# Balancing a Segway

Abhivarya Kumar[1], Jia Bhargava[2], Tanay Srinivasa[3], Vandita Lodha[4]

[1] CSAI, Plaksha University, India; [2] RCPS, Plaksha University, India

[3] RCPS, Plaksha University, India; [4] CSAI, Plaksha University, India

abhivarya.kumar; jia.bhargava; tanay.srinivasa; vandita.lodha @plaksha.edu.in

***Abstract— This report presents the development and deployment of reinforcement learning (RL) algorithms for balancing a Segway, a two-wheeled self-balancing robot modeled as an inverted pendulum on a cart. While traditional controllers such as PID and LQR have been widely used for this problem, RL-based approaches remain less explored, particularly in real-world hardware implementations. We derive the system's dynamics to create a more realistic simulation environment, introducing a discretized action space. Additionally, we take into consideration compensation for hardware non-idealities through sensor filtering and motor deadband compensation. Deep Q-Network (DQN) and Proximal Policy Optimization (PPO) agents are trained in simulation using both standard and the proposed state-action-based reward functions, then transferred and fine-tuned on custom-built Segway hardware. Experimental results show that the RL agents are able to understand the required behaviors required to balance the robot. The use of filtering on the suggested control action is found to experimentally reduce the control effort of the agent. It is found that the performance of agents is sensitive to initial conditions and hardware-specific tuning. This work demonstrates the feasibility and challenges of deploying RL controllers on real-world robotic systems with non-ideal components.***

***Keywords—CartPole, Control, Deep Q-Network, Proximal Policy Optimization, Sim-to-Real Transfer***

## I. INTRODUCTION

In this project we aim to tackle the balancing of a segway using Reinforcement Learning. A segway is a two-wheeled robot, typically modelled as an inverted pendulum on a cart. Traditional control techniques such as PID and LQR have been applied to this problem widely and implemented on hardware as well, while limited cases of the same have been done with Reinforcement Learning. The Cart-Pole problem is a classic problem in Reinforcement Learning textbooks, and hardware implementations of the same are typically in the form of a cart on a frictionless rail, and a pendulum mounted on a low friction joint. In our project we aim to implement these models on a segway hardware, whose components unideal and require an understanding of the system itself. We perform compensation techniques such as sensor filtering, and deadband compensation of DC motors to help the system perform more ideally.

Most of the work done on the Cart-Pole problem using RL based methods is a follow up on the work done by Barto and Sutton in their paper on Neuronlike adaptive elements that can solve difficult learning control problems[1]. However, the dynamics equations in this paper make certain simplifications in the derivation, making the environment easier to learn in for agents. Additionally, a binary action space is used, that is to either move left or right with a fixed force. These simplifications, while helpful in simulation, increase the simulation to hardware gap. This binary action space causes a high control effort and a high frequency of change in control action, causing high overshoot, and wear and tear on the motors. Our contribution lies in the usage of more realistic dynamics in the initial simulation training, followed by a further discretised action space allowing for small and large actions. Additionally, we propose a reward function based on the state action pair, which rewards the robot for being close to the ideal upright position, while also penalizing the control effort.

## II. RELATED WORKS

Mishra and Arora's 2022 work[2] applied DQN and its Double DQN extension to the CartPole balancing challenge, introducing the Huber loss as a replacement for the usual squared error in training. This adjustment yielded a markedly faster learning process – the DQN agent converged to the optimal policy in fewer episodes when trained with Huber loss. Furthermore, the Double DQN approach (which uses two networks to reduce overestimation of Q-values) outperformed the vanilla DQN, converging much quicker and with a smaller error margin. This paper's results indicate that addressing Q-value overestimation and using a stable loss function can substantially boost DQN's performance on CartPole, enabling more efficient and stable learning.

In the study by Rio, Jimenez and Serrano, Proximal Policy Optimization (PPO) was evaluated against A3C across the CartPole and Lunar Lander environments, highlighting its superior training stability at the cost of longer execution times. PPO employed a clipped surrogate objective to constrain policy updates, preventing large destabilizing changes and ensuring smoother learning progress. In CartPole, PPO achieved the success threshold (>475 reward) by 200 episodes, outperforming A3C in stability and recording a 10.5% success rate over episodes, compared to A3C's 4.45%.Despite longer training times ( ~452s for 400 CartPole episodes and ~16,833s for 5,000 Lunar Lander episodes), PPO's controlled policy updates proved advantageous for tasks where stability and predictable performance are prioritized.

Jo and Kim’s 2024 comparative study[4] provides a head-to-head evaluation of DQN, PPO, and A2C on the CartPole environment. Their findings clearly favor DQN: across 100 runs, the DQN agent learned to balance the pole more reliably and quickly than its PPO and A2C counterparts. DQN achieved higher rewards with less variance, indicating superior stability. In contrast, while PPO and A2C were able to solve CartPole, they did so with slower convergence or greater fluctuations in performance. The authors conclude that DQN significantly outperforms the policy-gradient algorithms, attributing its success to efficient experience replay and a well-managed exploration-exploitation trade-off. This study underscores the strengths of DQN in simple environments and provides insight into each algorithm’s strengths and weaknesses on the CartPole benchmark.

One drawback of the literature reviewed is in the fact that all comparisons are made on models trained in simulation, and don’t talk about its implementation on hardware. Other literature which discusses hardware use ideal components such as a cart mounted on a low friction rail with a pendulum. This gap leads towards the contribution added in this report. We first derive the dynamics equations of the simulation environment, then discuss the RL formulation and methodology, following which we discuss the implementation and training of the agents on simulation and fine tuning on hardware.

## III. Problem Statement

The main goal of this project is to balance the segway hardware for as long as possible while not moving the robot from its initial position. That is, the RL agent must choose an action F (or analogous) such that the robot must not exceed a particular lean while not moving more than a certain distance from its initial starting position.

## IV. Dynamics

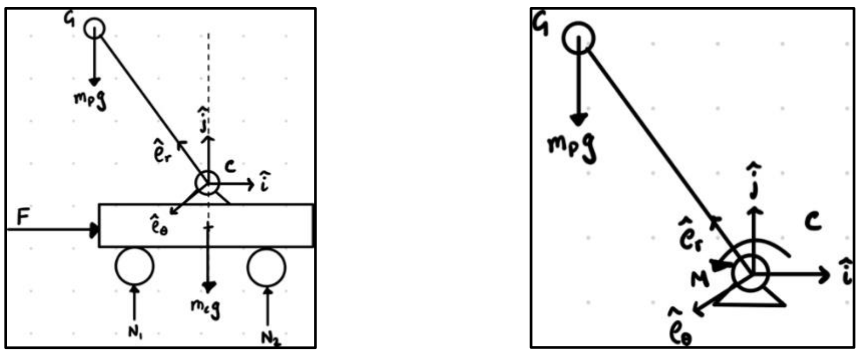

Fig 1: Free Body Diagrams of: (a) System, (b) Pendulum

Apply Linear Momentum Balance on (a),

$$F = m_c\ddot{x} + m_P\left(\ddot{x} + d\dot{\theta}^2\sin\theta - d\ddot{\theta}cos\theta\right) + b\dot{x}$$

Apply Angular Momentum Balance on (b),

$$0 = \ddot{\theta}\left(I + m_p d\right) - m_p d(\ddot{x}\cos\theta + g\sin\theta)$$

From these equations the state space is chosen to be $x$, $\theta$, $\dot{x}$, $\dot{\theta}$ and the control inputs are $F$.

## V. RL Formulation

The control input/action in the physics model derived above is the Force given into the system by the wheels. This force is generated by the motors, which provides a torque to the motor shaft. From the characteristic equation of a DC motor we get the rotation speed of the motor, the voltage applied to the motor, and the torque generated by the motor are all linearly related, and hence there is a linear mapping from the input voltage range of the motor to the output torque and shaft rate of the motor. Since the force and torque are linearly related as well, this can be extended to that. Hence, the action space here is a discretised voltage input into the motor, which breaks down the -12V to 12V into 41 categories, that is from -12V to 12V in 0.6V increments.

As derived in the dynamics, the required state space for the calculations includes $x$, $\theta$, $\dot{x}$, $\dot{\theta}$. We set the observable state range from -2.4m to +2.4m in x, and -12° to +12° in θ. The episode terminates if either x or θ goes beyond the specified range. There are no restriction in the velocities. The environment in simulation is governed by the dynamics derived in Section IV, however while deploying on the robot, there may be additional effects from unmodelled forces.

The reward functions that we try in our case are the Barto-Sutton[4] reward and a reward function based on the State and Action at each time step. The Barto-Sutton reward function gives a penalty of -1 at the time step when the episode has terminated based on the termination conditions mentioned above and a reward of 0 for every other time step in the episode. Another reward function that we experiment with is one that varies based on the state and the action chosen by the agent. It has a reward based on lean angle as well as an action penalty based on the severity of the chosen action.

In this project based on the literature survey we experiment with PPO and DQN, where the DQN is trained with Huber Loss, LQR Cost, Mean Squared Temporal-Difference. The PPO is trained with a combined loss that combines Clipped Surrogate Policy Loss, Value Function Loss, and an Entropy Bonus.

## VI. Methodology

### A) Agents:

One of the agents we experiment with is the Deep Q-Network (DQN), which minimizes the Mean Squared Temporal-Difference (TD) error between the current Q-value estimate and a bootstrapped target. Concretely, for a batch of transitions:

$$(s_t, a_t. r_t, s_{t+1}, d_t)$$

$$y_t = r_t + \gamma(1 - d_t) max Q_{target}(s_{t+1}, a')$$

$$L_{DQN} = E_{(s,a,r,s',d)}[(Q_{Online}(s_t, a_t) - y_t)^2]$$

where, $Q_{Online}$- is the current (learned) Q-network
$Q_{target}$ - is the periodically updated “target” network, to stabilize in learning.
γ - is the discount factor

The main characteristics of the DQN is it is an off-policy, value-based RL agent. Some of its key ideas are: Transitions are stored in a buffer and sampled at random to break correlation; A separate network is used to compute $y_t$ is updated slowly or periodically that improves stability. Lastly epsilon-greedy policies can be used while training.

The next algorithm we experiment with is Proximal Policy Optimization (PPO). For an on-policy batch of size NNN, SB3’s PPO optimizes a combined loss.

$$L_{PPO} = L^{clip} + c_v L^{vf} - c_e L^{ent}$$

Where $L^{clip}$, $L^{vf}$, $L^{ent}$ are the clipped surrogate policy loss, value function loss, and the entropy bonus as define below.

$$L^{clip} = -\frac{1}{N}\Sigma min(r_t(\theta)A_t\ clip(r_t(\theta), 1 - \epsilon, 1 + \epsilon)A_t)$$

$$L^{vf} = \frac{1}{N}\Sigma(V_\theta(s_t)R_t)^2$$

$$L^{ent} = -\frac{1}{N}\Sigma E_{a\sim\pi_\theta}[log\pi_\theta(a|s_t)]$$

From literature we choose the clip range ($\epsilon$) to be 0.2, the value loss coefficient to be 0.5, and entropy coefficient to be 0.

## B) SIMULATION ENVIRONMENT:

We use OpenAI's Gymnasium library to set up the training and deployment environment in simulation and on hardware. The default CartPole environment available on Gymnasium uses a binary action space, that is move left or right with a fixed force magnitude. Additionally, the dynamics are simplified neglecting frictional losses. Given this, using a PPO or DQN model on their default environment leads to the result where the pole is unable to stay upright. This can be seen in their official documentation for the environment, where in the GIF rendering the environment, the pole continues to fall beyond 12 degrees, which then terminates the episode, resetting the pole back at the center position. Hence, we modify the equations of motion as described in Section IV and additionally increase the discretization of the action space into 41 actions as described in Section V. This allows for finer control of the robot giving the agent the ability to choose smaller actions or no action when the robot is near the balanced state, and larger actions where it is farther away from the upright position.

## C) REWARD FUNCTIONS:

The Sutton-Barto reward, widely used throughout literature as well as Gymnasiums CartPole environment, penalizes the agent with a reward of -1 at the termination of the episode, and a reward of 0 otherwise. This reward structure in literature has proved useful, however it does not take into consideration the deviation of the robots state from the upright position, and does not penalize a large control effort, which was not a consideration due to the action space used in the paper. We propose the use of the State-Action Reward Function as defined by:

$$R_t = cos^2(\theta) - \left(1 - cos(\theta)\right) \times k \times (action - 20)^2$$

Where, $\theta$ is the current lean angle of the robot, and
*action* is the selected action (0 to 41) by the agent.

The above reward function rewards the agent more when it is near the upright position. It penalizes a large control effort more when the robot is near the upright position, and lesser when further away from the upright position. We use this State-Action Reward combined with the Sutto-Barton Reward

## D) EXPERIMENTAL SETUP:

The initial segway hardware was built as a Masters Project at the Indian Institute of Science, this hardware uses a Rhino 500 RPM 6kgcm DC motor, with quadrature encoders. The sensors used to measure the states are an MPU6050 which measures $\theta$ and $\dot{\theta}$. The encoders integrated with the motors are used to measure $x$ and $\dot{x}$. Both wheels of robot are attached together with a single axle, driven by a bevel gear powered by the motor. A Cytron MDD10A Motor Driver is used to drive the motor. In the Masters project an Arduino Uno was the primary microcontroller being used, however due to its restricted compute power, and difficulties in implementing RL agents on the microcontroller, the main microcontroller was shifted to a Raspberry Pi 3B+ for this project. Given this, all Arduino libraries used had to be re-written for the robot. Additionally, both sensors and the actuator have non-ideal behaviors in the form of noisy readings and a motor deadband. To compensate this, a complimentary filter, a type of sensor fusion algorithm, was used to merge the accelerometer and gyroscope readings to get reliable values of $\theta$. The coefficient was tuned to a strength of 0.96 to improve its step response. A low pass filter was added over the numerical derivative measured by the encoder, which is used to calculate $\dot{x}$. Lastly the kinetic and static deadband coefficients were measured for the motor.

The agent is first trained in simulation on the environment described in Section VI B and stored as a .zip file. This file is then loaded on the hardware and is deployed on a hardware environment with the same observation and action space and is further fine-tuned. The training process requires the user to manually position the robot to a state which is within 5 degrees of being upright, after which the robot starts the episode. When the episode terminates, the terminal requests the user to manually reposition it after which the next episode starts. To ensure timing control on the microcontroller, the priority of the task is set to highest, and is run on an isolated core on the microcontroller. The time between two consecutive actions is enforced at 10ms, based on the settling time of the sensors. A low pass filter was added to the control actions given by the agent before controlling the motor to prevent wear and tear from high frequency oscillations in velocity direction.

## VII. Contributions

The dynamic equations for the Segway, presented in Section II, were originally derived as part of a project for the Control Autonomy, Planning and Navigation (RO3003) course, which was taken by team members Jia and Tanay and instructed by Professor Shashank Tamaskar. In this course, traditional control techniques, including PID and LQR, were applied on the Segway using the derived dynamics. Additionally, the hardware was partially built at IISc by MTech students Nilay Srivastava and Vishwas Gajera, which was then modified by team members Jia and Tanay as a part of this course. For the reinforcement learning project, these equations of motion have been used to model the system; however, instead of conventional control methods, RL-based agents and policies are developed to balance and control the segway.

## VIII. Results

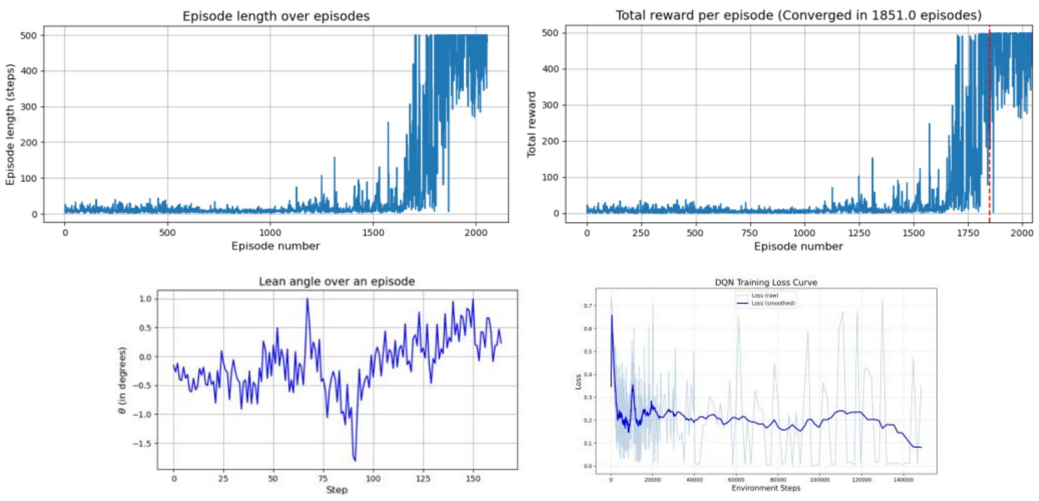

Fig 1: Training DQN in Simulation

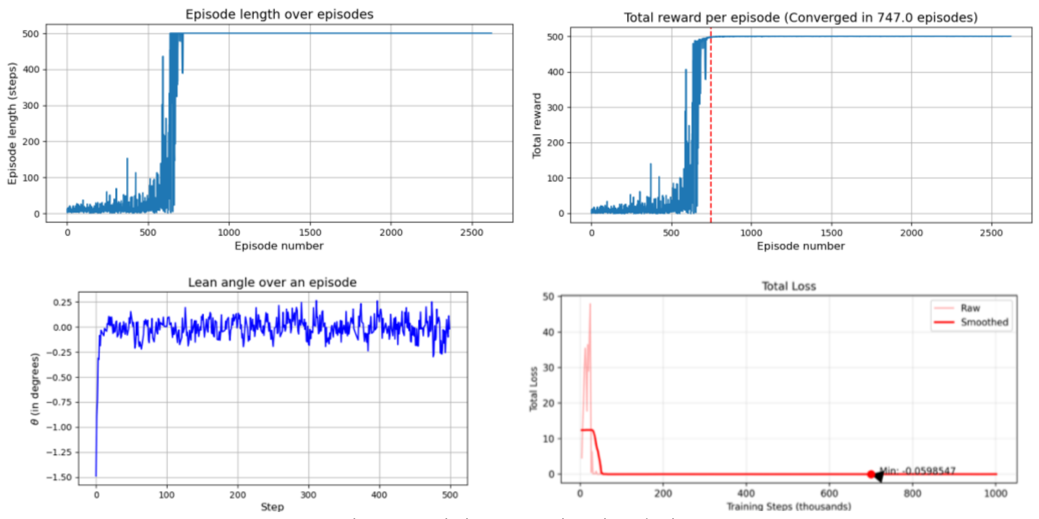

Fig 2: Training PPO in Simulation

DQN was trained for 200,000 time steps and PPO was trained for 1,000,000 steps in the simulations above. Convergence in for training is attained when an episode reward of 498 is received, where the maximum reward per episode is 500. From figures 1 and 2, it is observed that PPO converges faster (747 episodes) in simulation as compared to DQN (1850 episodes). Additionally, the episode reward for DQN fluctuates over episodes even after convergence. This contrasts with Jo and Kim's 2024 comparative study mentioned in Section II, where DQN outperformed all other models. The success of PPO in our simulations could be due to the combined clipped surrogate policy loss, value function loss, and the entropy bonus used during training.

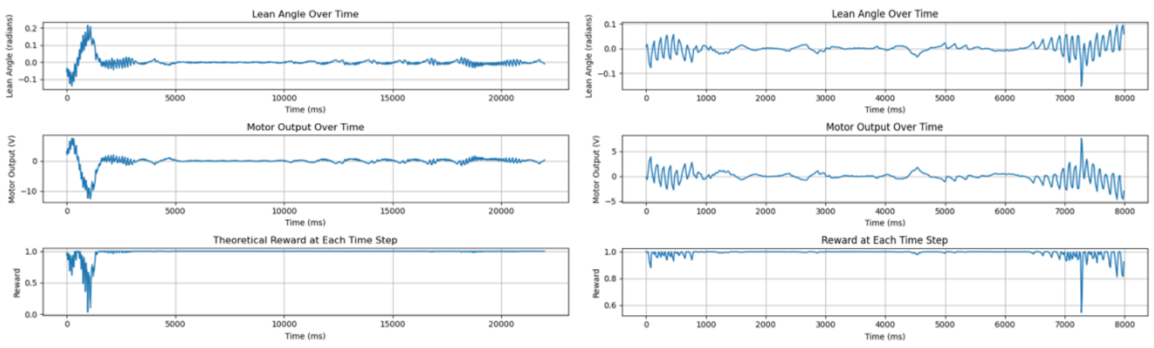

Fig 3: Deployment of PID (Left) and PPO (Right) on Robot

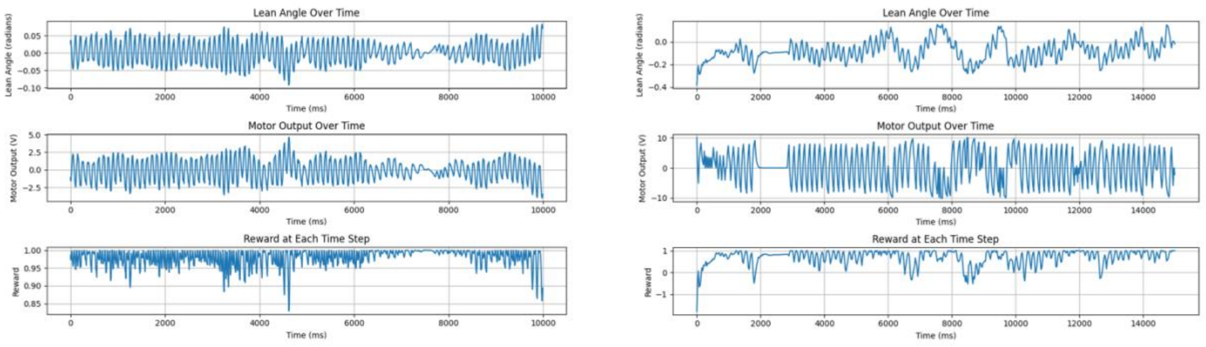

Fig 4: Deployment of DQN on Robot with Alpha 0.3 (Left) and 0.5 (Right)

Fig 3 shows the PID baseline tuned for the robot, in this case, the robot balances for about 20 seconds, after a disturbance is given to the system. The longest balance time achieved by the baseline was 40 seconds, however, a higher control effort was observed there due to oscillations caused by a high proportional gain. In the third subplot, the theoretical reward using the state-action reward is shown to compare with other deployed agents. On the right, the deployment of PPO can be seen. The robot without any disturbance given, was able to balance for 8 seconds. Fig 4 shows the deployment of DQN on the robot using two different values of a low pass filter. The lower the value of the filter coefficient, the more filtered the value is. This was required during deployment as the agent tended to rapidly change the direction of the control action. As control effort was one of the parameters tracked in our study, it can be clearly seen for alpha 0.3, the voltage range for control input, as well as magnitude of oscillations around the upright position is lower. The same alpha value was also used in the PPO model shown in Fig 3. The same trend was also observed for higher filter coefficient values while deploying the PPO model as well.

Despite the lower control effort with a lower alpha value with DQN, the robot balances for 4 seconds longer in the case with the 0.5 filter coefficient. It is important to note, that the balancing time on the hardware was very sensitive to the initial conditions for all RL agents. Additionally, it was observed that the offset in the lean angle was different for all the RL agents as well as the PID while balancing. That is, the target angle of lean had to be tuned during the deployment, such that the robot had an even tendency to fall on both sides. This tuning of the target angle increased the balance time for all controllers implemented on the hardware from a few seconds to the values observed. The balance times on hardware could be increased by further training on hardware, which we were unable to do due to the time intensive nature of training on hardware for multiple models. The DQN model was trained for a total of 1500 timesteps, and the PPO model was trained for a total of 1000 time steps for the data shown in the plots above.

## GITHUB REPOSITORY

https://github.com/tanayrs/CartNPendulum/tree/main

## REFERENCES

[1] A. G. Barto, R. S. Sutton and C. W. Anderson. *Neuronlike adaptive elements that can solve difficult learning control problems*. IEEE Transactions on Systems, Man, and Cybernetics, vol. SMC-13, no. 5, pp. 834-846, Sept.-Oct. 1983, doi: 10.1109/TSMC.1983.6313077.

[2] Shaili Mishra and Anuja Arora. *Double Deep Q Network with Huber Reward Function for Cart-Pole Balancing Problem* [J]. Int J Performability Eng, 2022, 18(9): 644-653.

[3] A. d. Rio, D. Jimenez and J. Serrano. *Comparative Analysis of A3C and PPO Algorithms in Reinforcement Learning: A Survey on General Environments*. IEEE Access, vol. 12, pp. 146795-146806, 2024, doi: 10.1109/ACCESS.2024.3472473.

[4] E.-H. Jo and Y. Kim. *Performance comparison of reinforcement learning algorithms in the CartPole game using Unity ML-Agents*. Journal of Theoretical and Applied Information Technology, vol. 102, no. 16, pp. 6076–6083, Aug. 2024.

[5] Farama Foundation. *Cart Pole – Gymnasium.* Gymnasium Documentation. [Online]. Available: https://gymnasium.farama.org/environments/classic_control/cart_pole