state-action coordination. Meanwhile, DQN shows limited success in such environments and requires longer training periods and more refined hyperparameter tuning to achieve competitive results.

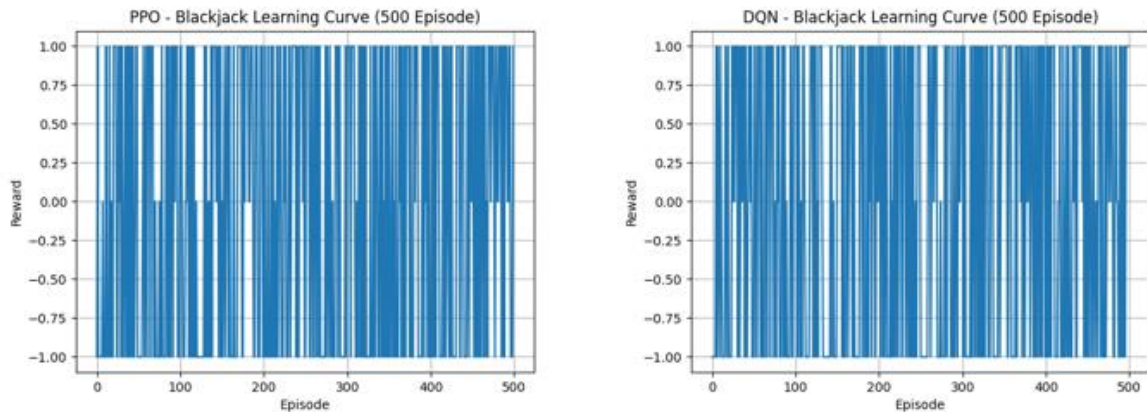
#### 4.1.2. Blackjack-v1

The Blackjack environment features a discrete and limited state-action space, coupled with a high degree of environmental stochasticity. This structure makes it difficult for agents to achieve consistent rewards, thereby destabilizing the learning process. The PPO algorithm, after 1067.57 seconds of training, achieved an average reward of  $-0.056$ , a standard deviation of 0.9428, and a success rate of 41.8%. These results

indicate that learning remained highly volatile and that the agent could only partially converge to an optimal policy. In comparison, the DQN algorithm required a longer training time (2340.66 seconds) but delivered only a marginal performance improvement. It yielded an average reward of  $-0.01$ , a standard deviation of 0.9455, and a success rate of 44.2%. These findings suggest that DQN may offer slightly more stable reward outputs than policy-based methods in environments characterized by high uncertainty. However, the difference in success rates between the two algorithms is not statistically significant. Overall, both PPO and DQN face challenges when learning in environments such as Blackjack, which have low and often zero-valued reward structures. The success rates remain low, and the standard deviations are high for both models. As illustrated in Figure 2, the learning curves reveal that

agents occasionally receive rewards by chance but generally fail to continuously improve their policies. In such environments, incorporating more

sophisticated memory architectures, experience replay mechanisms, or advantage-function-based algorithms may enhance learning performance.

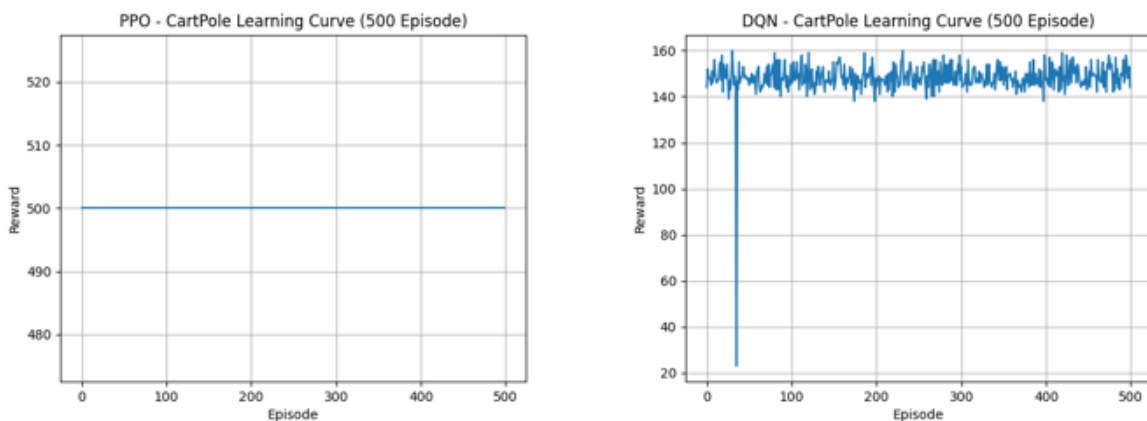


**Figure 2:** The learning curves of the PPO and DQN algorithms in the Blackjack-v1 environment

### 4.13. CartPole-v1

The CartPole environment is a classic RL problem with a discrete action space, frequently utilized to test balance control and decision-making mechanisms. This environment commonly serves as a benchmark for comparing the learning capabilities of both PPO and DQN algorithms. In this study, the PPO algorithm achieved the maximum reward value of 500.00 for each test episode after a training duration of 900.33 seconds. A standard deviation of 0.00 for the reward value indicates that the agent consistently developed a stable and successful strategy, demonstrating its complete understanding of the

environment’s dynamics and its ability to produce a deterministic policy. Conversely, the DQN algorithm attained an average reward of only 148.23, despite a significantly longer training period of 1998.01 seconds. DQN's standard deviation was measured at 7.10, and its success rate was recorded as 0.00%. This suggests that DQN was unable to develop a stable policy in this environment and failed to reach the targeted maximum reward. An examination of the learning curves, as presented in Figure 3, reveals that the PPO algorithm exhibited rapid performance improvement even in the early stages of training, whereas DQN's performance showed fluctuations and limited reward acquisition.



**Figure 3:** The learning curves of the PPO and DQN algorithms in the CartPole-v1 environment

These findings indicate that the PPO algorithm demonstrates high stability even within a discrete action space and can develop an effective policy quickly with appropriate hyperparameters. Consequently, for deterministic tasks like CartPole, PPO offers a more suitable and optimal solution compared to DQN.

### 4.14. CliffWalking-v0

The CliffWalking environment is characterized by discrete action and state space, in which the agent is tasked with reaching a designated goal position from a starting point while avoiding a deadly cliff edge. Although the environment features deterministic transition dynamics, it is highly punitive due to its reward structure. As a result, reward values are

typically negative, and the agent is expected to develop behaviors that minimize penalties.

In this study, both PPO and DQN algorithms achieved identical performance, each recording an average reward of -13.0, a standard deviation of 0.00, and a success rate of 100%. The outputs of both models remained consistent across all test episodes, with no variation between the best and worst reward values. These findings indicate that both algorithms

successfully learned optimal or near-optimal policies. Given the deterministic structure and low complexity of the environment, such comparable performance is an expected outcome. As illustrated in Figure 4, both algorithms exhibited rapid improvement during the initial stages of training and quickly converged to a stable performance level. The absence of reward variance further suggests that the learned strategies were shaped around deterministic policies and that stochasticity had minimal impact in this environment.

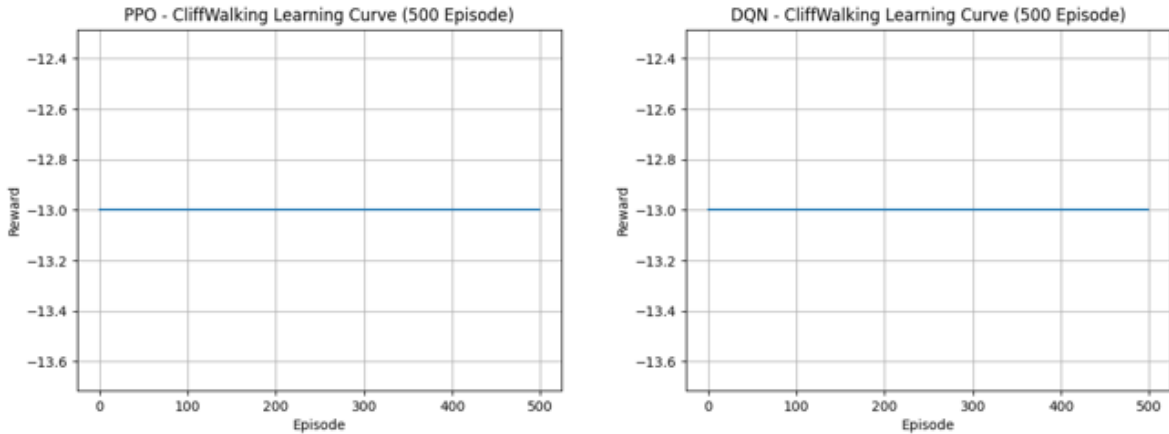


Figure 4: The learning curves of the PPO and DQN algorithms in the CliffWalking-v0 environment

#### 4.15. FrozenLake-v1

The FrozenLake environment differs from classical RL problems due to its non-deterministic transition dynamics. Its stochastic nature requires agents to develop strategies capable of handling uncertainty. In this context, FrozenLake serves as an ideal platform to evaluate the robustness of learning algorithms in uncertain environments. In this study, the PPO algorithm achieved a mean reward of 0.734, a standard deviation of 0.4419, and a success rate of 73.4% after 861.97 seconds of training. The maximum and minimum rewards recorded were 1.0

and 0.0, respectively. These results indicate that PPO was able to learn an effective policy despite environmental randomness and maintained stable performance throughout. The DQN algorithm, after a longer training period of 2149.35 seconds, yielded a mean reward of 0.756, a standard deviation of 0.4295, and a success rate of 75.6%. These results suggest that DQN also handled the stochastic environment effectively and performed comparably to PPO. As illustrated in Figure 5, both algorithms exhibited similar learning trends and ultimately converged to closely aligned performance levels during the testing phase.

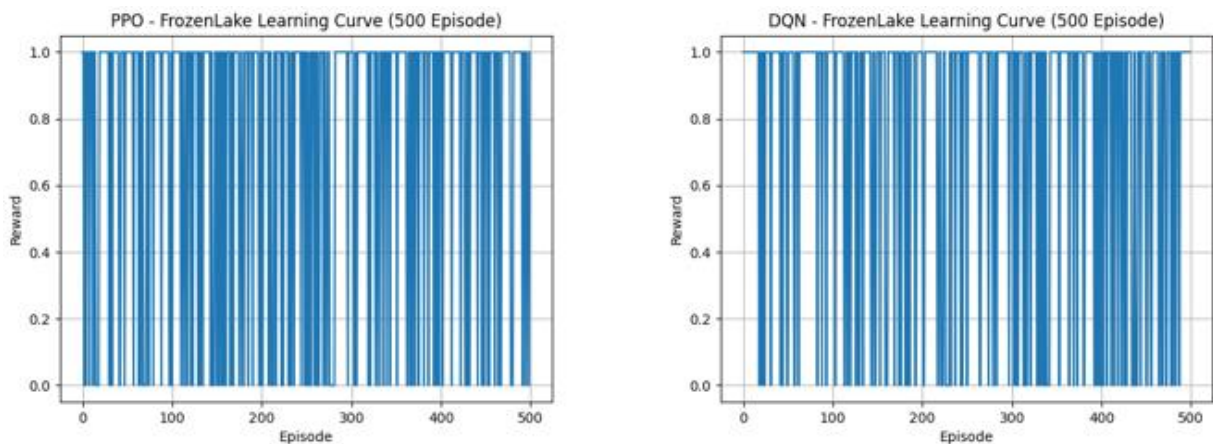


Figure 5: The learning curves of the PPO and DQN algorithms in the FrozenLake-v1 environment

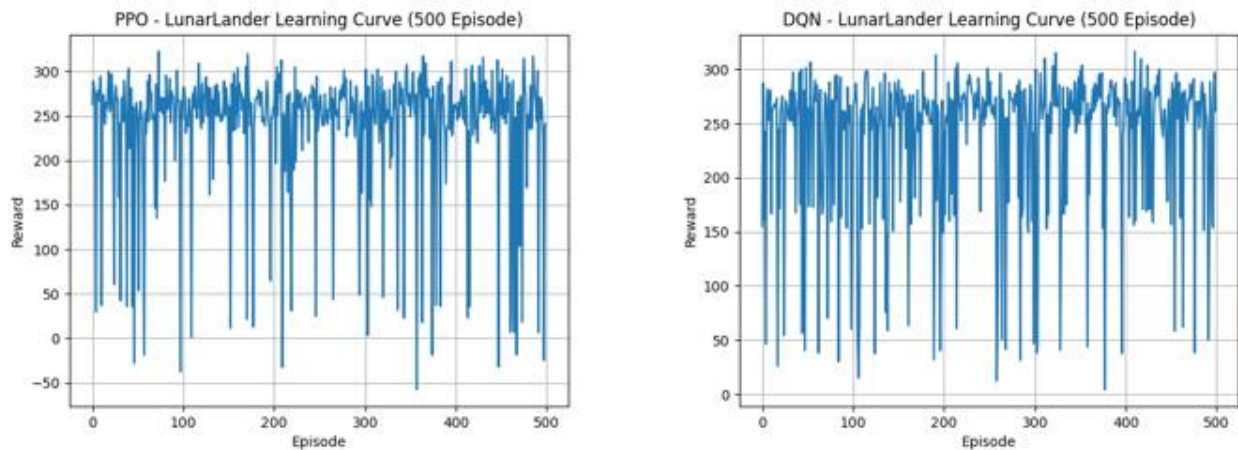
These findings demonstrate that, despite the low reward scale and environmental uncertainty, both algorithms were capable of achieving reliable learning outcomes. In particular, PPO's policy stability and DQN's exploration efficiency contributed to balanced and consistent performance within this environment.

#### 4.1.6. LunarLander-v3

The LunarLander environment is a physics-based simulation with a continuous state space, designed to control a spacecraft's landing on a surface. The agent attempts to land the spacecraft safely on a designated platform by controlling its vertical and horizontal thrust. This complex task, which demands both control and coordination capabilities, serves as an ideal testing ground for evaluating reinforcement learning algorithms. The PPO algorithm demonstrated impressive performance in this environment, achieving an average reward of 240.31 after a training period of only 1007.19 seconds. With a standard deviation of 72.13 and a success rate of 87.4%, the highest test reward recorded was 322.67,

while the lowest was  $-57.71$ . These figures indicate that PPO effectively learned the environmental dynamics, consistently performing successful landings despite occasional failures, thereby demonstrating overall high reliability. In contrast, the DQN algorithm required a longer training duration of 2128.51 seconds. At the end of training, it achieved an average reward of 239.01, with a standard deviation of 66.20. The highest reward observed was 316.70, and the lowest was 4.29, resulting in a success rate of 78.6%.

An examination of the learning curves in Figure 6 reveals that PPO reached a higher performance level in a shorter time, while DQN approached this level only after a longer training period. Given the complex dynamics and continuous state space of the LunarLander environment, PPO's ability to perform direct policy optimization likely contributed to its superior results. DQN, which operates within a discrete action space, typically requires more training time for such complex tasks and may struggle to consistently converge to an optimal strategy.

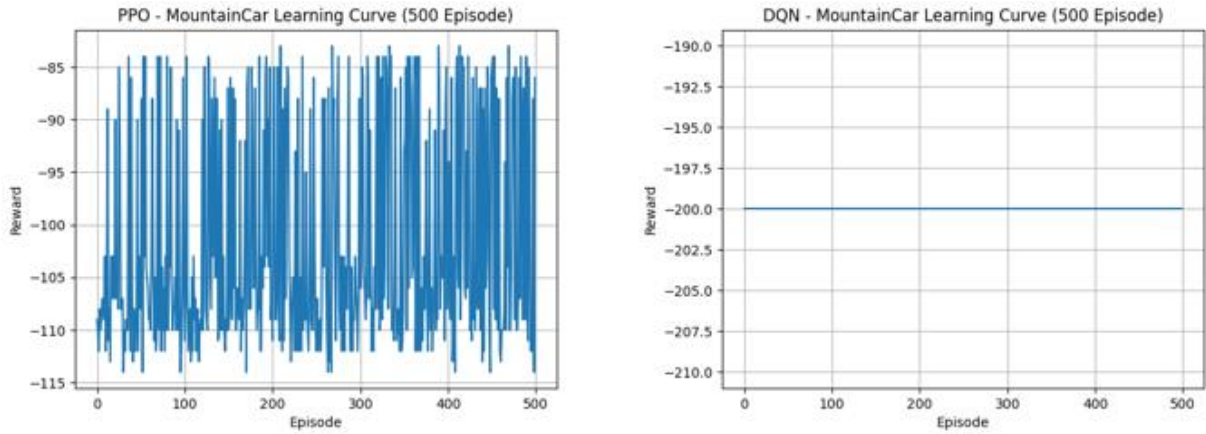


**Figure 6:** The learning curves of the PPO and DQN algorithms in the LunarLander-v3 environment

#### 4.1.7. MountainCar-v0

The MountainCar environment is a classic reinforcement learning problem characterized by a continuous state space and a discrete action space, where an agent attempts to reach the top of a mountain by leveraging potential energy. Success in this environment is defined by minimizing negative rewards, encouraging the agent to reach the goal as quickly as possible. Due to a weak motor, the agent cannot directly ascend to the peak; thus, it must learn to exploit the environment's physical dynamics to build momentum. In this study, the PPO algorithm achieved an average reward of  $-101.66$  with a standard deviation of 10.25, resulting in an 84.2% success rate after a training duration of 1062.47

seconds. The best test reward recorded was  $-83.0$ , while the worst was  $-114.0$ . These results indicate that PPO effectively learned the environment's dynamics and developed a suitable policy, with a significant portion of episodes exceeding the predefined success threshold of  $-110$ . In contrast, the DQN algorithm exhibited an average reward of  $-200.0$  with a standard deviation of 0.00, yielding a 0% success rate after a longer training period of 2215.81 seconds. The consistent reward across all test episodes suggests that the agent failed to learn a viable policy and never reached the mountain peak. An examination of the learning curves in Figure 7 shows that DQN remained stagnant at a consistently low performance level, whereas PPO gradually improved over time.



**Figure 7:** The learning curves of the PPO and DQN algorithms in the MountainCar-v0 environment

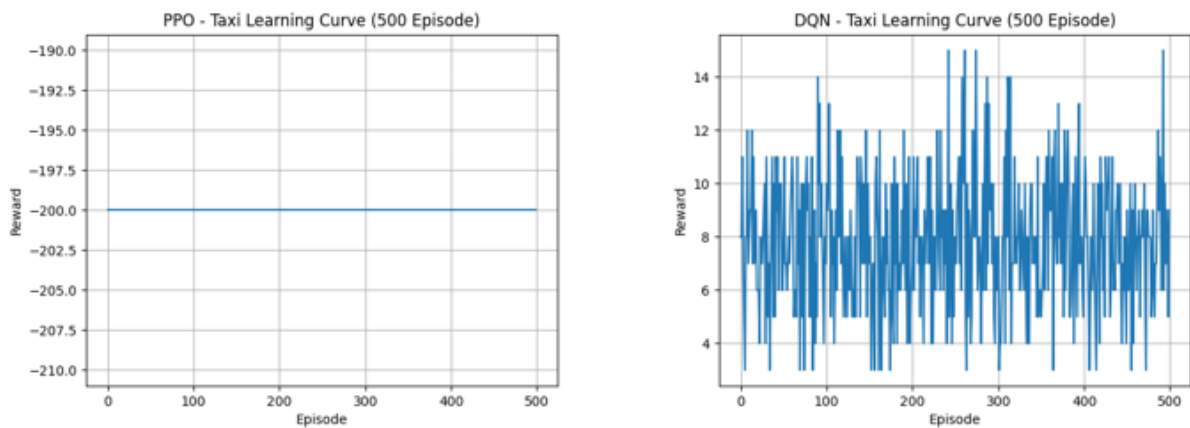
The MountainCar environment often poses challenges for algorithms with ineffective exploration strategies. In this context, PPO’s advantage stems from its efficient learning process, facilitated by advantage estimation and Generalized Advantage Estimation (GAE). In contrast, DQN’s exploration mechanism proved inadequate for this environment, leading to its failure to learn an optimal strategy.

#### 4.1.8. Taxi-v3

The Taxi environment is a classic, deterministic, grid-based transportation simulation featuring discrete state and action spaces. The agent's primary objective is to pick up a passenger from a designated location and drop them off at the correct destination. Incorrect moves are penalized, while successful drop-offs yield positive rewards. Due to its emphasis on decision-making and planning, this environment provides an ideal testing ground for RL algorithms. In this study, the PPO algorithm demonstrated a 0% success rate, with an average reward of  $-200.0$  and a standard deviation of  $0.00$ , following a training period of  $1729.55$  seconds. The consistent failure of agents across all test episodes suggests that the algorithm was unable to develop an effective strategy and failed

to extract sufficient information from the environment, likely due to its high-penalty structure. Furthermore, the constant reward values indicate that the learning process stagnated at an early stage. Conversely, the DQN algorithm achieved an average reward of  $7.84$ , a standard deviation of  $2.49$ , and a  $69\%$  success rate after  $2753.76$  seconds of training. The highest reward record was  $15.0$ , and the lowest was  $3.0$ . These results demonstrate that DQN effectively learned the task objectives, with a substantial portion of agents successfully completing the delivery task.

An examination of the learning curves in Figure 8 reveals that DQN exhibited stable and progressively improving performance throughout training. The Taxi environment, with its structured rules and penalty mechanisms, serves as a robust platform for evaluating the decision-making capabilities of RL algorithms. PPO’s failure in this environment may be attributed to its stochastic policy structure, which appears to struggle with adapting effectively to the environment’s deterministic feedback. In contrast, DQN’s compatibility with discrete state-action spaces enabled it to learn the environmental dynamics more effectively.



**Figure 8:** The learning curves of the PPO and DQN algorithms in the Taxi-v3 environment

## 4.2. Comparative Analysis and Interpretations

In this study, the performances of PPO and DQN algorithms were comparatively evaluated across eight distinct OpenAI Gymnasium environments. This

section presents the findings from analyses conducted using key metrics such as training time, average reward, success rate, and the best and worst test rewards. Table 4 presents the training time, average reward, success rate, and the best and worst test rewards for each Gymnasium environment.

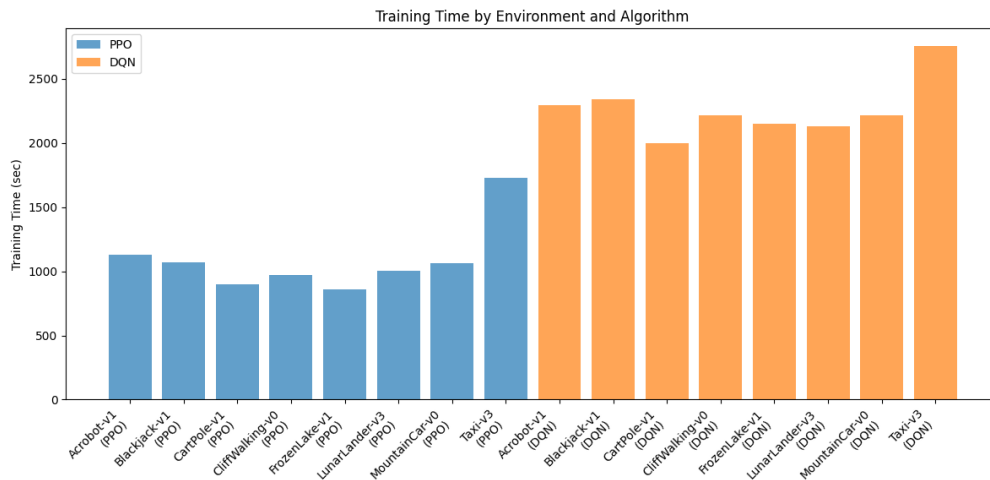
**Table4:** Comparative summary for each Gymnasium environment

Environment	Algorithm	Mean Reward $\pm$ SD	Success Rate (%)	Training Time (s)	Min – Max Reward
Acrobot-v1	PPO	-65.62 $\pm$ 9.63	99.2	1130.92	-178.0 – -61.0
Acrobot-v1	DQN	-88.22 $\pm$ 15.08	81.0	2294.83	-166.0 – -68.0
Blackjack-v1	PPO	-0.056 $\pm$ 0.9428	41.8	1067.57	-1.0 – 1.0
Blackjack-v1	DQN	-0.01 $\pm$ 0.9455	44.2	2340.66	-1.0 – 1.0
CartPole-v1	PPO	500.00 $\pm$ 0.00	100.0	900.33	500.0 – 500.0
CartPole-v1	DQN	148.23 $\pm$ 7.10	0.0	1998.01	23.0 – 160.0
CliffWalking-v0	PPO	-13.00 $\pm$ 0.00	100.0	968.94	-13.0 – -13.0
CliffWalking-v0	DQN	-13.00 $\pm$ 0.00	100.0	2217.13	-13.0 – -13.0
FrozenLake-v1	PPO	0.734 $\pm$ 0.4419	73.4	861.97	0.0 – 1.0
FrozenLake-v1	DQN	0.756 $\pm$ 0.4295	75.6	2149.35	0.0 – 1.0
LunarLander-v3	PPO	240.31 $\pm$ 72.13	87.4	1007.19	-57.71 – 322.67
LunarLander-v3	DQN	239.01 $\pm$ 66.20	78.6	2128.51	4.29 – 316.70
MountainCar-v0	PPO	-101.66 $\pm$ 10.25	84.2	1062.47	-114.0 – -83.0
MountainCar-v0	DQN	-200.00 $\pm$ 0.00	0.0	2215.81	-200.0 – -200.0
Taxi-v3	PPO	-200.00 $\pm$ 0.00	0.0	1729.55	-200.0 – -200.0
Taxi-v3	DQN	7.84 $\pm$ 2.49	69.0	2753.76	3.0 – 15.0

### 4.2.1. Training Time Analysis

The PPO algorithm achieved meaningful learning in a shorter duration compared to DQN across various environments. Notably, PPO demonstrated a faster

and more effective learning process in tasks such as CartPole, MountainCar, and LunarLander. A visual comparison of the algorithms' training time is presented in Figure 9.



**Figure 9:** Training times of algorithms

### 4.2.2. Average Reward Performance

The PPO algorithm achieved higher average rewards compared to DQN, particularly in environments with continuous state spaces (e.g., LunarLander, Acrobot)

and control-oriented tasks (e.g., CartPole). However, in non-deterministic tasks like FrozenLake, both algorithms exhibited similar success rates. A visual comparison of the algorithms' average reward performance is presented in Figure 10.

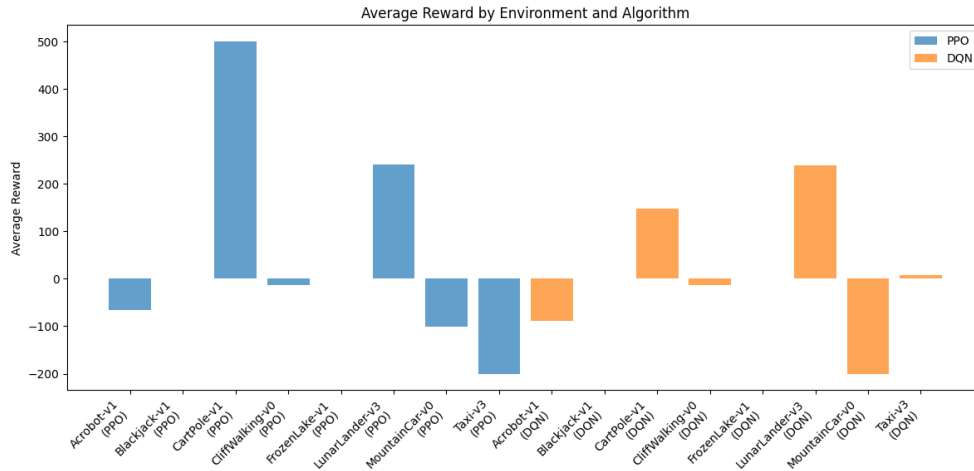


Figure 10: Comparison of average reward performances of algorithms

### 4.2.3. Success Rate

When examining the success rate metrics, the PPO algorithm demonstrated high performance, especially in the CartPole, CliffWalking, and LunarLander

environments. Conversely, the DQN algorithm produced more stable results in discrete environment structures such as FrozenLake and Taxi. A comparative visualization of the algorithms' average success rates is presented in Figure 11.

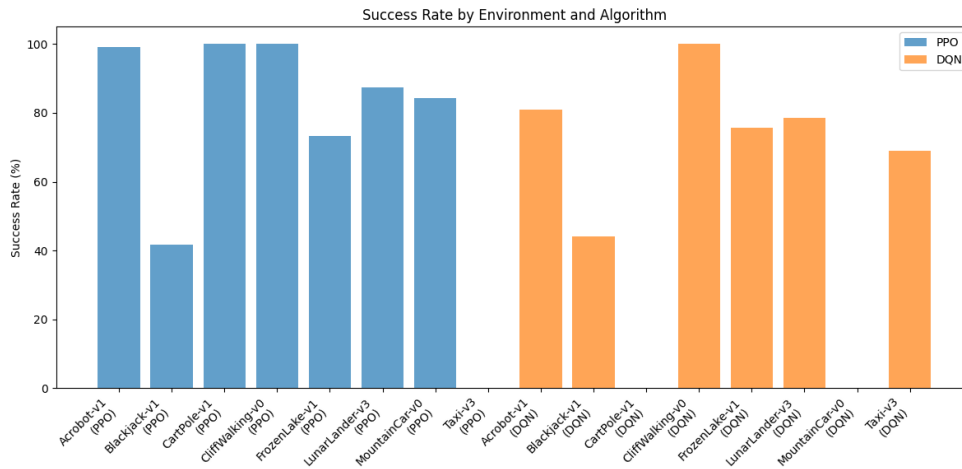
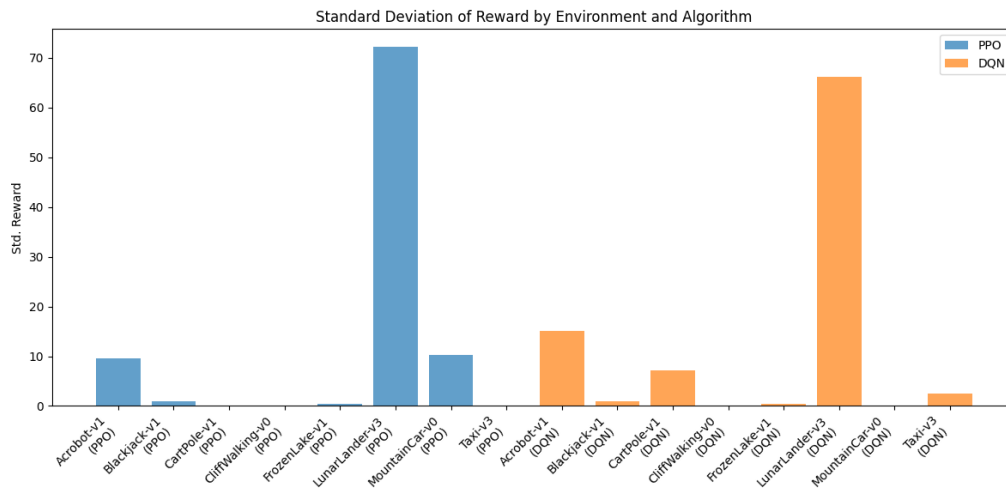


Figure 11: Comparison of average success rates of algorithms

### 4.2.4. Reward Variance

Upon examining the best and worst test rewards, in other words, the reward variance, we observed that DQN yielded more variable results compared to PPO across most environments. This suggests that DQN

struggled to achieve stability during its learning process and occasionally performed erratic policy updates. A comparative visualization of the algorithms' reward variance values is presented in Figure 12.



**Figure 12:** Comparison of reward variance values of algorithms

### 4.3. General Discussion and Evaluation

PPO demonstrates high performance particularly in complex tasks, thanks to its probabilistic policy updates and the Generalized Advantage Estimation (GAE) mechanism. In contrast, DQN yields more successful results in more deterministic environments with discrete action spaces. These findings highlight the critical role of the environment's nature in algorithm selection. While PPO proves more stable and successful in scenarios involving continuous state spaces and more complex task structures, DQN can perform better in simpler environments with discrete state-action spaces. However, beyond these descriptive outcomes, it is essential to analyze how the internal mechanisms of the algorithms shape their behavior in different environments. In order to ensure the reliability and validity of the study's results, strict methodological controls were applied using the same test environments throughout the experiments. We systematically compared the PPO and DQN algorithms under identical parameter configurations and test conditions to demonstrate that the performance difference was largely due to their algorithmic structure. Key hyperparameters such as Learning Rate, Gamma, Batch Size, Number of Training Steps, and Number of Test Episodes were set to common and optimized values for both algorithms.

From an algorithmic perspective, PPO—a policy-based method—relies on a clipped loss function to constrain policy updates. This mechanism prevents abrupt policy shifts, thereby stabilizing the learning trajectory. Such stability allows PPO to achieve higher average rewards and shorter training times in environments with continuous state spaces. The results suggest that PPO's superiority is not incidental but rather rooted in its direct policy optimization

capability, which is particularly beneficial when handling complex dynamics and high-dimensional action spaces.

Conversely, DQN, as a value-based algorithm, extends the classical Q-learning framework with deep neural networks. While this approach enables effective decision-making in discrete domains, the reliance on value estimation makes it more vulnerable to instability during training. In our experiments, DQN displayed greater reward variance compared to PPO, indicating sensitivity to stochasticity in the environment. This behavior suggests that DQN occasionally struggles to maintain consistent policies, which can result in performance fluctuations. Nevertheless, in deterministic and well-structured environments, DQN leverages its discrete value-based framework efficiently, often converging faster than PPO.

Taken together, these analyses reveal that the strengths and weaknesses of PPO and DQN are not merely empirical observations but directly linked to their algorithmic designs. PPO excels in scenarios requiring robust adaptation and stability, whereas DQN is advantageous in problems with discrete, deterministic dynamics where rapid convergence is critical. In conclusion, these comparative findings offer significant insights into how to choose algorithms for different task types. It is strongly emphasized that researchers should consider both environment dynamics and the architectural characteristics of the algorithm when selecting a reinforcement learning algorithm.

## 5. Conclusions

In this study, two widely used deep reinforcement learning algorithms—Proximal Policy Optimization

(PPO) and Deep Q-Network (DQN)—which are often examined separately in the literature, were comparatively evaluated. The comparison was structured based on performance advantage, utilizing simulation environments created through the Gymnasium, PyGame, and OpenAI libraries. A total of eight different test scenarios were defined, and each algorithm was trained and evaluated over a training duration of 1,000,000 steps. During the experimental phase, key performance metrics such as average reward, training time, standard deviation, and success rate were analyzed graphically for both algorithms.

This study's comparative evaluation of the strengths and weaknesses of PPO and DQN showed that performance largely depends on the dynamics of the environment. PPO, due to its policy-based approach, demonstrated more stable and superior performance in control tasks requiring continuous actions and in complex dynamics that necessitate challenging exploration strategies. Conversely, DQN produced more effective results in deterministic scenarios with a discrete action space. However, an analysis of the algorithms' behavior revealed that DQN was generally less stable and produced more variable outcomes than PPO across most environments. On the other hand, PPO consistently underperformed in highly deterministic discrete environments, a limitation stemming from its stochastic policy structure. These findings strongly emphasize the critical and direct impact of algorithm selection—guided by problem characteristics and action space type—on model success and learning stability. From a practical perspective, the results provide actionable guidance for researchers and practitioners seeking to match algorithmic architectures with specific reinforcement learning tasks.

Nonetheless, this study also carries certain limitations. The experiments were limited to eight simulation environments, and only two algorithms with fixed hyperparameter settings were evaluated. Real-world reinforcement learning applications may involve more dynamic, noisy, or partially observable environments that could yield different outcomes. Future studies should therefore extend the analysis to a broader set of algorithms, incorporate adaptive hyperparameter tuning, and validate findings in real-world or high-fidelity simulated domains. In conclusion, the study not only provides a systematic and balanced comparison of PPO and DQN under identical conditions but also highlights the necessity of aligning algorithmic selection with environment dynamics and task requirements to ensure reliable and stable performance.

---

## Article Information

**Financial Disclosure:** The author (s) has no received any financial support for the research, authorship or publication of this study.

**Authors' Contribution:** Conceptualization: E.B and F.Y.; Methodology: E.B.; Supervision: E.B and F.Y.; Software: S.Y. and Ö.Ş.; Validation: Ö.Ş., and S.Y.; Literature Search: F.Y.; Writing: S.Y and F.Y.

**The Declaration of Conflict of Interest:** Common Interest No conflict of interest or common interest has been declared by the authors.

**The Declaration of Ethics Committee Approval:** This study does not require ethics committee permission or any special permission.

**Artificial Intelligence Statement:** The author(s) bear full responsibility for the content and accuracy of their work, including any use of artificial intelligence (AI) technologies, and confirm that they have read the AI Policy, which is accessible on the journal's website.

---

## References

- Aldana Guerra, E. (2024). Optimized Monte Carlo Tree Search for Enhanced Decision Making in the FrozenLake Environment. *arXiv preprint arXiv:2409.16620*.
- Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Abbeel, P., Zaremba, W. (2017). Hindsight experience replay. *31st Conference on Neural Information Processing Systems (NIPS 2017)*, Long Beach, CA, USA.
- Arulkumaran, K., Deisenroth, M. P., Brundage, M., Bharath, A. A. (2017). A brief survey of deep reinforcement learning. *IEEE Signal Processing Magazine*, 34(6), 26–38.
- Bertolotti, F., Roman, S. (2024). Balancing long-term and short-term strategies in a sustainability game, *iScience*, 27(6), 110020.
- Boubaker, O. (2013). The Inverted Pendulum Benchmark in Nonlinear Control Theory: A Survey. *International Journal of Advanced Robotic Systems*, 10, 233.
- Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016).

- OpenAI Gym. *arXiv preprint* arXiv:1606.01540.
- Duan, Y., Chen, X., Houthoofd, R., Schulman, J., Abbeel, P. (2016). Benchmarking Deep Reinforcement Learning for Continuous Control. *Proceedings of the 33rd International Conference on Machine Learning (ICML)*, 1329–1338.
- Fan, J., Wang, Z., Xie, Y., & Yang, Z. (2020). A Theoretical Analysis of Deep Q-Learning. *Proceedings of the 2nd Conference on Learning for Dynamics and Control*, PMLR 120:486–489.
- Gillen, S., Molnar, M., Byl, K. (2020). Combining Deep Reinforcement Learning and Local Control for the Acrobot Swing-up and Balance Task, *59th IEEE Conference on Decision and Control (CDC)*, Jeju, Korea (South), pp. 4129-4134.
- Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning. *MIT Press*.
- Hafner, D., Pasukonis, J., Ba, J., & Lillicrap, T. (2025). Mastering diverse control tasks through world models. *Nature*, 1-7.
- Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., Meger, D. (2018). Deep Reinforcement Learning That Matters. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1).
- Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., Silver, D. (2018). Rainbow: Combining Improvements in Deep Reinforcement Learning. *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*, 3215–3222.
- Kiran, B. R., Sobh, I., Talpaert, V., Mannion, P., Al Sallab, A. A., Yogamani, S. (2022). Deep Reinforcement Learning for Autonomous Driving: A Survey, *IEEE Transactions on Intelligent Transportation Systems*, 23(6): 4909-4926.
- Kumar, A., Fu, J., Soh, M., Tucker, G., Levine, S. (2019). Stabilizing Off-Policy Q-Learning via Bootstrapping Error Reduction. *Advances in Neural Information Processing Systems*, 32.
- Kumar, S. (2020). Balancing a CartPole System with Reinforcement Learning — A Tutorial. *arXiv preprint* arXiv:2006.04938.
- Mahajan, S., Harikrishnan, R., Kotecha, K. (2022). Adaptive Routing in Wireless Mesh Networks Using Hybrid Reinforcement Learning Algorithm, *IEEE Access*, 10: 107961-107979.
- McCarthy, J., Minsky, M. L., Rochester, N., Shannon, C. E. (2006). A Proposal for the Dartmouth Summer Research Project on Artificial Intelligence, August 31, 1955. *AI Magazine*, 27(4), 12.
- Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540), 529–533.
- Nagendra, S., Podila, N., Ugarakhod, R., George, K. (2018). Comparison of Reinforcement Learning algorithms applied to the Cart Pole problem. *arXiv preprint* arXiv:1810.01940.
- Prasetyo, R. E., Sumanto, S., Chaidir, I., Supriyatna, A. (2025). Reinforcement learning for bitcoin trading: A comparative study of PPO and DQN. *Jurnal Mandiri IT*, 14(2), 159-169.
- Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., & Dormann, N. (2021). Stable-Baselines3: Reliable Reinforcement Learning Implementations. *Journal of Machine Learning Research*, 22(268), 1-8.
- Rio, A. d., Jimenez, D., Serrano, J. (2024). Comparative Analysis of A3C and PPO Algorithms in Reinforcement Learning: A Survey on General Environments, *IEEE Access*, 12: 146795-146806.
- Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint* arXiv:1707.06347.
- Shen, D. (2024). Comparison of Three Deep Reinforcement Learning Algorithms for Solving the Lunar Lander Problem. In *2023 International Conference on Data Science, Advanced Algorithm and Intelligent Computing (DAI 2023)* (pp. 187-199). Atlantis Press.
- Stapelberg, B., Malan, K. M. (2020). A survey of benchmarking frameworks for reinforcement learning. *South African Computer*

*Journal*, 32(2), 258-292.

- Sutton, R. S., Barto, A. G. (2018). Reinforcement Learning: An Introduction (2nd ed.). *MIT Press*.
- Towers, M., Kwiatkowski, A., Terry, J., Balis, J. U., De Cola, G., Deleu, T., Goulão, M., Kallinteris, A., Krimmel, M., KG, A., Perez-Vicente, R., Pierré, A., Schulhoff, S., Tai, J. J., Tan, H., Younis, O. G. (2024). Gymnasium: A Standard Interface for Reinforcement Learning Environments. *arXiv preprint arXiv:2407.17032*.
- Wang, X., Wang, S., Liang, X., Zhao, D., Huang, J., Xu, X., Dai, B., & Miao, Q. (2024). Deep Reinforcement Learning: A Survey. *IEEE Transactions on Neural Networks and Learning Systems*, 35(4), 5064-5078.
- Zha, D., Lai, K.-H., Huang, S., Cao, Y., Reddy, K., Vargas, J., Nguyen, A., Wei, R., Guo, J., Hu, X. (2020). RLCARD: A Platform for Reinforcement Learning in Card, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence (IJCAI-20)*, Yokohama, Japan, 5264-5266.
- Zhong, L. (2024). Comparison of Q-learning and SARSA Reinforcement Learning Models on the Cliff Walking Problem. In *Proceedings of the 2023 International Conference on Data Science, Advanced Algorithm and Intelligent Computing*, (pp. 207–213). Atlantis Press.
- Zhu, Y., Wang, Z., Chen, C., Dong, D. (2022). Rule-Based Reinforcement Learning for Efficient Robot Navigation with Space Reduction, *IEEE/ASME Transactions on Mechatronics*, 27(2): 846-857.