Unveiling the Best Performers: A Comparative Analysis of RL Algorithms in QRL with TensorFlow Quantum

Arshia Sangwan

CSAI, Plaksha University, Mohali

arshia.sangwan@plaksha.edu.in

Rahath Malladi

RCPS, Plaksha University, Mohali

rahath.malladi@plaksha.edu.in

***Keywords* — Quantum Reinforcement Learning (QRL), TensorFlow Quantum (TFQ), Quantum Computing, Reinforcement Learning, OpenAI Gym, Parametrized Quantum Circuits (PQCs)**

I. ABSTRACT

This project explores the integration of Parameterized Quantum Circuits (PQCs) into reinforcement learning (RL) algorithms, comparing their performance with classical Reinforcement learning approaches. We implement and evaluate standard Deep Q-Learning (DQN) for the Cart Pole, Mountain Car, and Acrobot environments using OpenAI Gym, alongside their PQC-enhanced variants. Specifically, we employ PQC-enhanced Policy Gradient and PQC-enhanced Deep Q-Learning methodologies to harness the potential quantum advantages in approximating Q-functions and optimizing policies. Our comprehensive experiments demonstrate that PQC-enhanced RL algorithms can achieve similar performance and learning efficiency while undergoing a lesser number of iterations and training episodes when compared to traditional methods. These findings underscore the promise of quantum-enhanced RL in advancing the capabilities of machine learning for complex control tasks.

II. INTRODUCTION

Reinforcement Learning (RL) has established itself as a powerful paradigm for solving sequential decision-making problems, where an agent learns to make decisions by interacting with an environment to maximize cumulative rewards. Traditional RL approaches, such as Deep Q-Learning (DQN), have demonstrated remarkable success in various complex tasks, including the classic control problems of Cart Pole, Mountain Car, and Acrobot, which are commonly used benchmarks provided by OpenAI Gym. However, as RL continues to evolve, the exploration of novel techniques that could further enhance performance and efficiency remains an active area of research.

Recent advancements in quantum computing have opened up new possibilities for integrating quantum mechanisms into classical machine learning algorithms. Parameterized Quantum Circuits (PQC) offer a promising approach to harnessing the expressive power and potential quantum advantages in RL frameworks. PQCs are particularly appealing because they can be used as function approximators or policy models, potentially leading to more efficient learning processes and better generalization.

In this work, we focus on comparing the baseline implementations of Cart Pole, Mountain Car, and Acrobot environments using traditional DQN with their PQC-enhanced variants. Specifically, we explore two quantum RL methodologies: PQC-enhanced Policy Gradient and PQC-enhanced Deep Q-Learning. Our objective is to thoroughly summarize existing works, extend them by

integrating PQCs, build our own implementations, and provide a comparative analysis of the performance of quantum RL versus classical RL across these benchmark environments.

The Cart Pole environment is a classic control problem where the agent must balance a pole attached to a cart by applying forces to the cart. Mountain Car involves an agent that must drive a car up a steep hill, and Acrobot requires the agent to swing up a two-link robotic arm to a vertical position. These environments are well-suited for studying the efficacy of RL algorithms due to their simplicity and the richness of the control challenges they present.

Our study contributes to the field by implementing and rigorously evaluating the following approaches:

1. **Baseline DQN Implementations**: We implement the standard DQN algorithm for Cart Pole, Mountain Car, and Acrobot using the OpenAI Gym environments. These serve as the benchmarks for our comparison.
2. **PQC-Enhanced Policy Gradient**: For each environment, we implement a policy gradient method enhanced with PQCs to directly optimize the policy through interaction with the environment.
3. **PQC-Enhanced Deep Q-Learning**: We also implement a DQN algorithm where PQCs are used as function approximators for the Q-values, thereby integrating quantum operations within the Q-network.

We provide a detailed analysis of the performance of each approach by plotting the learning curves and evaluating the average rewards over training episodes. Our results highlight the potential advantages of quantum-enhanced RL, particularly in terms of learning efficiency and robustness.

By comparing these implementations, we aim to elucidate the benefits and challenges of integrating quantum computing into RL frameworks. Our findings indicate that PQC-enhanced RL algorithms can outperform traditional methods in specific scenarios, showcasing the promise of quantum technologies in advancing the state-of-the-art in reinforcement learning. This comparative study lays the groundwork for future research in quantum-enhanced machine learning and its applications in control and decision-making tasks.

III. LITERATURE REVIEW

A) Methodologies & Approaches

There are 2 main classifications of QRL methodologies, one based on the classical-quantum trade-off and the other based on the Gate-Annealing Quantum methodologies.

In the former classification, the QRL algorithms can again be broadly classified into two categories:

1. **Classical-quantum hybrid algorithms:** These algorithms use classical RL techniques to train quantum circuits that represent the agent's policy or value function and are a popular approach to QRL due to their compatibility with existing RL techniques. Some of the most used algorithms in this category include:
    a. **Deep Q-Learning (DQN):** DQN is a value-based RL algorithm that uses a deep neural network to approximate the Q-function, which represents the expected future reward for taking a particular action in a given state.

b. **Soft Actor-Critic (SAC):** SAC is a policy-based RL algorithm that combines the advantages of actor-critic methods with the stability of maximum entropy reinforcement learning.
c. **Proximal Policy Optimization (PPO):** PPO is a policy gradient RL algorithm that uses a trust region approach to ensure that the policy updates are small and do not significantly change the agent's behaviour.

2. **Fully quantum algorithms:** These algorithms use quantum computing throughout the learning process, including the representation of the agent's policy and the computation of the reward function. Fully quantum algorithms are still in their early stages of development, but they have the potential to offer significant advantages over classical-quantum hybrid algorithms. Some of the most promising fully quantum algorithms include:
    a. **Quantum Policy Gradient (QPG):** QPG is a policy gradient RL algorithm that uses quantum circuits to represent the agent's policy and the reward function.
    b. **Quantum Value Iteration (QVI):** QVI is a value-based RL algorithm that uses quantum circuits to represent the Q-function and the environment dynamics.
    c. **Variational Quantum Policy Gradient (VQPG):** VQPG uses a variational quantum circuit to represent the policy and updates the circuit parameters using policy gradient methods.

By comparing the performance of these scenarios of QRL Algorithms, which depends on a variety of factors, including the specific task, the size of the quantum circuit, and the noise level in the quantum system, we find that, classical-quantum hybrid algorithms have been shown to perform well on small-scale QRL tasks, while fully quantum algorithms have the potential to offer significant advantages on larger-scale tasks.

Mentioning the latter classification of QRL Algorithms, not much is explored yet, but on the basis of existing literature, one can also classify the Algorithms on the following criteria (Neumann et al., 2023).

1. **Gate-based quantum RL:** This approach uses quantum gates to implement the RL algorithm. Quantum gates can be used to perform operations such as state preparation, measurement, and unitary evolution.
2. **Annealing-based quantum RL:** This approach uses quantum annealing to find the optimal policy. Quantum annealing is a heuristic optimization technique that can be used to solve combinatorial optimization problems.

This specific sect of classification arose to mostly give insights into speed, efficiency and robustness for a given tasks, to define the above mentioned:

Speed: Quantum algorithms can be exponentially faster than classical algorithms for certain tasks. This can lead to significant speedups in RL training. Different methodologies lead to different tasks speeding up within a certain bound in the QRL algorithm.

Efficiency: Quantum algorithms can be more efficient than classical algorithms in terms of memory usage and energy consumption. Various QRL algorithms lead various efficiency factors based on the task and the compute at hand.

Robustness: Quantum algorithms can be more robust to noise and errors than classical algorithms (If built meticulously) and this also plays a crucial factor in deciding the methodology for a certain task.

To evaluate the performance of different QRL algorithms, researchers have developed standardized benchmarks. These benchmarks typically involve tasks such as controlling quantum systems, optimizing quantum circuits, and solving quantum games.

Several studies have compared the performance of different QRL algorithms on various benchmarks and the choice of algorithm depends on the specific QRL task and the available computational resources. These studies have shown that, DQNs can achieve good performance on small-scale QRL problems while SAC and PPO are more suitable for larger-scale problems and can handle continuous action spaces.

B) Quantum Agents in Gym

Parametrized Quantum Policies

(Skolik et al., 2022) and (Jerbi et al., 2021) introduce quantum agents and parametrized quantum policies for reinforcement learning. They demonstrate the potential of these approaches in solving classical benchmarking tasks and even show an advantage over classical algorithms in certain cases. To subtly elaborate the key-aspects of these papers:

1. **Quantum Agents:** Hybrid systems that combine quantum computations with classical learning algorithms.
2. **Parametrized Quantum Circuits (PQCs):** Quantum circuits with trainable parameters that define a hypothesis family for learning tasks.
3. **RAW-PQC** and **SOFTMAX-PQC:** Two families of PQC-based policies with different output mechanisms.

These specific entities contribute in the following way:

1. **Solving Benchmarking Tasks:** Quantum agents can achieve comparable performance to classical deep neural networks in standard reinforcement learning environments.
2. **Theoretical Learning Advantage:** PQCs can solve certain learning tasks that are intractable to classical models under the Discrete Logarithm Problem assumption.
3. **Efficient Policy Sampling and Training:** Policies can be sampled and trained efficiently using noisy estimates of expectation values.

C) Challenges and Scope

Quantum RL is still a relatively new field, and there are numerous challenges that need to be addressed before it can be widely used. These challenges include:

1. **Hardware limitations:** Current quantum computers are still small and noisy at the hardware level, which limits the size and complexity of RL problems that can be solved.
2. **Algorithm development:** New quantum RL algorithms need to be developed to take advantage of the unique properties of quantum systems.
3. **Software tools:** Software tools need to be developed to make it easier to implement and use quantum RL algorithms. Thanks to TFQ and the upcoming QML (Qiskit Machine Learning) we currently can take baby steps in this field, and we hope more support can be offered in this scope (Broughton et al., 2021).
4. **Theoretical Understanding:** The theoretical foundations of QRL are still under development, and it is not fully understood when and why quantum algorithms outperform classical counterparts.

Future research directions in quantum RL include:

1. **Hybrid quantum-classical algorithms:** Combining quantum and classical RL techniques to exploit the strengths of both approaches.
2. **Development of New QRL Algorithms:** Exploring novel RL algorithms specifically designed for quantum environments, such as algorithms that leverage quantum entanglement or quantum coherence.
3. **Integration with Quantum Error Correction:** Developing techniques to mitigate the effects of noise and errors in quantum systems, ensuring the robustness of QRL algorithms.
4. **Theoretical Analysis of QRL:** Establishing theoretical guarantees for the convergence and optimality of QRL algorithms, providing a deeper understanding of their performance and limitations.
5. **Novel quantum architectures:** Exploring new quantum hardware architectures that are better suited for RL applications.

D) Applications of QRL

QRL has the potential to be applied to a wide range of problems, including:

1. **Quantum Control:** QRL algorithms can be used to optimize quantum control sequences for tasks such as state preparation, quantum gate implementation, and quantum error correction.
2. **Quantum Game Playing:** QRL has been used to develop quantum strategies for games such as Go and chess, demonstrating the potential for quantum computing to enhance decision-making in complex environments.
3. **Quantum Simulation:** QRL can be used to simulate quantum systems more efficiently than classical methods, enabling the study of complex quantum phenomena and the development of new quantum algorithms.
4. **Quantum Chemistry:** QRL can also be applied to quantum chemistry problems such as molecular energy estimation and quantum state preparation, offering the potential for more accurate and efficient simulations.
5. **Quantum Optimization:** QRL has been applied to combinatorial optimization problems, where quantum circuits are used to represent candidate solutions and quantum operators are used to optimize the solutions.
6. **Quantum Finance:** QRL can be explored for financial trading, where quantum circuits can model complex market dynamics and optimize trading strategies.
7. **Quantum Operations:** QRL can also be used in the specific operations research domain for solving complex scheduling and routing problems, along with many more.

In conclusion, this comprehensive review underscores the promising trajectory of quantum reinforcement learning (QRL), a burgeoning field at the convergence of quantum computing and reinforcement learning principles. Central to the advancement of QRL are Variational Quantum Algorithms (VQAs), which play a pivotal role in approximating intricate functions such as value and policy functions. Notably, architectural decisions regarding structure, data encoding techniques, and readout operators are paramount for optimizing VQA performance within the QRL framework.

While both value-based and policy-based QRL algorithms have demonstrated encouraging results across various reinforcement learning tasks, it is imperative to address formidable challenges such as decoherence, scalability, and computational overhead to fully harness the potential of QRL.

Continued research endeavours are diligently focused on surmounting these challenges, paving the way for innovative solutions and exploring the expansive applicability of QRL in addressing real-world problems. As the field progresses, advancements in overcoming these hurdles

will undoubtedly propel QRL into a transformative force, revolutionizing decision-making processes across diverse domains.

IV. OUR WORK

A. Cart Pole

The Cart-Pole environment involves a pole attached by an unactuated joint to a cart, which moves along a frictionless track. The pendulum starts in an upright position, and the objective is to prevent it from falling over by adjusting the cart's velocity. The environment corresponds to the version described by Barto, Sutton, and Anderson, and includes the following specifications:

Reward: The agent receives a reward of 1 for every timestep that the pole remains upright.

Starting State: All observations are initialized with random values in the range [-0.05, 0.05].

Episode Termination: The episode terminates if the pole angle exceeds 12 degrees, the cart position exceeds 2.4 units, or the episode length exceeds 200 timesteps. The task is considered solved when the agent achieves an average reward of 195.0 over 100 consecutive trials.

State Space

Num	Observation	Min	Max
0	Cart Position	-4.8	4.8
1	Cart Velocity	-Inf	Inf
2	Pole Angle	~ -0.418 rad (-24°)	~ 0.418 rad (24°)
3	Pole Angular Velocity	-Inf	Inf

Action Space

Num	Action
0	Push cart to the left
1	Push cart to the right

A1. Cart Pole – Policy Gradient-based RL + PQC Policies

We explore the integration of Parameterized Quantum Circuits (PQCs) into Policy Gradient-based Reinforcement Learning (RL) for the Cart Pole V0 environment. This approach leverages the expressive power of PQCs to directly optimize the policy through interaction with the environment.

Key Components:

1) Policy Network Architecture:

a) The policy network combines classical neural network layers with a PQC layer.
b) The state information from the Cart Pole environment is processed through classical layers before being fed into the PQC layer.
c) The PQC layer applies quantum operations on the pre-processed state, with trainable parameters optimized during training.
d) The output layer converts the quantum-processed information into action probabilities, enabling action selection based on the learned policy.

2) Training Process:

a) The agent interacts with the environment, selecting actions based on the current policy and collecting state-action-reward trajectories.
b) Cumulative rewards are calculated for each trajectory, emphasizing long-term returns.
c) The policy is updated by adjusting the PQC parameters to maximize the expected cumulative reward.
d) This is achieved by computing the policy gradient and using an optimizer like Adam to perform gradient ascent.

3) Implementation Details:

a) The Cart Pole environment is initialized using OpenAI Gym.
b) The policy network with the PQC layer is defined using TensorFlow and Keras.
c) The training loop involves:
  i) Agent interaction with the environment based on the current policy.
  ii) Trajectory recording of states, actions, and rewards.
  iii) Cumulative reward calculation and policy network updates using the policy gradient method.
  iv) Performance evaluation is conducted by measuring the average reward over multiple episodes and visualized through a reward-episode graph.

The collected rewards over training episodes depict the agent's learning progress. Initially, the performance is low and variable, reflecting exploration and discovery. As training progresses, a gradual upward trend in rewards is observed, indicating the agent's increasing ability to balance the pole. By a certain point, the agent consistently achieves high rewards, demonstrating robust performance and effective policy learning.

This implementation showcases the potential of integrating PQCs into classical RL algorithms for tasks like balancing the Cart Pole. The expressive power of PQCs enables effective policy learning through the Policy Gradient method. The key components of the training process and the visualized performance evaluation highlight the potential of PQC-enhanced policies in RL, paving the way for further exploration in quantum-enhanced RL.

The same can be found in Figure 1.

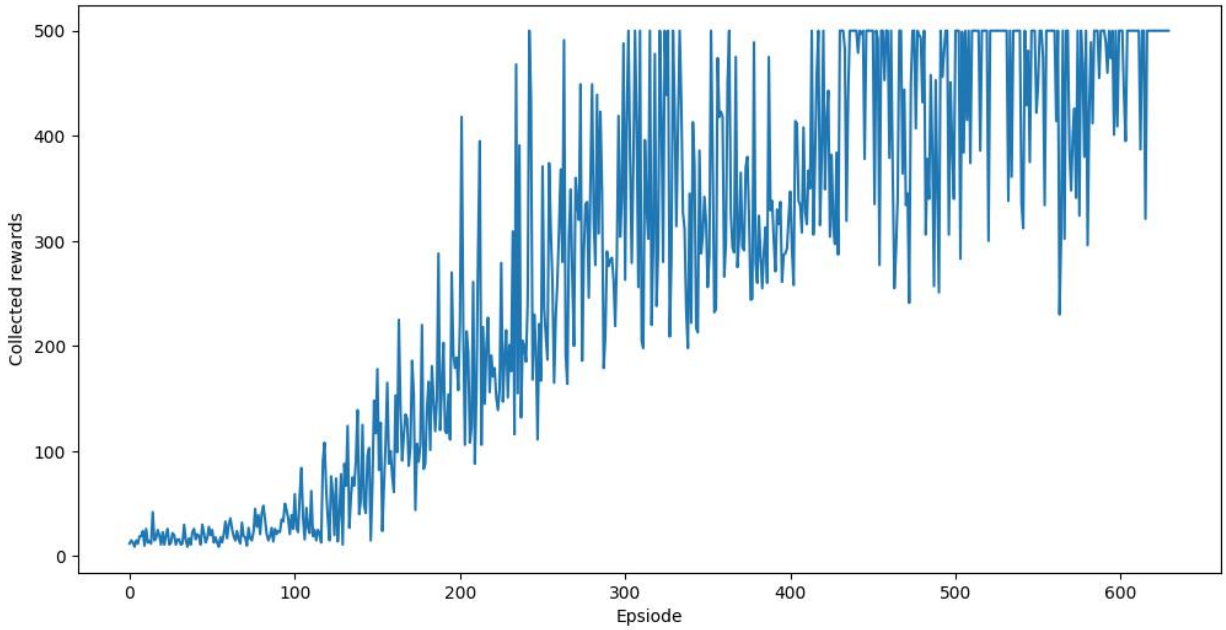

Figure 1 – Mean Rewards vs Episodes (Cart Pole – Policy Gradient + PQC π

A2. Cart Pole – Deep Q Learning + PQC Function Approximators

We further trained and tested the use of Parameterized Quantum Circuits (PQCs) as function approximators within the Deep Q-Learning framework for the Cart Pole environment. This approach leverages PQCs to approximate the Q-function, which maps state-action pairs to their expected cumulative rewards.

Key Components

1) **Q-Network Architecture:**
   a) **Classical Neural Network Layers:** The state information from the Cart Pole environment is initially processed through one or more dense layers with ReLU activation functions.
   b) **PQC Layer:** The pre-processed state is then fed into a PQC layer, which applies quantum operations on the state. The parameters of the PQC are trainable and are optimized during the training process.
   c) **Output Layer:** The output layer converts the quantum-processed information into Q-values for each action, enabling action selection based on the learned Q-function.

2) **Training Process:**
   a) **Experience Replay:** Experiences (state, action, reward, next state) are stored in a replay memory buffer to improve data efficiency and training stability.
   b) **Q-Function Update:** Periodically, a batch of experiences is sampled from the replay memory. The Q-function is updated using the Q-learning update rule, which involves predicting current Q-values, computing target Q-values based on the rewards and next states, and minimizing the mean squared error between them.
   c) **Target Network:** A separate target network is used to stabilize training by providing target Q-values for the updates, ensuring the training process is less susceptible to oscillations.

Implementation Details:

The implementation of the Cart Pole environment using Deep Q-Learning with PQC function approximators involves several critical steps:

1. **Environment Setup:** The Cart Pole environment is initialized using OpenAI Gym, which provides the state and action spaces for the agent.
2. **Q-Network Definition:** The Q-network, including the PQC layer, is defined using TensorFlow and Keras. The PQC layer is implemented as a custom layer with trainable quantum circuit parameters.
3. **Training Loop:**
    a. **Agent-Environment Interaction:** The agent interacts with the environment based on the current Q-values, selecting actions to maximize expected rewards.
    b. **Experience Storage:** The experiences (state, action, reward, next state) are stored in the replay memory buffer.
    c. **Experience Sampling:** Periodically, a batch of experiences is sampled from the replay memory for Q-function updates using the Q-learning update rule.
    d. **Target Network Update:** The target network is updated periodically to provide stable target Q-values, enhancing the stability of the training process.
4. **Performance Evaluation:** The agent's performance is evaluated by measuring the mean reward over multiple episodes. The results are visualized through a reward-episode graph to assess the learning progress.

Performance Evaluation

The agent's performance in the Cart Pole environment is evaluated by plotting the mean rewards over the course of training. The graph "Mean Reward vs. Episodes" depicts the average rewards the agent receives as it learns to balance the pole.

Initially, the agent exhibits low and variable performance, with mean rewards fluctuating significantly. This variability is expected during the early stages of training due to the exploration of different actions and state trajectories. As training progresses, particularly after around 500 episodes, a noticeable upward trend in mean rewards is observed. This indicates the agent's increasing proficiency in balancing the pole.

By around 1500 episodes, the agent demonstrates robust performance, with mean rewards frequently approaching higher values. Despite occasional fluctuations, the overall trend indicates that the agent has effectively learned a policy to balance the pole, achieving consistently high rewards.

The same can be found in Figure 2.

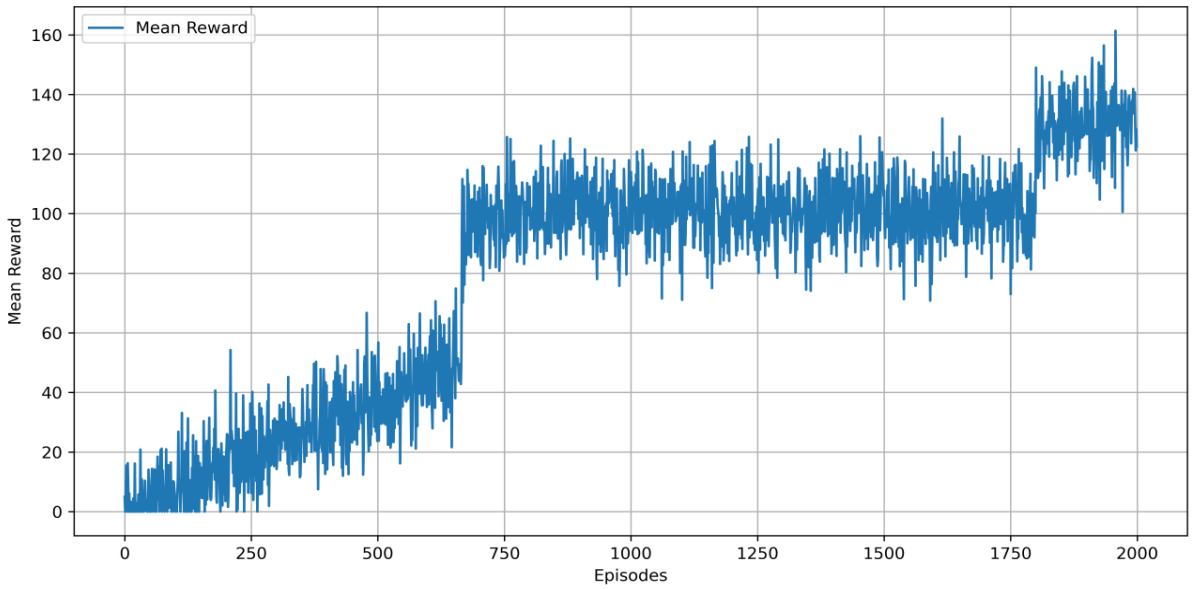

Figure 2 - Mean Rewards vs Episodes (Cart Pole – Deep Q Learning + PQC Function Approximators)

A3. Cart Pole – Baseline Implementation

As a baseline for comparison, we implemented the classic Cart-Pole system using the OpenAI Gym framework. The Cart-Pole environment is a fundamental benchmark problem in reinforcement learning, where the goal is to balance a pole attached to a moving cart by applying forces to the cart.

The dynamics of the Cart-Pole system are governed by the physics of the pendulum and cart interaction. At each timestep, the agent selects an action that results in applying a force to the cart. The environment then updates the state variables based on the following equations of motion:

1. **Force Calculation:** Depending on the action, a force of a fixed magnitude is applied to the cart.

2. **Acceleration Calculation:** The cart's acceleration and the pole's angular acceleration are computed based on the current state and applied force.

3. **State Update:** Using the Euler method or semi-implicit Euler method, the cart's position and velocity, as well as the pole's angle and angular velocity, are updated.

The environment's `step` function performs these calculations and returns the new state, reward, a Boolean indicating whether the episode has terminated, and additional information.

Interaction with the Environment:

1. **Initialization:** The environment initializes its parameters, including gravity, mass of the cart and pole, length of the pole, and force magnitude. The observation and action spaces are defined, and the initial state is randomized within specified bounds.

2. **Stepping:** The `step` function takes an action as input, computes the resultant force, updates the state variables, checks for termination conditions, and returns the new state and reward.

3. **Resetting:** The `reset` function reinitializes the state to a random configuration within the specified bounds.

4. **Rendering:** The environment provides a rendering function to visualize the Cart-Pole system. This function creates a graphical representation of the cart, pole, and track, and updates the visualization at each timestep.

Experimental Setup and Performance Evaluation:

The performance of the agent is evaluated through iterative training and parameter updates. The training process involves:

1. **Data Collection**: Collecting trajectories by running the agent in the environment and recording the states, actions, rewards, and next states.

2. **Training:** Using the collected data to train a neural network that approximates the Q-function, which predicts the expected cumulative rewards for state-action pairs.

3. **Evaluation:** Periodically evaluating the agent's performance by running the environment and recording the average reward over multiple episodes.

The agent's performance is tracked and plotted to visualize the learning progress. The plot of "Average Reward vs. Iteration" demonstrates how the agent improves its policy over time, gradually achieving higher average rewards as training progresses.

Initially, the agent exhibits low and highly variable performance, with average rewards fluctuating significantly. This variability is expected as the agent explores different actions and state trajectories to learn an effective policy. The learning curve shows a slow start, indicating the agent's initial attempts to understand the environment dynamics and the effect of its actions.

As training progresses, particularly after around 200 iterations, there is a noticeable improvement in performance. The average rewards begin to increase steadily, reflecting the agent's improving ability to keep the pole balanced. This phase represents the agent's learning process, where it starts to identify and exploit effective actions to maximize the reward.

Around 400 iterations, the agent's performance shows significant improvement, with the average reward consistently rising and approaching the maximum possible value. This indicates that the agent is successfully learning to maintain the pole in an upright position for longer durations. The fluctuations in reward values decrease, and the overall trend becomes more stable.

From around 600 iterations onwards, the agent consistently achieves high average rewards, often nearing the maximum reward of 200. This demonstrates that the agent has effectively learned a policy to balance the pole, resulting in sustained high performance. The occasional dips in reward reflect the inherent exploration process or occasional failures, but the overall trend remains high.

The graph (Figure 3) clearly shows that by the end of the training, the agent has achieved a robust policy for balancing the pole, as indicated by the high and consistent average rewards. The upward trajectory of the learning curve demonstrates the agent's successful learning and adaptation capabilities in the Cart-Pole environment.

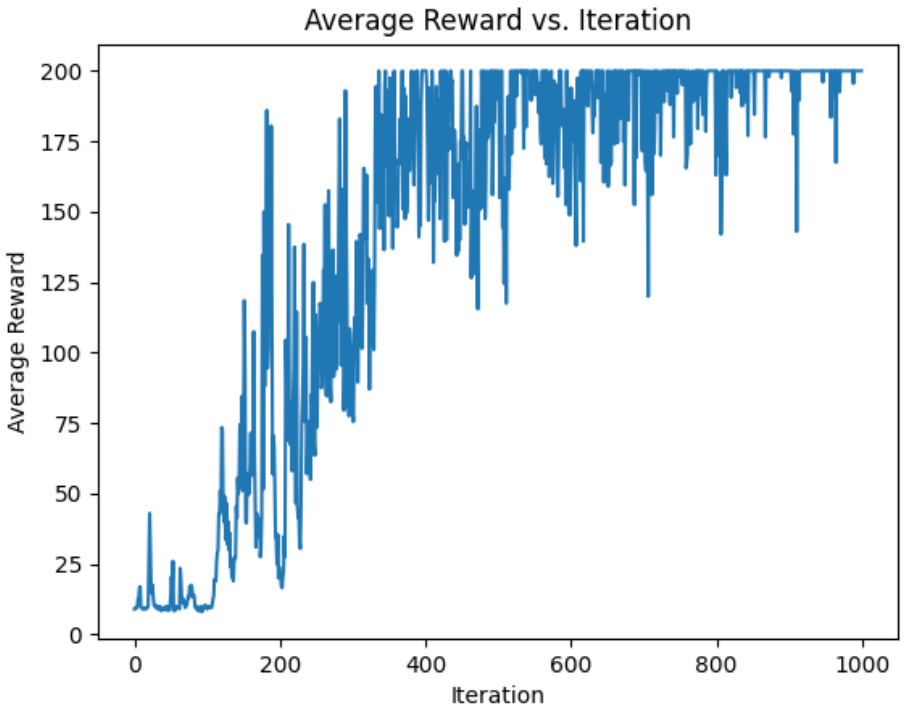

Figure 3 – Mean Rewards vs Episodes (Cart Pole – DDQN)

B. Mountain Car

State Space

Num	Observation	Min	Max	Unit
0	position of the car along the x-axis	-Inf	Inf	position (m)
1	velocity of the car	-Inf	Inf	position (m)

Action Space

Num	Observation	Value	Unit
0	Accelerate to the left	Inf	position (m)
1	Don’t accelerate	Inf	position (m)
2	Accelerate to the right	Inf	position (m)

B1. Mountain Car – Policy Gradient-based RL + PQC Policies

As we tackle the Mountain Car environment from the OpenAI Gym, which is a classic control problem that has been widely studied in the RL literature, let’s understand the scenario a bit. The Mountain Car environment involves a car situated in a valley between two hills. The goal is to drive the car up one of the hills by applying appropriate force inputs to the car. However, the car's engine is not powerful enough to scale the hill directly from the valley, so it must gain momentum by

oscillating back and forth to build up the required energy. This environment presents a challenging task for RL agents due to the non-linear dynamics and the need for long-term planning and exploration.

This approach leverages the expressive power of quantum circuits to represent the policy network. We employ a re-uploading PQC architecture, which consists of interleaved layers of parameterized quantum operations and classical non-linearities. The quantum circuit is constructed using single-qubit rotations and entangling layers, with the circuit parameters and input state being jointly optimized during training.

The re-uploading PQC layer is defined as a custom TensorFlow layer, which takes as input the current state of the environment and produces a probability distribution over the possible actions. The circuit parameters and input state are encoded as trainable variables, with the latter being processed through a non-linear activation function. The output of the quantum circuit is then mapped to action probabilities using a linear layer followed by a SoftMax activation.

We train the agent using the REINFORCE algorithm, which is a policy gradient method that directly optimizes the policy parameters to maximize the expected return. The algorithm proceeds in an episodic manner, where multiple episodes are collected in parallel and used to compute the gradients of the policy parameters with respect to the cumulative rewards. These gradients are then applied to update the trainable variables using an Adam optimizer.

The training process involves gathering episodes by interacting with the Mountain Car environment using the current policy. For each episode, the agent's states, actions, and rewards are recorded in trajectories. The returns for each timestep are computed using exponentially decaying rewards, and these returns are used as weights in the REINFORCE loss function. The gradients of the loss with respect to the trainable variables are computed using automatic differentiation and applied to update the parameters of the quantum circuit and the input state encoding.

The performance of the agent is evaluated by plotting the mean rewards per episode over the course of training. As shown in the graph (Figure 4), the agent initially exhibits highly variable performance, with episodes yielding both positive and negative rewards. However, as training progresses, the agent gradually learns to navigate the Mountain Car environment more effectively, and the mean rewards steadily improve.

It is noteworthy that the graph displays a characteristic oscillatory pattern, with periods of improvement interspersed with temporary dips in performance. This behaviour is expected in complex RL environments, as the agent occasionally encounters states or situations that temporarily hinder its progress, leading to a temporary decrease in rewards. However, the overall trend demonstrates the agent's ability to learn an effective policy for solving the Mountain Car task using the quantum circuit-based approach.

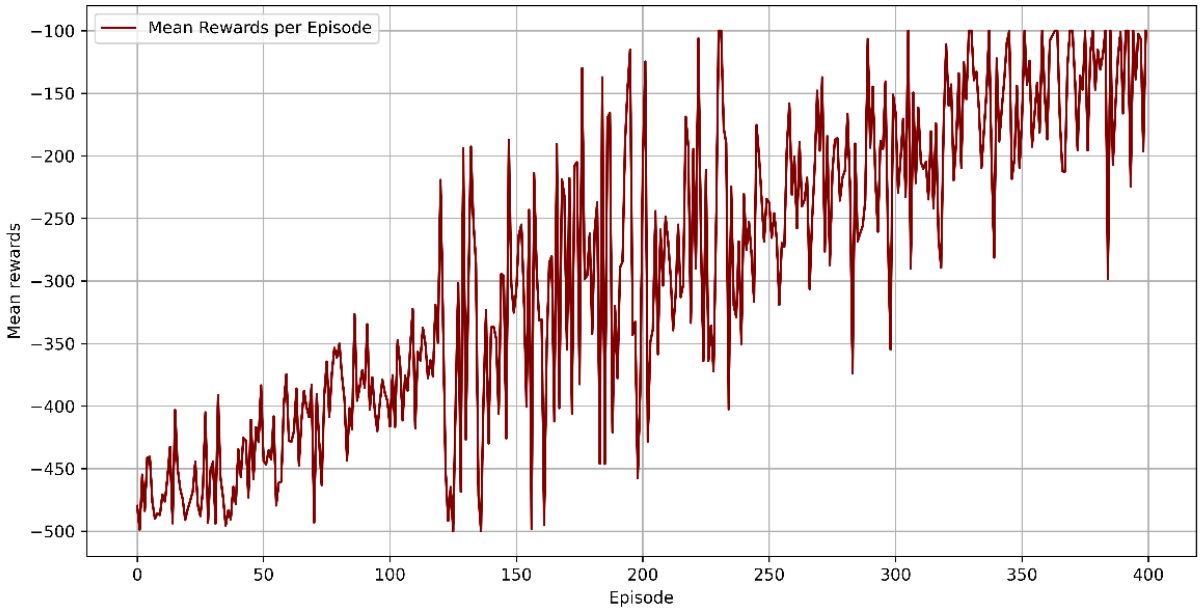

Figure 4 – Mean Rewards vs Episodes (Mountain Car – Policy Gradient + PQC π)

B2. Mountain Car – Deep Q Learning + PQC Function Approximators

This approach leverages the expressive power and potential quantum advantages offered by PQCs to approximate the Q-function, which maps state-action pairs to their expected cumulative rewards. We employ a data re-uploading PQC architecture, consisting of interleaved layers of parameterized quantum operations and classical non-linearities. The quantum circuit is constructed using single-qubit rotations and entangling layers, with the circuit parameters being optimized during training.

The PQC layer is defined as a custom TensorFlow layer, which takes as input the current state of the environment and produces a vector of Q-values, one for each possible action. The circuit parameters are encoded as trainable variables, and the output of the quantum circuit is processed through a rescaling layer to obtain the final Q-value estimates.

We train the agent using the Q-learning algorithm, which iteratively updates the Q-function by minimizing the temporal difference error between the predicted Q-values and the target Q-values computed from the observed rewards and the estimated future returns. The agent interacts with the Mountain Car environment in an episodic manner, where experiences are stored in a replay memory buffer. During training, batches of experiences are sampled from the replay memory, and the Q-function is updated based on the sampled transitions.

The training process involves several key components:

1. **Interaction with the environment:** At each step, the agent interacts with the Mountain Car environment by selecting an action based on an epsilon-greedy policy. The selected action is either the one with the highest predicted Q-value (exploitation) or a random action (exploration), determined by a coin flip with probability epsilon. The agent receives a reward and the next state after executing the chosen action.

2. **Experience replay:** The transition tuple, consisting of the current state, action, reward, next state, and a flag indicating whether the episode terminated, is stored in the replay memory buffer. This buffer allows the agent to learn from past experiences, improving data efficiency and stabilizing the training process.

3. **Q-function update:** At regular intervals, a batch of transitions is sampled from the replay memory, and the Q-function is updated using the Q-learning update rule. The target Q-values are computed by taking the maximum Q-value of the next state and adding the immediate reward. The Q-function parameters are then optimized to minimize the Huber loss between the predicted Q-values and the target Q-values, using backpropagation and an Adam optimizer.

4. **Exploration and exploitation trade-off:** The epsilon parameter, which determines the probability of selecting a random action (exploration) versus the action with the highest predicted Q-value (exploitation), is initially set to a high value and gradually decayed over time. This approach ensures sufficient exploration early in the training process and transitions towards more exploitation as the agent learns an effective policy.

The performance of the agent is evaluated by plotting the mean rewards per episode over the course of training. As shown in the graph (Figure 5) the agent initially exhibits highly variable performance, with episodes yielding both positive and negative rewards. This behaviour is expected in complex RL environments, where the agent must explore various action sequences and state trajectories to discover an effective policy.

As training progresses, the agent gradually learns to navigate the Mountain Car environment more effectively, and the mean rewards improve. However, just as in the previous case, the graph displays a characteristic oscillatory pattern, with periods of improvement interspersed with temporary dips in performance. This behaviour is common in RL tasks, as the agent occasionally encounters states or situations that temporarily hinder its progress, leading to a temporary decrease in rewards. Nevertheless, the overall trend demonstrates the agent's ability to learn an effective Q-function and policy for solving the Mountain Car task using the PQC-based function approximation approach.

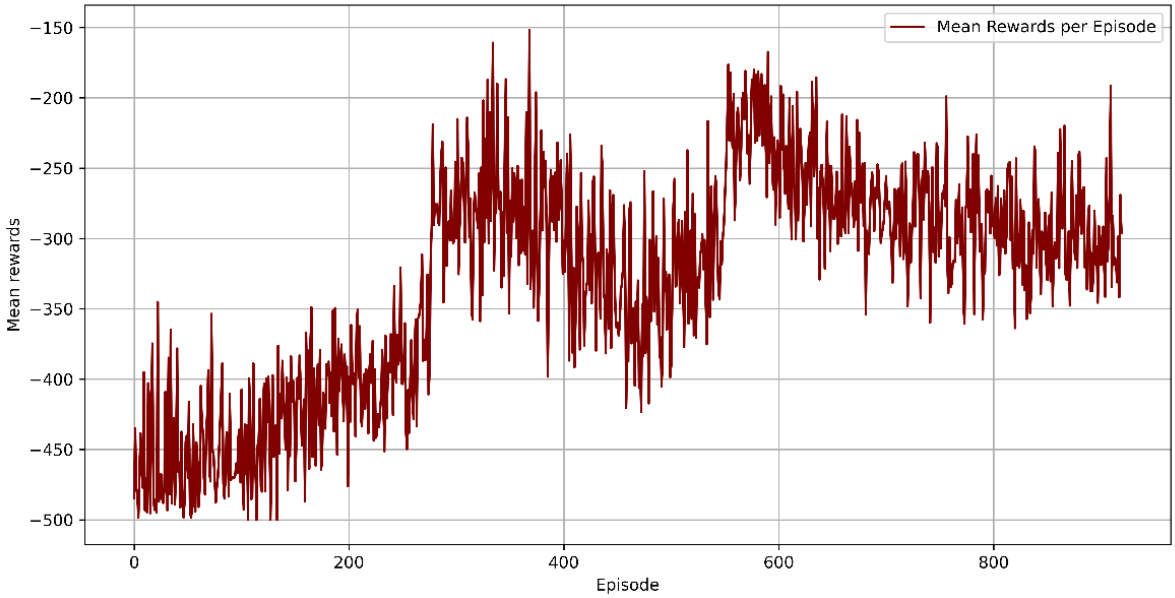

Figure 5 - Mean Rewards vs Episodes (Mountain Car – Deep Q Learning + PQC Function Approximators)

B3. Mountain Car – Baseline

As a baseline for comparison, we implemented a Deep Q-Network (DQN) to approximate the Q-function which maps state-action pairs to their expected cumulative rewards. The Q-function is represented by a neural network, trained using experience replay and a target network to enhance training stability. The agent interacts with the MountainCar-v0 environment, balancing exploration and exploitation to learn an effective policy.

The DQN agent employs a neural network comprising two hidden layers with 400 and 300 neurons, respectively, followed by an output layer with linear activation. The neural network takes the current state as input and produces Q-values for each possible action. The parameters of the network are optimized during training to minimize the mean squared error (MSE) loss between the predicted Q-values and the target Q-values. The model is compiled using the Adam optimizer with a learning rate of 0.005.

The agent interacts with the MountainCar-v0 environment episodically, with each episode consisting of a sequence of states, actions, and rewards. The main components of the implementation include:

1. **Environment Interaction:** At each timestep, the agent selects an action based on an epsilon-greedy policy, balancing exploration and exploitation. The environment returns the next state, reward, and a termination flag upon executing the selected action. The epsilon-greedy policy involves selecting a random action with probability epsilon (exploration) or the action with the highest predicted Q-value (exploitation).

2. **Experience Replay:** Transition tuples (current state, action, reward, next state, done) are stored in a replay memory buffer, which allows the agent to learn from past experiences. This improves data efficiency and stabilizes the training process by breaking the correlation between consecutive experiences.

3. **Q-Function Update:** Periodically, a batch of transitions is sampled from the replay memory, and the Q-function is updated using the Q-learning update rule. Target Q-values are computed by adding the immediate reward to the discounted maximum Q-value of the next state. The network parameters are then optimized to minimize the MSE loss between the predicted and target Q-values. This process involves predicting the action values for the current and next states and updating the Q-values for the selected actions.

4. **Exploration-Exploitation Trade-off:** The epsilon parameter, which determines the probability of selecting a random action versus the action with the highest predicted Q-value, is initially set to a high value and gradually decays over time. This approach ensures sufficient exploration during the early stages of training and transitions to more exploitation as the agent learns an effective policy. Epsilon is decayed by a factor of 0.99 each episode, ensuring it does not fall below a minimum threshold.

The agent's performance is evaluated by tracking and printing episodic rewards. The trained models are saved whenever the agent achieves the highest rewards. Initially, the agent exhibits variable performance, with rewards fluctuating between positive and negative values. As training progresses, the agent learns to perform better in the environment, leading to an overall improvement

in mean rewards. This trend demonstrates the agent's capability to learn an effective Q-function and policy for the MountainCar-v0 task using the DQN approach.

The process of training the DQN agent involves crucial components such as environment interaction, experience replay, Q-function updates, and managing the exploration-exploitation trade-off. During training, if the agent successfully reaches the goal state within the allowed steps, a significant reward is given to reinforce this successful behaviour. If the agent fails, rewards are given proportionally based on the distance travelled and the velocity of the car. The same can be observed in the following Figure 6.

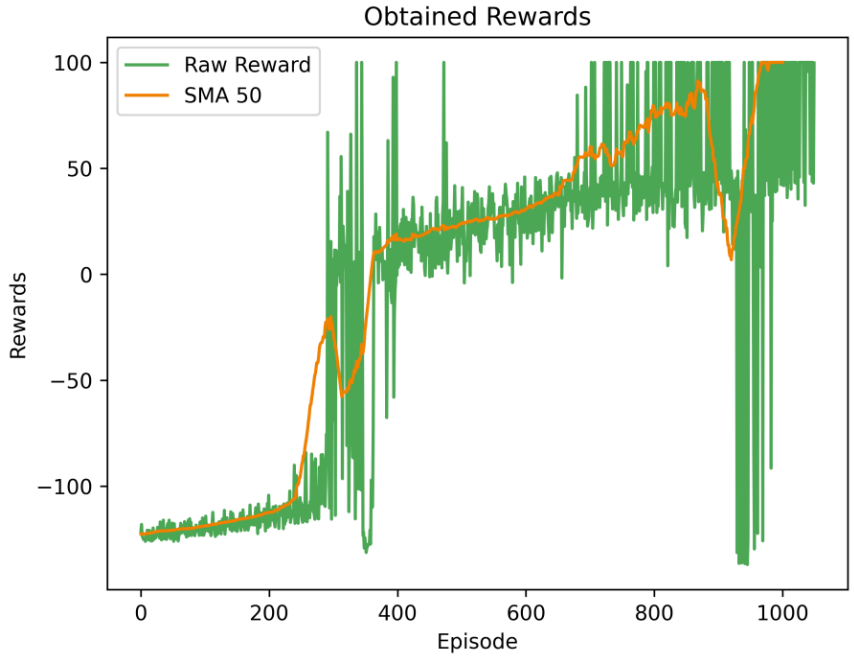

Figure 6 - Rewards and Simple Moving Average (SMA) of 50 rewards vs Episodes

C. Acrobot

State Space

Num	Observation	Min	Max
0	Cosine of `theta1`	-1	1
1	Sine of `theta1`	-1	1
2	Cosine of `theta2`	-1	1
3	Sine of `theta2`	-1	1
4	Angular velocity of `theta1`	~ -12.567 (-4 * pi)	~ 12.567 (4 * pi)
5	Angular velocity of `theta2`	~ -28.274 (-9 * pi)	~ 28.274 (9 * pi)

Action Space

Num	Action	Unit
0	apply -1 torque to the actuated joint	torque (N m)
1	apply 0 torque to the actuated joint	torque (N m)
2	apply 1 torque to the actuated joint	torque (N m)

C1. Acrobot – Policy Gradient-based RL + PQC Policies

The Acrobot environment presents a formidable challenge, requiring precise control of a double-jointed robot arm to swing up and balance at an inverted position. This task is particularly demanding due to its high-dimensional state space (of size 6) and non-linear dynamics. The implementation is the same as the previous agents as well, just to provide a gist, the following is the methodology.

Our approach harnesses the expressive capabilities and potential quantum advantages of Parameterized Quantum Circuits (PQCs) to represent the policy network. These PQCs map the environment's current state to a probability distribution over available actions. We adopt a re-uploading PQC architecture, comprising interleaved layers of parameterized quantum operations and classical non-linearities. The quantum circuit, constructed using single-qubit rotations and entangling layers, undergoes joint optimization of circuit parameters and input state during training.

Implemented as a custom TensorFlow layer, the re-uploading PQC layer takes the environment's current state as input and outputs a probability distribution over possible actions. Circuit parameters and input state are encoded as trainable variables, with the latter processed through a non-linear activation function. Output from the quantum circuit is then mapped to action probabilities via a linear layer followed by a SoftMax activation.

Training employs the REINFORCE algorithm, a policy gradient method optimizing policy parameters to maximize expected return. Training occurs episodically, with multiple parallel episodes used to compute gradients of policy parameters relative to cumulative rewards. These gradients update trainable variables using an Adam optimizer.

During training, episodes are gathered by interacting with the Acrobot environment using the current policy. Agent states, actions, and rewards are recorded in trajectories. Returns for each timestep, computed using exponentially decaying rewards, weight the REINFORCE loss function. Gradients of this loss update trainable variables via automatic differentiation, optimizing both quantum circuit parameters and input state encoding.

Initially, the agent's performance fluctuates widely, reflecting the exploration of action sequences and state trajectories. Over time, the agent learns to swing the Acrobot to the inverted position, leading to improved mean rewards. However, an oscillatory pattern emerges, with occasional dips caused by encountering states hindering progress temporarily.

Though noise may obscure the true trend, the overall plot demonstrates the agent's ability to learn an effective policy for the Acrobot task. Our work underscores the potential of quantum-

enhanced reinforcement learning in complex decision-making tasks, showcasing the efficacy of parameterized quantum circuits in such environments. The same can be observed in Figure 7.

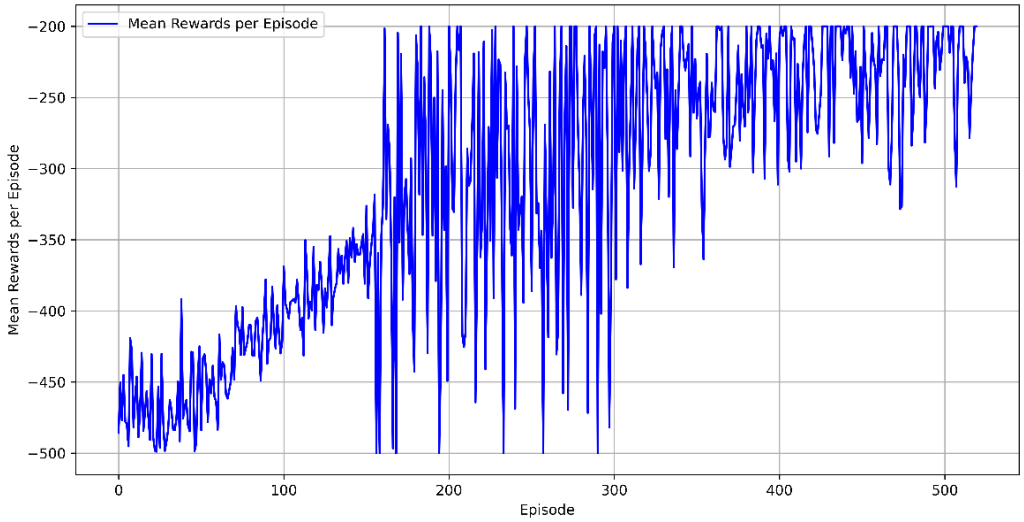

Figure 7 - Mean Rewards vs Episodes (Acrobot – Policy Gradient + PQC π)

C2. Acrobot – Deep Q Learning + PQC Function Approximators

In this implementation, the task is tackled using Deep Q-Learning with Parameterized Quantum Circuits (PQCs) as function approximators. Our strategy capitalizes on the expressive capabilities and theoretical advantages inherent in PQCs to approximate the Q-function, pivotal in mapping state-action pairs to their respective expected cumulative rewards. We adopt a data re-uploading PQC architecture, comprising interleaved layers of parameterized quantum operations and classical non-linearities. The quantum circuit, crafted through single-qubit rotations and entangling layers, undergoes optimization of its parameters during training. The implementation is the same as the previous ones, just to offer a gist, the following is the methodology.

Functioning as a custom TensorFlow layer, the PQC layer receives the current environmental state as input and yields a vector of Q-values, each corresponding to a possible action. The circuit parameters, encoded as trainable variables, drive this process, with the output of the quantum circuit further refined through a rescaling layer to furnish final Q-value estimates.

Our agent is trained utilizing the Q-learning algorithm, iteratively refining the Q-function by minimizing the temporal difference error between predicted and target Q-values, computed from observed rewards and estimated future returns. Interaction with the Acrobot environment transpires episodically, with experiences stored in a replay memory buffer. Periodically, batches of experiences are sampled from this buffer, informing updates to the Q-function.

Key facets of the training regimen encompass:

1. **Interaction with the environment:** At each step, the agent interacts with the Acrobot environment, selecting actions based on an epsilon-greedy policy, balancing between exploitation of known actions and exploration of new ones.

2. **Experience replay:** Transition tuples, encapsulating the current state, action, reward, next state, and termination flag, are catalogued in a replay memory buffer, fostering learning from past experiences.

3. **Q-function update:** Batched transitions from the replay memory inform updates to the Q-function, optimizing parameters via backpropagation and an Adam optimizer to minimize the Huber loss between predicted and target Q-values.

4. **Target network:** Enhancing stability, a separate target Q-network, periodically syncs with the primary Q-network weights, mitigating correlations between target and predicted Q-values.

5. **Exploration and exploitation trade-off:** The epsilon parameter governs the balance between exploration and exploitation, initially favouring exploration and gradually transitioning to exploitation as the agent refines its policy.

The evaluation of agent performance entails plotting mean rewards per episode over training epochs. Initial variability in performance gives way to gradual improvement, albeit punctuated by periodic dips—a characteristic trait of RL tasks beset by temporary setbacks. Thus, our agent showcases iterative Q-function refinement, offering promising prospects for quantum-enhanced reinforcement learning in complex decision-making domains. The same can be observed in Figure 8.

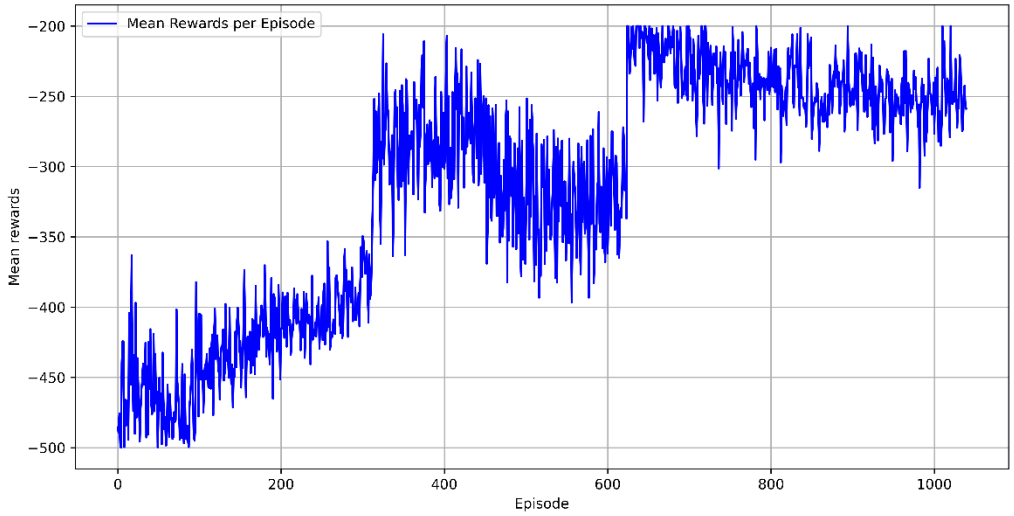

Figure 8 - Mean Rewards vs Episodes (Acrobot – Deep Q Learning + PQC Function Approximators)

C3. Acrobot – Baseline

To have a baseline for comparison of our quantum-based policies, we implemented the Acrobot-v1 by leveraging a deep Q-network (DQN) to approximate the Q-function which maps the state-action pairs to their expected cumulative rewards. The Q-function is represented by a neural network trained using experience replay and a target network to enhance training stability. The agent interacts with the Acrobot-v1 environment, balancing exploration and exploitation to learn an effective policy.

The DQN agent employs a neural network with two hidden layers of 24 neurons each, followed by an output layer with linear activation. The network takes the current state as input and produces Q-values for each possible action. The network parameters are optimized during training to minimize the mean squared error (MSE) loss between the predicted Q-values and the target Q-values.

The agent interacts with the Acrobot-v1 environment episodically, with each episode consisting of a sequence of states, actions, and rewards. The main components of the implementation include:

1. **Environment Interaction**: At each timestep, the agent selects an action based on an epsilon-greedy policy, balancing exploration and exploitation. The environment returns the next state, reward, and a termination flag upon executing the selected action.

2. **Experience Replay**: Transition tuples (current state, action, reward, next state, done) are stored in a replay memory buffer, which enables the agent to learn from past experiences. This enhances data efficiency and stabilizes the training process.

3. **Q-Function Update**: Periodically, a batch of transitions is sampled from the replay memory, and the Q-function is updated using the Q-learning update rule. Target Q-values are computed by adding the immediate reward to the discounted maximum Q-value of the next state. The network parameters are optimized to minimize the MSE loss between the predicted and target Q-values.

4. **Exploration-Exploitation Trade-off**: The epsilon parameter, determining the probability of selecting a random action (exploration) versus the action with the highest predicted Q-value (exploitation), is initially high and gradually decays over time. This ensures sufficient exploration during early training stages and transitions to more exploitation as the agent learns an effective policy.

The agent's performance is evaluated by plotting the accumulated episodic rewards over the course of training. The graph, "Accumulated episodic reward vs. num. of episodes," depicts the accumulated episodic rewards (light blue) and the average accumulated episodic rewards (dark blue) against the number of episodes.

Initially, the agent exhibits highly variable performance, with rewards fluctuating significantly. This variability is expected as the agent explores different actions and state trajectories to learn an effective policy. As training progresses, there is a noticeable upward trend in the accumulated episodic rewards, indicating that the agent is learning to perform better in the environment.

By around 400 episodes, the agent begins to stabilize, showing a more consistent improvement in performance. The average accumulated episodic reward continues to rise, reflecting the agent's increasing proficiency in navigating the Acrobot-v1 environment. Although there are still fluctuations, the overall trend is positive, demonstrating that the agent is effectively learning and improving its policy over time.

The performance graph highlights the agent's capability to learn an effective Q-function and policy for the Acrobot-v1 task using the DQN approach. The characteristic oscillations in the accumulated episodic rewards are typical in reinforcement learning tasks, where the agent occasionally encounters challenging states or situations, leading to temporary performance dips. Nonetheless, the overall upward trend signifies successful learning and adaptation by the DQN agent.

This implementation showcases the effectiveness of the DQN approach in solving the Acrobot-v1 environment by leveraging neural networks, experience replay, and a target network.

The training process involves crucial components such as environment interaction, experience replay, Q-function updates, and managing the exploration-exploitation trade-off. The results, as visualized in the performance evaluation graph, indicate that the agent can successfully learn to navigate the environment, demonstrating the potential of DQN for similar reinforcement learning tasks. Through iterative training and parameter adjustments, the agent's performance improves, highlighting the robustness and adaptability of the DQN methodology in complex reinforcement learning environments. The same can be observed in Figure 9.

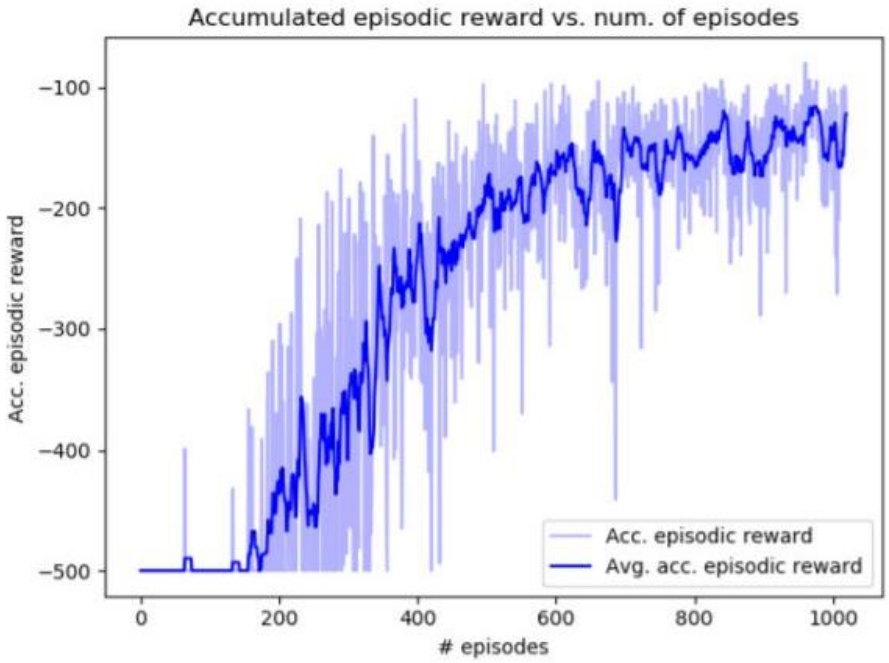

Figure 9 - The accumulated episodic reward and its average over the most recent 10 episodes for the DQN model for the Acrobot task (Mean Accumulated Rewards vs Episodes)

D. Comparison

We finally compare and contrast the three reinforcement learning (RL) methods applied to all three Cart Pole, Mountain Car and Acrobot environments: Policy Gradient-based RL with PQC Policies, Deep Q-Learning with PQC Function Approximators, and the Baseline Deep Q-Network (DQN) / Double Deep Q-Network (DDQN). We analyze the efficacy, computational complexity, and practical implications of each approach to determine the viability and advantages of using quantum computing in RL. Let's begin!

D1. Cart Pole

A1. Cart Pole – Policy Gradient-based RL + PQC Policies

Overview:

- **Method:** Uses Parameterized Quantum Circuits (PQCs) to directly optimize the policy network, mapping the current state to a probability distribution over actions.
- **Architecture:** Combines classical neural network layers with a PQC layer.
- **Training:** Utilizes the Policy Gradient method (REINFORCE algorithm) with an Adam optimizer.

Complexity:

- **Time Complexity:** High due to joint optimization of quantum circuit parameters and state encoding, along with the episodic nature of policy gradient methods.
- **Space Complexity:** Significant, given the need to store quantum circuit parameters and classical network weights.

Performance:

- **Variability:** Initial performance is highly variable due to extensive exploration.
- **Improvement:** Gradual improvement with an upward trend in rewards, indicating effective learning.
- **Quantum Advantage:** The expressive power of PQCs may enhance the policy network's performance by capturing complex patterns more efficiently.

A2. Cart Pole – Deep Q-Learning + PQC Function Approximators

Overview:

- **Method:** Uses PQCs to approximate the Q-function, mapping state-action pairs to expected rewards.
- **Architecture:** Integrates classical neural network layers with a PQC layer.
- **Training:** Q-learning algorithm with experience replay and target networks to stabilize learning.

Complexity:

- **Time Complexity:** Moderate to high, due to iterative updates of the Q-function and periodic synchronization with the target network.
- **Space Complexity:** Significant, comparable to classical DQN, with additional overhead from quantum circuit parameters.

Performance:

- **Variability:** Initial variability with gradual stabilization over time.
- **Improvement:** Mean rewards improve with training, demonstrating effective Q-function approximation.
- **Quantum Advantage:** Quantum-enhanced function approximation shows promising potential for complex RL tasks, leveraging the expressive power of PQCs.

A3. Cart Pole – Baseline Deep Q-Network (DQN)

Overview:

- **Method:** Uses a classical neural network to approximate the Q-function.
- **Architecture:** Neural network with two hidden layers (400 and 300 neurons) and a linear output layer.
- **Training:** DQN algorithm with experience replay and target networks.

Complexity:

- **Time Complexity:** Moderate, determined by the depth of the neural network and batch size for updates.
- **Space Complexity:** Relatively lower compared to PQC methods, limited to classical neural network weights.

Performance:

- **Variability:** Typical RL learning curve with initial variability and eventual stabilization.
- **Improvement:** Effective in learning and improving the control policy for the Cart Pole task.
- **Robustness:** Well-understood and widely used, making it a robust benchmark.

D2. Mountain Car

B1. Mountain Car – Policy Gradient-based RL + PQC Policies

Overview:

- **Method:** Uses Parameterized Quantum Circuits (PQCs) to represent the policy network, mapping the current state to a probability distribution over actions.
- **Architecture:** Re-uploading PQC with interleaved layers of quantum operations and classical non-linearities.
- **Training:** Utilizes the REINFORCE algorithm, a policy gradient method optimized with an Adam optimizer.

Complexity:

- **Time Complexity:** High due to joint optimization of quantum circuit parameters and state encoding, as well as the episodic nature of REINFORCE.
- **Space Complexity:** Significant, given the need to store quantum circuit parameters and classical network weights.

Performance:

- **Variability:** Initial performance is highly variable due to extensive exploration.
- **Improvement:** Gradual improvement with characteristic oscillatory patterns indicating the typical learning process of RL agents.
- **Quantum Advantage:** Potential quantum advantages in expressiveness may enhance the policy network's performance.

B2. Mountain Car – Deep Q-Learning + PQC Function Approximators

Overview:

- **Method:** Uses PQCs to approximate the Q-function, mapping state-action pairs to expected rewards.
- **Architecture:** Data re-uploading PQC with single-qubit rotations and entangling layers.
- **Training:** Q-learning algorithm with experience replay and target networks to stabilize learning.

Complexity:

- **Time Complexity:** Moderate to high, due to iterative updates of the Q-function and periodic synchronization with the target network.
- **Space Complexity:** Significant, comparable to classical DQN, with additional overhead from quantum circuit parameters.

Performance:

- **Variability:** Initial variability with gradual stabilization over time.
- **Improvement:** Mean rewards improve with training, demonstrating effective exploration-exploitation balance.
- **Quantum Advantage:** Quantum-enhanced function approximation shows promising potential for complex RL tasks.

B3. Mountain Car – Baseline Deep Q-Network (DQN)

Overview:

- **Method:** Uses a classical neural network to approximate the Q-function.
- **Architecture:** Neural network with two hidden layers (400 and 300 neurons) and a linear output layer.
- **Training:** DQN algorithm with experience replay and target networks.

Complexity:

- **Time Complexity:** Moderate, determined by the depth of the neural network and batch size for updates.
- **Space Complexity:** Relatively lower compared to PQC methods, limited to classical neural network weights.

Performance:

- **Variability:** Typical RL learning curve with initial variability and eventual stabilization.
- **Improvement:** Effective in learning and improving the control policy for the Mountain Car task.
- **Robustness:** Well-understood and widely used, making it a robust benchmark.

D3. Acrobot

C1. Acrobot – Policy Gradient-based RL + PQC Policies

Overview:

- **Method:** Uses Parameterized Quantum Circuits (PQCs) to represent the policy network, which maps the current state to a probability distribution over actions.
- **Architecture:** Re-uploading PQC with interleaved layers of quantum operations and classical non-linearities.
- **Training:** Utilizes the REINFORCE algorithm, an episodic policy gradient method optimized with an Adam optimizer.

Complexity:

- **Time Complexity:** High due to the joint optimization of quantum circuit parameters and state encoding, and the episodic nature of REINFORCE.
- **Space Complexity:** Significant, given the need to store and process quantum circuit parameters and classical network weights.

Performance:

- Initial performance is highly variable due to extensive exploration.
- Gradual improvement with oscillatory patterns indicating the typical learning process of RL agents.
- Quantum advantages potentially enhance the expressiveness of the policy network.

C2. Acrobot – Deep Q-Learning + PQC Function Approximators

Overview:

- **Method:** Employs PQCs to approximate the Q-function, mapping state-action pairs to expected rewards.
- **Architecture:** Data re-uploading PQC with single-qubit rotations and entangling layers.
- **Training:** Q-learning algorithm with experience replay and target networks to stabilize learning.

Complexity:

- **Time Complexity:** Moderate to high, due to iterative updates of the Q-function and periodic synchronization with the target network.
- **Space Complexity:** Significant, but comparable to classical DQN, with additional overhead from quantum circuit parameters.

Performance:

- Exhibits initial variability, stabilizing over time with improvements in mean rewards.
- Incorporates effective exploration-exploitation balance using epsilon-greedy policies.
- Quantum-enhanced function approximation shows promising potential for complex RL tasks.

C3. Acrobot – Baseline Deep Q-Network (DQN)

Overview:

- **Method:** Uses a classical neural network to approximate the Q-function.
- **Architecture:** Neural network with two hidden layers and a linear output layer.
- **Training:** DQN algorithm with experience replay and target networks.

Complexity:

- **Time Complexity:** Moderate, determined by the depth of the neural network and batch size for updates.
- **Space Complexity:** Relatively lower compared to PQC methods, limited to classical neural network weights.

Performance:

- Demonstrates typical RL learning curve with initial variability and eventual stabilization.
- Effective in learning and improving the control policy for the Acrobot task.
- Well-understood and widely used, making it a robust benchmark.

Final Comparative Analysis

Advantages of Quantum Computing in RL:

- **Expressive Power:** PQCs can capture complex patterns and correlations potentially more efficiently than classical networks.
- **Potential Speedups:** Theoretically, quantum circuits can provide computational speedups for certain tasks, though practical advantages are still under investigation.

Challenges and Limitations:

- **Computational Overhead:** Quantum methods require significant computational resources, both in terms of time and memory.
- **Current Technological Limitations:** Quantum hardware is still in its infancy, limiting the practical implementation and scalability of PQC-based approaches.

Time and Space Complexity:

- **PQC Methods:** Higher time complexity due to the optimization of quantum circuits and state encoding. Space complexity is also higher, given the need to handle quantum and classical parameters.
- **Classical DQN:** More efficient in terms of computational resources, with well-established training procedures and lower overhead compared to PQC methods.

V. CONCLUSION

Quantum computing in reinforcement learning (RL) illuminates a frontier of untapped potential, especially through the rich expressiveness of Parameterized Quantum Circuits (PQCs). However, this burgeoning field faces significant hurdles, such as elevated computational overhead and current practical constraints. In contrast, classical methods like Deep Q-Learning (DQN) remain steadfast and effective, serving as a reliable benchmark for comparison.

As quantum hardware continues its rapid evolution, the chasm between theoretical promise and practical application is poised to diminish. This progression heralds an era where quantum-enhanced RL could revolutionize complex decision-making tasks. The fusion of efficient quantum hardware and advanced pattern-recognizing quantum methodologies could propel RL into unprecedented realms of capability.

Quantum Reinforcement Learning stands on the cusp of transforming the technological landscape, poised to usher in a new age of innovation. As we look to the future, the integration of quantum computing in RL holds the promise of creating a world where decision-making is not only more sophisticated but also remarkably efficient, paving the way for breakthroughs that could redefine our interaction with technology.

GitHub Repository - Quantum_RL-Arshia_S-&-Rahath_M

REFERENCES

1. Broughton, M., Verdon, G., McCourt, T., Martinez, A. J., Yoo, J. H., Isakov, S. V., Massey, P., Halavati, R., Niu, M. Y., Zlokapa, A., Peters, E., Lockwood, O., Skolik, A., Jerbi, S., Dunjko, V., Leib, M., Streif, M., Von Dollen, D., Chen, H., … Mohseni, M. (2021). *TensorFlow Quantum: A Software Framework for Quantum Machine Learning* (arXiv:2003.02989; Version 2). arXiv. http://arxiv.org/abs/2003.02989

2. Jerbi, S., Gyurik, C., Marshall, S. C., Briegel, H. J., & Dunjko, V. (2021). *Parametrized quantum policies for reinforcement learning* (arXiv:2103.05577). arXiv. http://arxiv.org/abs/2103.05577

3. Neumann, N. M. P., de Heer, P. B. U. L., & Phillipson, F. (2023). Quantum reinforcement learning. *Quantum Information Processing*, *22*(2), 125. https://doi.org/10.1007/s11128-023-03867-9

4. Skolik, A., Jerbi, S., & Dunjko, V. (2022). Quantum agents in the Gym: A variational quantum algorithm for deep Q-learning. *Quantum*, *6*, 720. https://doi.org/10.22331/q-2022-05-24-720

APPENDIX

Due to the intense mathematics and detailed reinforcement learning approaches involved in this project, a comprehensive exposition is beyond the scope of this report. However, the sources that provided insights and clarity on these concepts are listed below for reference:

1. The idea and intuition behind using PQCs for Machine Learning - Ryan Sweke - Should we use parameterized quantum circuits for machine learning? - IPAM at UCLA (youtube.com)
2. The In-depth course on Reinforcement Learning - DeepMind x UCL | Introduction to Reinforcement Learning 2015 - YouTube

These resources offer a thorough understanding of the mathematics and intuition underlying our methodology. Given the time constraints and the complexity of the mathematical details, we have opted to provide these links to help connect and deepen the theoretical insights related to our implementation. Rest assured, we understand the entire Reinforcement Learning framework in our implementation (given that we had already taken a course on Reinforcement Learning) and the requirement and utility of using PQCs to support the RL paradigm.

We extend our deepest gratitude to Prof. Nitin Upadhyaya for delivering an exceptional course with engaging methodology and pedagogy. Our heartfelt sincerest thanks to Mr. Viraj Daniel D'Souza for his unwavering support, encouragement, and motivation throughout the course and the project, without which it would've never been possible. We also express our sincere appreciation to Prof. Sandeep Manjanna for imparting foundational knowledge of Reinforcement Learning, which was instrumental in the development of this project. We are profoundly grateful for your guidance and support.

Thank you!