Quantum Cartpole: Investigating Parameterized Quantum Circuits with Policy Gradient Methods

Albert Haladay
University of Nottingham

September 2024

Abstract

This investigation explores Parameterised Quantum Circuits (PQCs) as an alternative to neural networks for policy gradient methods in Reinforcement Learning (RL). The REINFORCE, Advantage Actor-Critic (A2C), and Proximal Policy Optimisation (PPO) algorithms are implemented using both a classical neural network and a hybrid between a PQC and a neural network. These agents are trained and tested in the cartpole environment. The results show that although hybrid quantum agents have more volatile training curves, overall they perform their given tasks at a similar ability or better than their classical counterparts. These results imply that quantum reinforcement learning agents have the potential to significantly enhance autonomous agents in a variety of applications.

1 Introduction

In this project, we will develop and evaluate the performance of classical and hybrid quantum agents trained with different policy gradient methods in the cartpole environment.[1] Reinforcement Learning (RL) is an area of machine learning where an agent can learn from an environment in the form of states, actions and rewards. This area focuses on finding optimal policies that lead an agent to make the best decisions. This is a very practical area of machine learning and has many applications in AI for games[2] and all kinds of autonomous robotics, such as self-driving cars.[3] Traditionally, an RL algorithm will use a neural network to capture its policy. However, this project investigates if quantum circuits can be used to enhance RL agents. This is done by replacing the classical neural network with a hybridisation between a Parameterised Quantum Circuit (PQC) and a neural network. The hybrid between a classical neural network and PQC is known as a Hybrid Quantum Circuit (HQC). Currently, modern RL methods do not scale well with overly complex problems. This is because if the state or action space becomes too large, the computational cost of training the agent becomes unfeasible. However, the ability of quantum computing to process large amounts of data in parallel using HQCs may address this limitation.

Quantum computing is a technology that leverages superposition and entanglement to perform calculations in a fundamentally different way from classical computing. The inherent parallelism of entangled circuits holds the potential to solve problems previously thought to be uncomputable. Quantum computing can also potentially exploit its strengths to solve optimisation and machine learning tasks more efficiently than classical computers.[4]

Quantum machine learning is the study of the intersection between quantum computing and machine learning, with the goal being to harness the latent computing power in quantum circuits to improve the strength of machine learning tasks. Quantum Reinforcement Learning (QRL) is the corresponding area in quantum machine learning that this paper focuses on, this is an actively developed area of research with new techniques created every day. Similarly to traditional RL, QRL must capture a policy and to leverage the strength of quantum computers the policy must be captured using a quantum circuit. Leveraging quantum computing in this way has already been shown to increase the expressive power of a policy,[5] but our understanding of the advantages, weaknesses and potential of quantum agents is vague and problem-specific. To address this, the cartpole problem is used since it has a clear objective and is often used as a benchmark problem for RL agents.

1.1 Related Work

The performance of classical algorithms has been well-tested and have been benchmarked using problems like cartpole before.[6] Similarly, other studies have done the same for the quantum counterparts of these algorithms.[7] There is a rising need to understand how quantum circuits can be applied to RL, due to the fast development of quantum computing technology.[8] This paper aims to bridge the gap between theory and practice by implementing the agents and evaluating whether they hold a computational advantage.

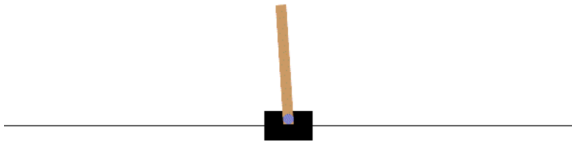


Figure 1: The cartpole environment from OpenAI Gym in Python.

1.2 The Cartpole Problem

The cartpole problem is a simple reinforcement learning environment with the goal being to balance a pole attached to a cart. The environment consists of a cart that can move in either direction on a track and a pole fixed to the cart which can rotate freely. The agent's task is to keep the pole balanced for as long as possible by moving the cart along the track.

1.2.1 Success Criterion

Since the task of any agent in the environment is to keep the pole balanced for as long as possible, the criterion for success should be for the agent to reach the maximum number of steps in an episode. For cartpole the maximum number of steps that can be taken in each episode is 500, but a single instance of the agent succeeding is not a good method to determine that the agent has finished training. To address this, the criterion for the agent to have 'solved' the cartpole problem is for the average episode length of the last 100 episodes to be greater than 475, as this implies the agent has achieved a 95% success rate in the last 100 episodes.

1.2.2 States, Actions and Rewards

The state of the system is composed of four continuous variables: cart position, cart velocity, pole angle, and pole angular velocity. The action space of the agent is discrete, being either a left or right motion. Traditionally the reward of the agent is a constant positive signal for each step the agent remains alive, however, to speed up learning and encourage the agent to remain in the centre of the track, reward shaping was implemented. This also helps to add more complexity to the problem, allowing us to compare other aspects of the resulting policy instead of just how often it succeeds. The new reward, r, is calculated as follows:

$$r = \frac{1}{1 + 0.5 \cdot position^2}$$

The *position* represents the deviation of the cart from the centre of the environment, after being normalised using the distance where the cart falls off the track.

1.3 Policy Gradient Methods

Policy gradient methods are a class of RL algorithms focusing on learning a parameterised policy that maps states to actions. The policy is learned by updating parameters to maximise actions that lead to a higher expected return, usually the discounted expected return to stabilise training. For a trajectory of length τ with rewards R_i where $i =$

$1, \ldots, \tau$, the discounted expected return is calculated as

$$G_t = \sum_{i=t}^{\tau} \gamma^{i-t} R_i \,,$$

where γ is the discount factor. This discount factor determines the influence that future rewards have at the current timestep. The behavioural implication of this is that an agent with a higher discount factor will prioritise long-term rewards.

1.3.1 REINFORCE

REINFORCE[9] is the simplest policy gradient method, which learns the optimal policy by simply updating parameters in the direction that maximises the discounted expected return. Mathematically, this is done using gradient ascent to maximise an objective function $J(\theta) = \mathbb{E}_\pi [G_t]$, where $\pi(s_t, a_t|\theta)$ is the probability of choosing action a_t in state s_t with parameters θ. The update rule to maximise the expected returns is $\theta \to \theta + \alpha \nabla_\theta J(\theta)$, where α is the learning rate. Using the property that $\nabla ln [f(x)] = f'(x)/f(x)$, the gradient of the objective function can be written as:

$$\begin{aligned}
\nabla_\theta J(\theta) &= \nabla_\theta \mathbb{E}_\pi [G_t] \\
&= \nabla_\theta \sum_{t=1}^{\tau} G_t \, \pi_\theta(a_t|s_t) \\
&= \sum_{t=1}^{\tau} G_t \, \pi(a_t|s_t) \, \nabla_\theta \, ln [\pi_\theta(a_t|s_t)] \\
&= \mathbb{E}_\pi \{ G_t \, \nabla_\theta \, ln [\pi(a_t|s_t)] \} \,.
\end{aligned}$$

Sampling this policy allows for this parameter update equation,

$$\theta \to \theta + \alpha \nabla_\theta G_t \, ln [\pi(a_t|s_t)] \,. \tag{1}$$

1.3.2 Advantage Actor Critic

The Advantage Actor-Critic[10] (A2C) is an attempt at combining PG methods with value-based methods to improve training stability. It does this by using an 'actor' to predict the policy and a 'critic' to predict the value function (an estimate of the discounted expected reward). Instead of directly updating the policy in an unstable manner with the high variance of the discounted expected reward, the policy is updated with the advantage $A_t = G_t - V_\theta(s_t)$, where $V_\phi(s_t)$ is the value function and ϕ are the parameters of the critic. Although the variance on G_t is high, the variance of A_t will be lower due to the value function centring the changes in G_t around zero. Hence the update functions for the actor and critic are:

$$\theta \to \theta + \alpha \nabla_\theta A_t \, ln [\pi(a_t|s_t)] \,, \tag{2}$$

$$\phi \to \phi + \beta \nabla_\phi (G_t - V_\phi(s_t))^2. \tag{3}$$

1.3.3 Proximal Policy Optimisation

Proximal Policy Optimisation[11] (PPO) is a policy gradient method similar to A2C but with further improved training stability. This is because the changes made to the policy

are limited to a clipped to a 'trust' region, preventing any massive changes from negatively affecting the policy and forcing the policy to be updated in smaller steps. The parameter update equation for PPO is:

$$\theta \to \theta + \alpha \nabla_\theta \, min \left\{ A_t r_t(\theta), clip(r_t(\theta), 1+\epsilon, 1-\epsilon) \right\}, \tag{4}$$

where the probability ratio is $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$.

1.3.4 Entropy Regularisation

Now that the objectives of the policy gradient methods have been defined, how can the exploration and exploitation of the agents be balanced? Such a method is not required for algorithms like REINFORCE, because it takes samples directly from the probabilities of the policy. For agents trained with the A2C and PPO algorithms, it is much more likely for an agent to converge prematurely to a sub-optimal policy instead of exploring other policies. The solution to this is to encourage a higher entropy on the predictions made by the policy, as this will make exploration occur more frequently. The formula for the entropy of a policy is:

$$E(\pi(s)) = \sum_a \pi(a|s) \log \pi(a|s). \tag{5}$$

The balance between exploration and exploitation is created by using a technique called entropy regularisation, which adds a term to the objective function $J(\theta) = \mathbb{E}_\pi \left[G_t\right] + \lambda E(\pi)$. The entropy, $E(\pi)$ is weighted by λ. This weighting determines how much entropy should be encouraged, thus controlling the exploration-exploitation balance.

1.4 Parameterised Quantum Circuits

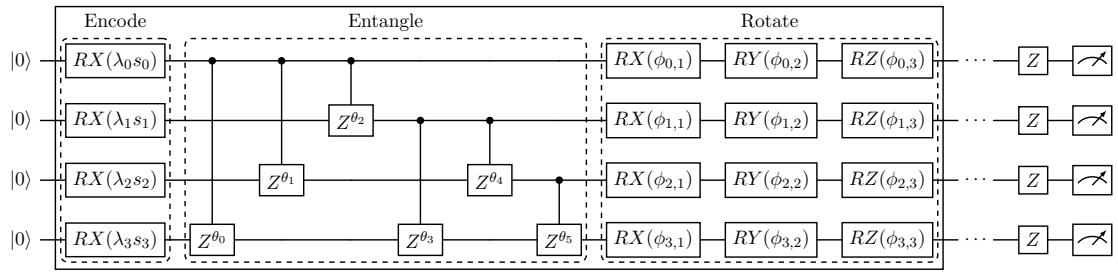


Figure 2: The architecture of a single layer inside the re-uploading PQC used in this project. λ, θ, ϕ are the parameters for encoding, entangling and rotating respectively, and s is the state of the agent.

Quantum circuits are the fundamental systems of quantum computing. Using a series of qubits and quantum gates, the qubits can perform calculations similar to a classical computer, with the advantages of superposition and entanglement.

Qubits are the quantum analogue of a classical bit and are the fundamental unit of information in quantum computing. A classical bit can be either zero or one, but a qubit can be in a superposition between the two states. Qubits can also become entangled, meaning the state of the qubits can depend on each other. This a uniquely quantum phenomenon that gives qubits their enhanced ability to process information in parallel.

Quantum gates are operations that act on qubits and are used to directly manipulate them. In the context of quantum machine learning, the most useful gates will be the rotation gates and entangling gates, as rotation gates are used to modify the qubits' state and entangling gates add non-linearity to the circuit.

A PQC is a quantum circuit which has gates that depend on a learnable parameter. It is due to this that PQCs can be used to represent a policy in the same manner as a neural network. In a contemporary PQC, the 'input' of the circuit is encoded into the qubits using quantum gates at only the start of the circuit, before any parameterised gates are applied. However, this single-shot encoding can come with many issues, such as the inability to capture non-linear relationships between inputs, and vanishing gradients for deep circuits.

The solution to this is to use a re-uploading PQC, where the input is repeatedly encoded into the circuit. This allows the circuit to capture more intricate patterns and fundamentally improves the expressive power of the circuit.[12]

For this project, the structure of the re-uploading PQC is visualised in Figure 2. Each layer of the PQC consists of an encoding stage, a variational entanglement stage, and a rotation stage.

1.4.1 Encoding

In a PQC, encoding is the process of mapping classical input data into quantum states through the use of rotation gates. For the PQC in this project, the state of the environment is encoded by weighting the state and then using RX gates to rotate the qubits by the weighted inputs. This parameterises the state and then introduces that information into the quantum circuit. This approach enhances the circuit's ability to express complex relationships from the input data, as the re-uploading of the parameterised inputs allows the circuit to extract the broadest spectrum of features available. However, as the circuit deepens, the gradient variance becomes lower, and there is a risk of gradient vanishing.[13]

1.4.2 Entangling

Entanglement is a quantum phenomenon where two quantum states become dependent on each other. In a quantum circuit, this is usually achieved with an entangling gate such as a CNOT or a controlled-Z gate and can be used to create non-linear relationships between the qubits. For this PQC, the entanglement gate used is a parameterised controlled-Z gate, which entangles two qubits but also takes a learnable parameter that controls how much entanglement is applied. This allows for more flexible control over the strength of the entanglement between qubits. Previous work has shown that parameterised entangling gates outperform fixed gates.[14]

The method of entangling is incredibly significant for a quantum circuit. Since all the non-linearity in a PQC arises from entanglement, the circuit's ability to represent complex data depends on the method used. In Figure 2, the diagram of the layer depicts an all-to-all entanglement. All-to-all entanglement is where every qubit is entangled with every other qubit instead of other configurations. Recent studies have found that all-to-all entanglement improves the expressive power of the circuit.[15]

1.4.3 Rotating

In a quantum circuit, rotation gates are gates that modify the state by rotating the qubit on the Bloch sphere. The Bloch sphere is a geometrical representation of qubit state space in three dimensions. Parameterising the rotations provides a method to directly modify the state of the qubits. For this project, each layer of the PQC has three rotations per qubit, one on each of the x, y and z axes. Three rotations are used to make sure that the circuit has high flexibility, as three rotations are a requirement to represent all the possible transformations on the Bloch sphere.

1.4.4 Training Parameters

Traditionally, parameters in a classical neural network are trained by calculating the gradient of the loss function and using an optimisation strategy to iteratively update each parameter. This becomes a glaring issue for PQCs, as qubits are fundamentally different from classical bits. This makes it impossible to backpropagate a quantum circuit in the same way as a neural network. To overcome this, quantum circuits are forced to rely on techniques to estimate the gradient, such as the parameter shift rule.[16] The parameter shift rule is especially useful since it avoids noisy finite difference approximations. The formula for the parameter shift rule is:

$$\nabla_\theta f(\theta) = \frac{1}{2}\left[f\left(\theta + \frac{\pi}{2}\right) - f\left(\theta - \frac{\pi}{2}\right)\right].$$

The only drawback of using this technique to calculate the gradient of a quantum circuit is that it is a numerical technique that requires multiple evaluations of the circuit, making it quite a computationally expensive method.

2 Methodology

This section will describe the implementations of the three policy gradient methods used to solve the cartpole problem. These methods are applied to both a PQC and a classical neural network. The following will break down the data that was gathered, the architectures, and the hyperparameters of the agents.

The agents in this project were created by applying the RL algorithms to TensorFlow models, with the quantum agents using Tensorflow Quantum to replace the body of the algorithm with custom-made PQCs so that the circuit can be simulated. This experimental setup requires very specific versions of OpenAI Gym, Cirq, TensorFlow, and TensorFlow Quantum. This is made reproducible using Docker, as the code contains a docker file that can produce an environment necessary for the code to run on any platform.

2.1 Performance Metrics

There are many metrics to evaluate the training of the agents. The metrics we use are the total reward gathered in an episode, the episode length for each training step, the average reward during an episode, and the cumulative reward throughout training. Each metric expresses something different about the way the agent is learning. The total reward directly reflects how the agent is performing over time. However, the cumulative reward shows how the agent is learning over time, represented by the gradient of its curve.

The episode length expresses how often the agent meets the success criterion for the environment and the process of how it learns to do so. The average reward conveys the performance of the agent regardless of the episode length.

There are two metrics available to us to compare the differences between policies — the reward distribution, and success rate of the agents. The reward distribution provides us with insights as to how well a policy is performing. Conversely, the success rate will tell us how often the agent solves the task. The gathered rewards can be visualised, and since an episode length reaching the limit corresponds to a successfully solved task, the episode lengths can be used to calculate the success rate.

During training, the total reward and episode length will be gathered for each training step. This will also allow the cumulative reward and average reward to be computed.

Once the agents are trained, it will be time to gather performance data on the agents' policies. 100 episodes of the environment will be sampled using the policy for each agent. The reward at each step and the length of every episode will be gathered, and this will then be used to visualise the reward distribution for that policy. The episode length will also be used to compute the success rate for the policy of each agent.

2.2 Agent Architecture

In this section, the architectures of the classical and hybrid quantum agents will be described. There is more than one architecture because REINFORCE is a pure policy gradient method, whereas A2C and PPO are actor-critic methods and require a different architecture. In REINFORCE, the agent is just a function that maps states directly to actions. However, A2C and PPO require a modified architecture that outputs the policy (actor) and discounted expected return (critic). The architecture for the REINFORCE agent can be seen in Figure 3a, and consists of an input layer, a hidden or a PQC layer, and then a dense output layer with a softmax activation function to normalise the output probabilities. The architecture for the actor-critic agents is shown in Figure 3b, it consists of an input layer, a hidden or a PQC layer, then two dense layers as outputs, the policy and value, which predict the action probabilities and expected return of the action respectively.

After analysing Figure 3, it can be observed that inside the body of both architectures is a 'Hidden/PQC layer'. For classical RL this layer is a hidden layer, and the agent will be comprised of a neural network. However, for QRL, that layer is replaced with a re-uploading PQC, applying the output from the last layer as the inputs to be re-uploaded. The measurements of the circuit are then taken as the outputs of the layer. The outputs of the PQC are then passed to a dense layer in REINFORCE, and two dense layers in A2C and PPO. This is done to transform the output of the PQC and makes the agent an HQC. Since the structure of the re-uploading PQC has a qubit for each component of the state, the number of outputs for that layer will be four. These outputs must be reinterpreted as a discrete policy and also an estimate for the discounted expected return in the actor-critic methods. The simplest way to do this is with a classical post-processing layer to interpret the measurements from the PQC. It is because of this that our quantum agents are considered to be hybrid and not pure PQCs.

For the REINFORCE algorithm, the agents consist of a parameterised policy that directly maps states to action probabilities. The classical agent is a neural network with four input units. The network consists of a hidden layer with twelve units and a hyperbolic tangent activation, then a dense output layer with two units and a softmax activation.

(a) The architecture of the REINFORCE agent.

(b) The architecture of the actor-critic style agents.

Figure 3: The architectures of the pure policy gradient and actor-critic methods.

In total, the neural network has 86 parameters. The quantum agent is composed of a re-uploading PQC with seven layers, as visualised in Figure 2. The four measurements of the PQC are then connected to a dense layer with two units and a softmax activation function. In total, this agent has 164 parameters. This is nearly double the number of parameters in the classical agent. Inside the PQC of the agent, 28 parameters are used for encoding the state, 42 are used for variational entanglement, 84 are for rotating the qubits, and the remaining 10 parameters are from the post-processing dense layer. This final layer converts the outputs of the PQC into the action probabilities.

For the A2C and PPO algorithms, the classical agent is a neural network comprised of four input units. This network consists of a hidden layer with 12 units and a hyperbolic tangent activation, a dense output layer with two units and softmax activation that represents the policy, and a dense layer with one unit and linear activation that represents the value. The classical agent has 89 parameters in total, 60 of them being shared parameters in the hidden layer, of which 26 are exclusively for the policy and 13 are for the value. The quantum agent has a PQC with seven layers. This PQC has four outputs that connect to the dense policy and dense value output layers, which have softmax and linear activation functions respectively. As in the classical model, the quantum model also requires the policy to be composed of two units and the value to have one. The quantum agent has 169 parameters, identical to the quantum agent trained with REINFORCE. However, an extra 5 parameters are used to transform the output of the PQC into the value, by using a dense layer with one unit and a linear activation.

The reason that the quantum agents are allocated more parameters is not to needlessly add complexity but to ensure that the expressive power of the PQCs is fully leveraged. This allows the agent to explore a richer set of possible solutions.

2.3 Hyperparameters

This section will cover the choice of hyperparameters for each agent and the justification behind those choices.

2.3.1 REINFORCE

There are only two hyperparameters in the REINFORCE algorithm, the learning rate and the discount factor. The learning rate controls how large the optimisation steps are, and the discount factor determines the importance of future rewards. If the learning rate is too high, the agent may never converge as it will make destructive changes to the policy. If the learning rate is too small, the agent may converge to a suboptimal policy. A discount factor closer to one implies that the agent should put future rewards before immediate rewards. A discount factor closer to zero implies that the agent should chase greedily after rewards. The algorithm for learning parameters with REINFORCE is,

1. Sample a trajectory using the current policy.
2. Use the discount factor to compute the expected return for each step of the trajectory.
3. Update the policy parameters using Equation 1.
4. Repeat 1-3 until the average length of the last 100 trajectories is greater than 475.

For the classical agent, the optimal learning rate and discount factor were found through trial and error to be 0.01 and 0.99 respectively, similarly in the quantum agent they were 0.009 and 0.99. The learning rate for the quantum agent was slightly lower since the agent would suffer from extreme reward fluctuations during training with a high learning rate. There are still large fluctuations in the reward for the quantum agent, but it successfully solves the cartpole environment.

2.3.2 A2C

The large amount of hyperparameters in A2C allows for a significant increase in control over the training of the agents. The hyperparameters were: the learning rate, the discount factor, the loss weights, the entropy coefficient, and the gradient normalisation constant. The loss weights govern how the actor and critic loss are weighted before calculating the gradients, as changing this can allow for balanced learning in the scenario where one loss dominates another in the training process. The entropy coefficient is used after calculating the entropy of the current prediction with Equation 5 to be used as a measure of unpredictability, the entropy is then weighted by the entropy coefficient and applied to the loss function during training to encourage exploration. Gradient normalisation is a type of gradient clipping, which prevents the gradients from becoming too large in one update step by normalising them to a certain value as this can help improve the stability of an agent. The algorithm for learning the parameters with A2C is,

1. Sample a trajectory using the current policy.
2. Use the discount factor to compute the expected return for each step of the trajectory.
3. Use the expected return to calculate the advantage.
4. Update the actor using Equation 2.
5. Update the critic using Equation 3.
6. Repeat 1-5 until the average length of the last 100 trajectories is greater than 475.

For the classical model, the optimal learning rate was found to be 0.01, the discount factor was 0.99, the entropy coefficient was 0.01, the gradient normalisation constant was 0.75 and the loss weights were 1.0 and 0.75 for the actor and critic respectively.

The quantum agent had identical hyperparameters, but the learning rate was reduced to 0.006 and the entropy coefficient was set to zero to promote stability during training.

2.3.3 PPO

PPO is an improvement on A2C which works by updating the agent in smaller steps. It has a few more hyperparameters than A2C which aims to improve the stability of the training process. These new hyperparameters are the clipping parameter and the number of epochs. The clipping parameter controls the size of the trust region described in Section 1.3.3. The number of epochs is the amount of times the training step will be repeated so that the agent will effectively learn from the available trajectory. A clipping parameter that is too small can prevent the policy from converging on an optimum, conversely, a large clipping parameter can allow destructive changes to be applied to the policy. If the number of epochs is too small, the agent may not fully learn from trajectory, if the number of epochs is too high the agent may overfit to specific trajectories. The algorithm for learning parameters with PPO is

1. Sample a trajectory using the current policy, while storing the action probabilities to be used in the training step.
2. Use the discount factor to compute the expected return for each step of the trajectory.
3. Use the expected return to calculate the advantage.
4. Update the actor using Equation 4.
5. Update the critic using Equation 3.
6. Repeat 3-5 for the number of epochs.
7. Repeat 1-6 until the average length of the last 100 trajectories is greater than 475.

The classical agent was trained with a learning rate of 0.01, a discount factor of 0.99, an entropy coefficient of 0.001, a gradient normalisation constant of 0.5, a clipping parameter of 0.2, the number of epochs as 5, and equal loss weights of 1.0 for both the actor and the critic.

The quantum agent shared the same discount factor, gradient normalisation constant, and clipping parameter as the classical model. However, the learning rate was halved to 0.005 and the entropy coefficient was set to zero for stability. In addition, the number of epochs was set to 3, as repeated measurements with the quantum simulation slowed down the training process. The loss weights were set to 1.0 and 0.9 for the actor and critic respectively.

2.4 Implementation Challenges

The implementation of the agents contained an endless supply of challenges for this project. Specifically, the hyperparameters of all the quantum agents are *extremely* sensitive, with even a small deviation in learning rate or discount factor causing the models never to

reach the success criterion for the cartpole task. This problem was also exacerbated by the time it took to train the quantum models. The classical models all took around five minutes to train but the added complexity of simulating the quantum circuit, and calculating the gradients, made each quantum agent take around 20 minutes to train. Originally this problem was much worse, but the implementation of TensorFlow graphs dramatically reduced the training time for the quantum models. Even still, the process of tuning hyperparameters for the quantum models is made much more difficult by the long training time.

The combination of the long training time and sensitive hyperparameters made the quantum models especially time-consuming. Fortunately, the classical counterpart for each algorithm usually exhibited more stable training, less sensitivity to hyperparameters, and a significantly lower training time.

Another factor of note was the normalisation of the state, as the quantum agents were almost completely non-functional without this, likely due to information being lost in encoding when the state was interpreted as an angle for the rotation gate. This issue did not affect the classical agents, but adding normalisation may have slightly improved their training stability. The expected returns also had zero-mean normalisation applied during training to enhance the agents' performance, since this prevents excessively large updates to the policy and significantly helps improve the stability.

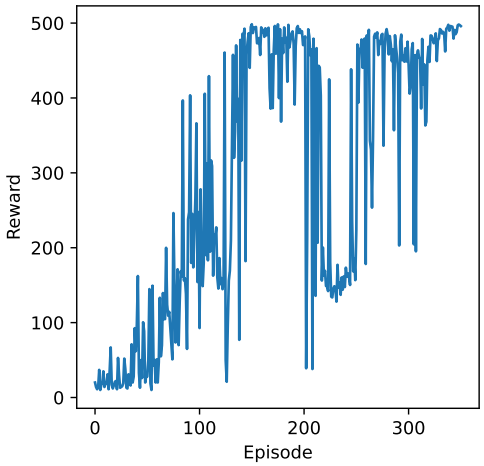
3 Results

Figures 4, 5, and 6 are graphs of the total reward generated by the agents during training, grouped by the algorithm used to train them. These graphs describe how the agent was performing throughout its training. An upward trend implies that the agent is learning, and a downward trend implies that the agent is then forgetting. In addition, lots of noise implies unstable training. All of the agents are seen to follow an upward trend, however, the quantum agents are generally noisier. This shows that all the agents are successfully solving the environment, but the training of the quantum agents are more unstable.

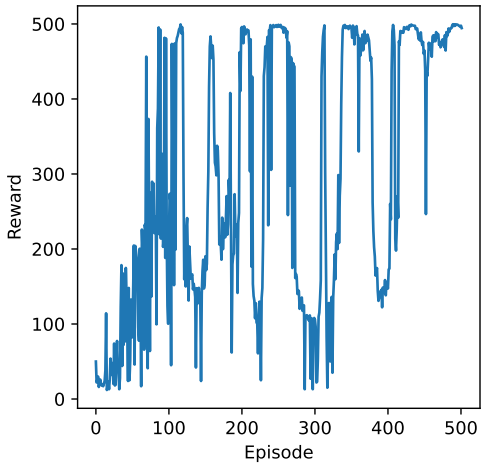
While Figures 4, 5, and 6 show the total reward generated by each agent during training, Figures 7, 8, and 9 show the average reward. The average reward negatively correlates with the average distance from the centre of the track, hence, a higher average reward indicates a more controlled agent. All graphs appear to start at one (the maximum reward). This is because the agent loses balance before travelling far from the centre of the track, which inflates the average reward. Throughout the training, the average reward appears to oscillate for all the agents. However, the graphs all start from one and then approach one when the training is finished.

Figure 10 shows the cumulative reward of each agent throughout their training process, grouped by the algorithm that trained them. The cumulative reward during training is a direct reflection of the agent's learning, since a steep gradient implies that the agent is performing well, and a horizontal gradient implies poor performance. Therefore, the rate at which the gradient changes reflects how the agent is learning.

Figure 11 depicts the distribution of each agent's rewards for 100 episodes in the cart-pole environment. This is grouped again by the algorithm they were previously trained on. Note that the graph y-axes have different scales to better visualise the tails of the distribution. The intent behind this graph is to visualise the performance of the policies that each agent has learned. A policy with a high median and low variance reward is

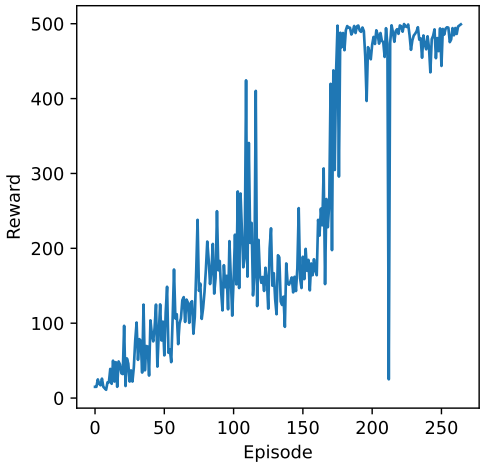


(a) The reward during training for the classical neural network.

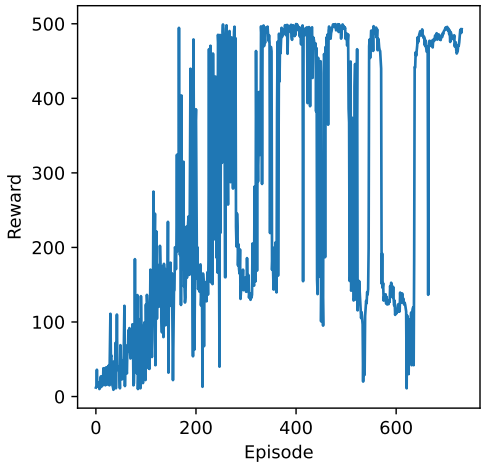


(b) The reward during training for the hybrid PQC.

Figure 4: The total reward for each episode of training the agents with REINFORCE.



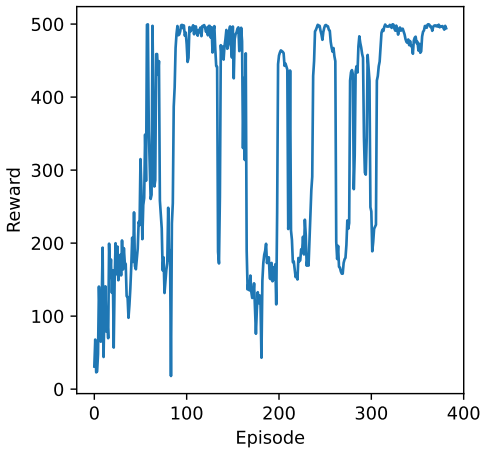
(a) The reward during training for the classical neural network.



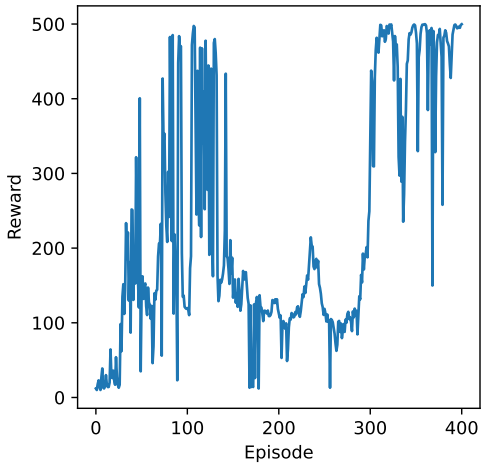
(b) The reward during training for the hybrid PQC.

Figure 5: The total reward for each episode of training the agents with A2C.

performing consistently better than a policy with a low median and high variance reward. While it may be difficult to see, the graphs show that the median reward for both classical and quantum agents increases with the complexity of the training algorithm, which is as you would expect. The quantum agent trained using PPO performed the best with a median reward of 0.999 and an interquartile range of 0.002. The quantum agent trained using REINFORCE performed the worst with a median reward of 0.990 and an interquartile range of 0.028.

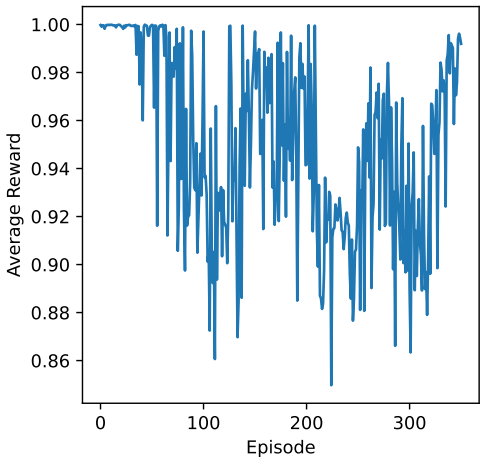


(a) The reward during training for the classical neural network.

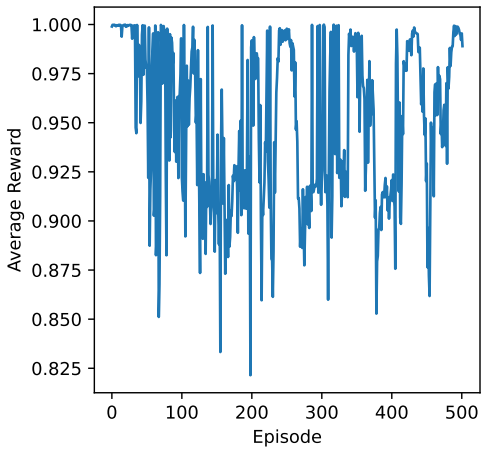


(b) The reward during training for the hybrid PQC.

Figure 6: The total reward for each episode of training the agents with PPO.



(a) The average reward during training for the classical neural network.



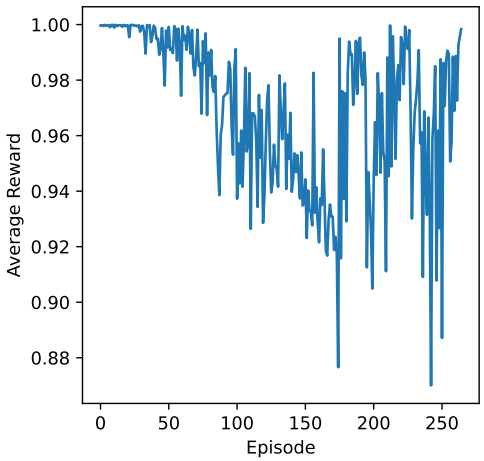
(b) The average reward during training for the hybrid PQC.

Figure 7: The average reward for each episode of training the agents with REINFORCE.

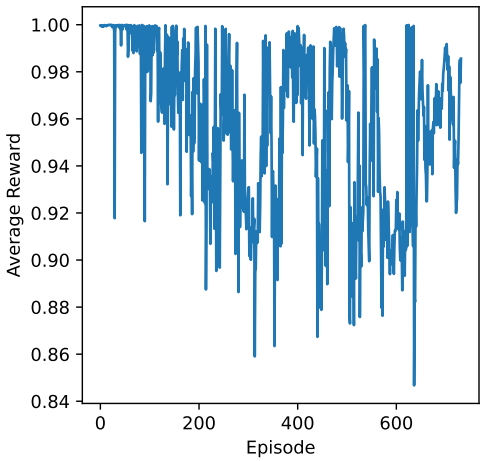
4 Discussion

The results show that the quantum agent trained using PPO performed the best with an almost perfect policy, whereas the quantum agent trained with REINFORCE performed the worst, with a slightly lower median, and a much larger interquartile range. The results also showed that the training curves of the quantum agents exhibited much more noise than their classical counterparts.

Examining the average reward in Figures 7, 8 and 9, the average reward of every agent seems to oscillate wildly. This is almost certainly because the agent is fluctuating between

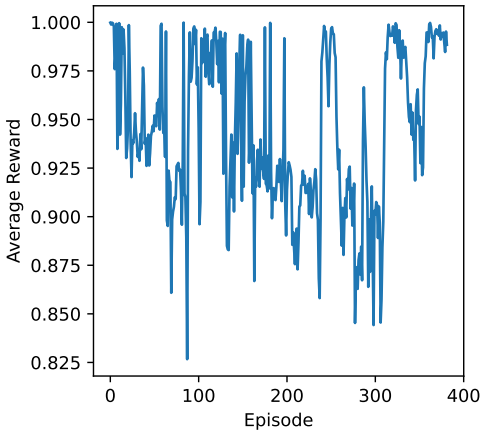


(a) The average reward during training for the classical neural network.

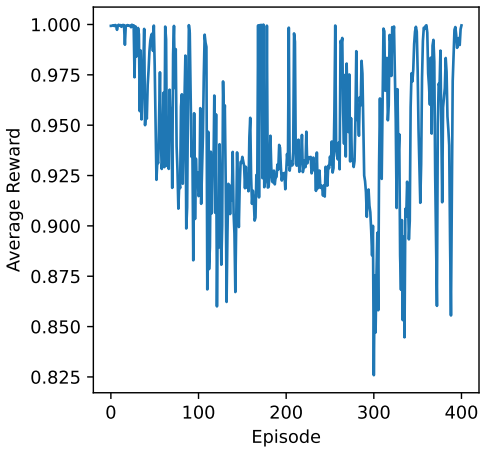


(b) The average reward during training for the hybrid PQC.

Figure 8: The average reward for each episode of training the agents with A2C.



(a) The average reward during training for the classical neural network.

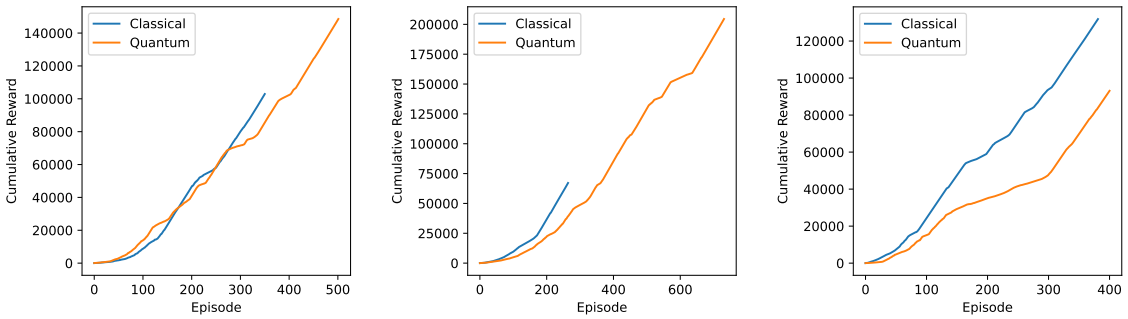


(b) The average reward during training for the hybrid PQC.

Figure 9: The average reward for each episode of training the agents with PPO.

prioritising the episode length and the distance from the centre of the track. This occurs because the agent is maximising the expected reward, and the expected reward depends on only the episode length and the position from the centre of the track.

Examining the agents trained with REINFORCE, the reward during training can be seen in Figure 4. The rewards during training for the classical agent can be seen in Figure 4a. This shows the agent exhibiting a clear upward trend before it converges to the maximum total reward of 500. The rewards during training for the quantum agent can be seen in Figure 4b. This graph shows that the reward fluctuates heavily throughout training, fortunately, there is still an overall upward trend. The fluctuations are likely due

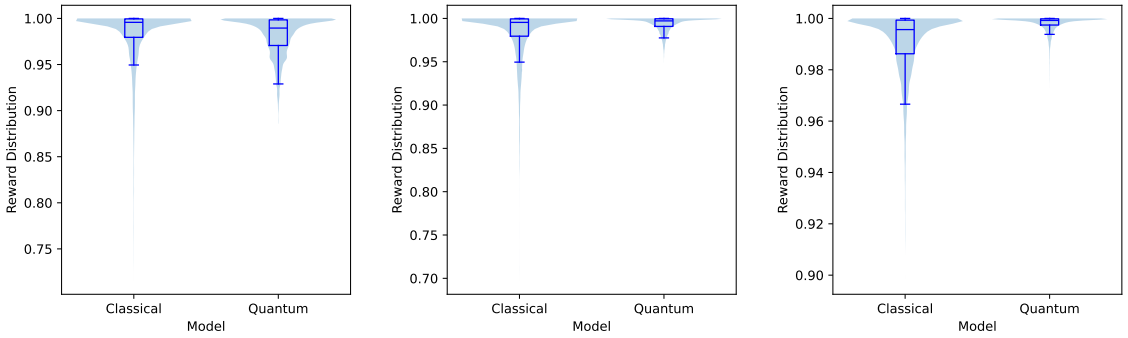


(a) The cumulative reward for the agents trained with REINFORCE.

(b) The cumulative reward for the agents trained with A2C.

(c) The cumulative reward for the agents trained with PPO.

Figure 10: The reward distribution and density for each agent.



(a) The reward distribution and density for the agents trained with REINFORCE.

(b) The reward distribution and density for the agents trained with A2C.

(c) The reward distribution and density for the agents trained with PPO.

Figure 11: The reward distribution and density for each agent.

to the formation of barren plateaus when updating the quantum circuit,[17] which leads to exponentially smaller gradients as training progresses. This is the case for the gradient estimates in all the PQCs and is the largest contributor to their unstable learning.

For the A2C agents, the classical agent exhibited remarkably stable training, which can be seen in Figure 5a. In this figure, the variance in the total reward of an episode is visibly lower than that of all the other agents, which would suggest an optimal policy was reached. The quantum agent also had a good learning curve in Figure 5b, but the rewards become significantly noisier as the upward trend increases.

For the PPO agents, the rewards during training of the classical agent can be seen in Figure 6a. This shows the reward during training exhibiting an extremely rapid increase in rewards and then a period of high variance in reward. The noisy rewards could suggest that the agent was suffering from incorrectly tuned hyperparameters. The reward during training for the quantum model is visualised in Figure 6b. This graph shows a rather abnormal learning curve since the agent seems to forget and begin performing poorly at episode ∼150 and then experiences an extremely rapid increase in reward at episode ∼300. This may be due to catastrophic inference in action, as the rewards seem to suddenly drop from the maximum before abruptly recovering again. Since the agent was improving, the

trajectories that the agent was being trained on were changing as well, leading to unstable changes to the agent's policy, and ultimately catastrophic inference.

Cumulative reward is an important metric in RL as the total reward that an agent has accumulated directly reflects the learning progress of the agent. A steep gradient implies that the agent was learning rapidly and a more horizontal gradient would suggest that the learning has slowed down or even reversed. The cumulative reward during training for all the agents is shown in Figure 10.

The cumulative reward for the agents trained with REINFORCE in Figure 10a shows that the classical agent had a mostly smooth training curve and solved the cartpole problem after just 350 episodes. Comparatively, the quantum agent had a much choppier training curve and took 500 episodes to train. The choppy dents in the curve reflect the agents performing poorly during training, as they correspond to the drops in performance in Figure 4.

The agents trained with A2C in Figure 10b show dramatic differences compared to all the other agents. The classical agent solved cartpole in 260 episodes, whereas the quantum agent took 730 episodes. The graph clearly shows that the classical agent learned at a faster rate with a much smoother curve. In comparison, the quantum agent had a very rough curve, and took a much longer amount of time to train.

For the agents trained with PPO, Figure 10c shows that the classical agent is not as smooth as the other classical agents. However, the gradient of this graph for the classical agent is almost constant, indicating that the classical agent learned at a fixed rate throughout training. The cumulative reward of the quantum agent seems to indicate that the reward saw a large drop due to the large section of episodes with a horizontal gradient. This drop in rewards can also be seen in Figure 6b.

Although analysing the performance of the agents during their training expresses a great deal about the resulting agent, it is still crucial to assess the policy learned by the agent to see how it behaves in the environment. Originally, the evaluation was going to include a bar chart of the success rate of the policy. This is the percentage of episodes that the agent solves cartpole. However, after taking 100 sample episodes the success rate for all the models was 100%, this was not always the case for some of the classical agents. After minor adjustments to the hyperparameters, these issues were eradicated. Another method for determining the performance of a policy was to sample the rewards and visualise them as a distribution. The reward distribution from the 100 episodes of testing and can be seen in Figure 11.

In Figure 11a, the distribution of rewards for agents trained with REINFORCE can be seen, with both agents having a similar range of rewards albeit with the quantum agent having a slightly lower median and larger spread. This would suggest that for the policies trained with REINFORCE, the classical agent performs marginally better than the quantum agent.

This pattern does not continue for the agents trained with A2C as in Figure 11b the quantum agent considerably outperforms the classical agent, with a much lower variance in reward and values much closer to one. It should be noted that although the quantum agent saw a massive improvement from REINFORCE to A2C, the distribution of the classical agent remained mostly unchanged.

The difference in performance for the agents trained with PPO in Figure 11c is even more exaggerated than with A2C, although both agents show improved performance with a much lower spread than any other algorithm, the quantum agent has nearly reached an

objectively perfect policy with a median of 0.999 and an interquartile range of 0.002.

After analysing the training of the agents using their cumulative reward and analysing the policy of the agents using large samples of the agent's performance in the environment, we are left with a confusing scenario where the quantum models seem to perform better overall but experience volatile training. The volatile training is most likely due to a common problem with PQCs, barren plateaus,[17] the current solution to this issue is to perform better parameter initialisations and could be an avenue for future work.

In comparison, the classical agents seem to have smoother training but a higher variance in rewards for the finished model. The performance of the agents increasing with the algorithm complexity was expected, as transitioning from REINFORCE to A2C to PPO adds more methods to stabilise training. This enables agents trained with more stable algorithms to converge to more optimal policies.

This project has successfully bridged a gap between theory and practice, as it has confirmed that PQCs outperform classical neural networks and hold an advantage in policy performance.

Given more time, continuous versions of the algorithms would have been implemented to train agents in environments with a continuous action space, such as the pendulum problem. This would have provided another perspective to compare quantum circuits to classical neural networks. However, these continuous agents were never brought into existence due to the time required to train and fine-tune the quantum agents. An environment like the pendulum problem would have served as a great comparison, not because it is a benchmark but because the combination of its complex reward function and action space would give the quantum and classical agents an opportunity to be compared using a more complex classical control environment. If such time were available, continuous agents would not be the only addition to this project, the parameter initialisation for the PQCs would have been geared towards improving the training stability of the quantum circuits as well.

5 Conclusion

The evaluation of the results confirms that the agents perform the best with PPO and the worst with REINFORCE. This is as you would expect, considering REINFORCE is a vanilla policy gradient method, A2C introduces a critic to reduce the variance of gradient estimates, and PPO improves upon A2C by preventing large and destructive changes to the policy.

Comparing the quantum and classical agents, for REINFORCE the distribution of rewards was similar, showing that the difference in policy was not massive. The classical agent showed a slight advantage in performance, however, for the actor-critic methods, the reduced variance of gradient estimates caused the quantum agents to experience a massive improvement in the performance of the policy.

This result appears to be contrary to how the agents learned during training, as the classical agents converged much faster in every case and their learning curves visibly had less noise. However, these phenomena are both two sides of the same coin. The quantum agents perform better because the PQCs can use superposition and entanglement to express more complicated relationships. However, the quantum agents' training is more noisy because gradient vanishing is more common the deeper a circuit is.

The best algorithm for both classical and quantum agents is undoubtedly PPO, as

forcing the policy to update in small steps prevents large and destructive policy changes. This is reflected in the results of this project since the two best-performing agents were both trained with the PPO algorithm.

There were many limitations to this study, the largest being the computational cost of the quantum simulations. Furthermore, there is no guarantee that the simulated quantum circuit can fully capture the expressive potential of a PQC. If this study had implemented agents for a problem with a continuous action space, it would have expanded on the comparison between RL and QRL, as a continuous action space would require a much more complicated policy. This could have helped quantify the expressive power of both classical and quantum agents.

To build on this study, future work could apply all the agents to a more complicated environment or change the initialisation and architecture of the quantum agents to test what configurations have the lowest instability during training.

In summary, the performance of the agents directly correlated with the complexity of the algorithms, with the quantum agents vastly outperforming the classical agents for actor-critic methods. The difficulty with the quantum circuits is their sensitivity to noise which leads to very unstable training. Overall, the results imply that quantum reinforcement learning agents have enhanced potential and could be used in a variety of applications to significantly enhance autonomous agents, such as in self-driving cars and game AI.

References

[1] Andrew G. Barto, Richard S. Sutton, and Charles W. Anderson. "Neuronlike adaptive elements that can solve difficult learning control problems". In: *IEEE Transactions on Systems, Man, and Cybernetics* SMC-13.5 (1983), pp. 834–846. DOI: 10.1109/TSMC.1983.6313077.

[2] Ruben Rodriguez Torrado et al. "Deep Reinforcement Learning for General Video Game AI". In: *2018 IEEE Conference on Computational Intelligence and Games (CIG)*. 2018, pp. 1–8. DOI: 10.1109/CIG.2018.8490422.

[3] Jens Kober, J. Andrew Bagnell, and Jan Peters. "Reinforcement learning in robotics: A survey". In: *The International Journal of Robotics Research* 32.11 (2013), pp. 1238–1274. DOI: 10.1177/0278364913495721.

[4] Jacob Biamonte et al. "Quantum machine learning". In: *Nature* 549.7671 (2017), pp. 195–202. DOI: 10.1038/nature23474.

[5] Zhan Yu et al. *Provable Advantage of Parameterized Quantum Circuit in Function Approximation*. 2023. URL: https://arxiv.org/abs/2310.07528.

[6] Martin Riedmiller, Jan Peters, and Stefan Schaal. "Evaluation of Policy Gradient Methods and Variants on the Cart-Pole Benchmark". In: *2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*. 2007, pp. 254–261. DOI: 10.1109/ADPRL.2007.368196.

[7] André Sequeira, Luis Paulo Santos, and Luis Soares Barbosa. "Policy gradients using variational quantum circuits". In: *Quantum Machine Intelligence* 5.1 (2023), p. 18. DOI: 10.1007/s42484-023-00101-8. URL: https://doi.org/10.1007/s42484-023-00101-8.

[8] H. Riel. "Quantum Computing Technology". In: *2021 IEEE International Electron Devices Meeting (IEDM)*. 2021, pp. 1.3.1–1.3.7. DOI: 10.1109/IEDM19574.2021.9720538.

[9] Ronald J. Williams. "Simple statistical gradient-following algorithms for connectionist reinforcement learning". In: *Machine Learning* 8.3 (1992), pp. 229–256. DOI: 10.1007/BF00992696.

[10] Volodymyr Mnih et al. *Asynchronous Methods for Deep Reinforcement Learning*. 2016. URL: https://arxiv.org/abs/1602.01783.

[11] John Schulman et al. *Proximal Policy Optimization Algorithms*. 2017. URL: https://arxiv.org/abs/1707.06347.

[12] Maria Schuld, Ryan Sweke, and Johannes Jakob Meyer. "Effect of data encoding on the expressive power of variational quantum-machine-learning models". In: *Physical Review A* 103.3 (Mar. 2021). DOI: 10.1103/physreva.103.032430.

[13] Andrea Skolik et al. "Layerwise learning for quantum neural networks". In: *Quantum Machine Intelligence* 3.1 (2021), p. 5. DOI: 10.1007/s42484-020-00036-4.

[14] Stig Elkjær Rasmussen and Nikolaj Thomas Zinner. "Parameterized Two-Qubit Gates for Enhanced Variational Quantum Eigensolver". In: *Annalen der Physik* 534.12 (Oct. 2022). DOI: 10.1002/andp.202200338.

[15] Sukin Sim, Peter D. Johnson, and Alán Aspuru-Guzik. "Expressibility and Entangling Capability of Parameterized Quantum Circuits for Hybrid Quantum-Classical Algorithms". In: *Advanced Quantum Technologies* 2.12 (2019), p. 1900070. DOI: https://doi.org/10.1002/qute.201900070.

[16] K. Mitarai et al. "Quantum circuit learning". In: *Phys. Rev. A* 98 (3 Sept. 2018), p. 032309. DOI: 10.1103/PhysRevA.98.032309.

[17] Muhammad Kashif et al. "Alleviating Barren Plateaus in Parameterized Quantum Machine Learning Circuits: Investigating Advanced Parameter Initialization Strategies". In: *2024 Design, Automation & Test in Europe Conference & Exhibition*. 2024, pp. 1–6. DOI: 10.23919/DATE58400.2024.10546644.