Reinforcement learning with partitioned quantum circuits

Darryn Biervliet

Supervisors:
V. Dunjko & S.C. Marshall

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl 13/08/2023

Abstract

The potential impact of machine learning, as it stands today, combined with its projected future implications, positions it as a prime candidate to harness the potential benefits offered by quantum mechanical exploits. However, the realization of these benefits is significantly hindered due to the limited size and noise of current quantum computers. To address these constraints, techniques are being developed to run large quantum algorithms on virtual qubits. Recent advancements in these methods have, under specific conditions, reduced their exponential overhead to an arbitrary computational budget, making them applicable for computationally intensive tasks. In this work, we apply such a scheme to a policy gradient-based reinforcement learning agent. This partitioning method enables us to reduce the quantum model's required qubits by half. We demonstrate that this agent can solve a classic control learning task efficiently with only a moderate amount of computational overhead. Furthermore, we show that increasing this budget enhances learning performance to a point where it is comparable to that of an agent using an unpartitioned quantum model. These findings may inspire further exploration of quantum-enhanced reinforcement learning agents in environments where model requirements surpass the available qubits on current quantum machines.

Contents

1 Introduction

In 2021, DeepMind published the paper "Reward is Enough" which hypothesizes that most, if not all, known attributes of intelligence—such as knowledge, learning, social intelligence, and generalization are emergent abilities of reward-maximizing behavior[24]. The machine learning framework that formalizes this goal-seeking optimization problem is called reinforcement learning (RL) and the organization behind the claim has substantial evidence to support it: In 2016, they used AlphaZero to beat the world champion in Go; in 2020, they introduced MuZero, which is capable of learning multiple challenging games from scratch; and in 2023, they utilized AlphaDev to make algorithmic breakthroughs in sorting algorithms [23, 14]. These accomplishments, among others, point towards the idea that reinforcement learning might play a pivotal role on the path to general-purpose artificial intelligence.

The potential impact that reinforcement learning, falling under the broader umbrella of machine learning, could have on society makes it an appealing candidate to harness the advantages of quantum computers. Quantum computers exploit physics to perform computations, thereby achieving exponential speedups over classical machines in a currently limited set of domains. To fully harness the capabilities of quantum computers, we are encouraged to explore applications where the potential benefits would be most profound. A common consensus is that machine learning is one of these domains.

For quantum computers, however, finding any practical advantages in machine learning proves to be a challenging task: Some initially perceived quantum advantages turn out to be replicable on classical machines, while other more definitive quantum advantages become negligible due to the necessity of first loading classical data onto the device, effectively nullifying any potential speedups[1]. These limitations, among others, have caused a growing debate about whether the search for genuine quantum supremacy within machine learning is the most fruitful approach, or if we instead should focus more on hybrid approaches that integrate quantum models into an end-to-end learning pipeline alongside classical algorithms [22]. Regardless of the ultimate goal, be it enhancement or supremacy, any decisive research is constrained by a more fundamental limitation of current quantum machines: the number and accuracy of qubits. This bottleneck hampers our ability to evaluate and benchmark quantum enhanced models at the scale classical algorithm operate, stalling progress toward any practical quantum machine learning applications.

Fortunately, there is a growing body of research that goes towards adding virtual qubits to quantum computers, which enables us to run higher dimensional algorithms on quantum machines smaller than the algorithm would naively require. Under strict conditions where an exact original output is demanded, these solutions are bound by the "no free lunch theorem"—that is, reducing the number of qubits results in exponential computational overhead, making them impractical for most applications. Yet, if we relax this exactness requirement and aim for a reasonable approximation, the computational overhead can be adjusted to fit within an arbitrary computational budget. In this work we will investigate the computational budget required to achieve satisfactory approximations for reinforcement learning tasks. Succesfull training might open up the possibilities of scaling up quantum reinforcement learning experiments towards more complex environments, which might allow future works to reach more conclusive statements about any advantages (or disadvantages) of using QRL algorithms.

2 Quantum Computing

Ever since we gained the ability to precisely manipulate single quantum systems there has been a keen interest in how to leverage the unique effects that occur for our computational benefits. Quantum computing is the field that leverages these phenomena, governed by the laws of quantum mechanics, to enhance information processing tasks. In the following section, we will provide a brief overview of some fundamental topics in quantum computing

2.1 Quantum mechanics

Quantum mechanics is a mathematical framework designed to describe and form the basis of physical theories at atomic and subatomic levels. It is deeply rooted in the principles of linear algebra. In the subsequent sections, we will provide a conscise overview of the concepts of quantum mechanics that are essential for a deeper understanding of quantum computing. The insights presented are largely derived from the works of [2, 8, 16].

2.1.1 Quantum state

Quantum mechanics operates within the Hilbert space. This space is a complex vector space equipped with an inner product. Every quantum mechanical system can be characterized by a state vector, which is a unit vector within the Hilbert space, often referred to as the state space. Assuming we have a two-dimensional state space complemented by a set of orthonormal basis vectors $|0\rangle$ and $|1\rangle$, any state vector within our space can be expressed as a linear combination of these basis vectors:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \tag{1}$$

Here, both α and β are complex numbers. For $|\psi\rangle$ to qualify as a unit vector, it must adhere to the normalization condition $\langle\psi|\psi\rangle = 1$. The quantities α and β, termed as amplitudes, are complex numbers that must satisfy the equation $|\alpha|^2 + |\beta|^2 = 1$. This concept can be extrapolated to an N-dimensional state space having N basis states. Nevertheless, in quantum computing, the primary focus is on the fundamental two-dimensional state, known as the qubit, and its composites. Analogous to a classical bit, a qubit can adopt a value of 0 or 1. However, unique to quantum mechanics, it can also assume a simultaneous combination of both states. This intriguing phenomenon is termed as superposition.

2.1.2 Composite systems

Combining multiple qubits and their respective state spaces into a larger state space is done using the tensor product. Suppose there are two states V and W, each with a set of basis vectors $|i\rangle$ and $|j\rangle$. The tensor product of V and W is a new state space $(V \otimes W)$ of which the basis is a linear combination of the elements in $|i\rangle \otimes |j\rangle$. The dimension of $(V \otimes W)$ is the product of the dimensions of V and W. Consequently, we can define the composite of a 2-qubit system as the tensor product of the state spaces of the individual systems.

$$|\psi\rangle = \alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle \tag{2}$$

$|00\rangle$ is a shorthand notation for $|0\rangle \otimes |0\rangle$. An n-qubit system can be described by a state vector in a 2^n-dimensional state space. This 2^n-dimensional state space requires 2^n basis states and amplitudes to describe the state vector, meaning that the state space grows exponentially, i.e., $\mathcal{O}(2^n)$.

2.1.3 Quantum evolution

In order to perform useful operations with qubits, they have to be able to change over time. Let's say we have a quantum state $|\psi\rangle$ at time t. We can describe the state at time $t+1$ by applying a unitary operator U to the state at time t.

$$|\psi\rangle_{t+1} = U|\psi\rangle_t \tag{3}$$

For $|\psi\rangle_{t+1}$ to adhere to the rules laid out in 2.1.1, we need U to be unitary. A unitary operator is a linear operator that preserves the norm of the vectors to which it is applied, i.e., it maps a vector of norm 1 to another vector of norm 1. For a matrix U to be unitary, its inverse U^{-1} must be equal to its conjugate transpose $U^\dagger$. An important implication is that any valid unitary applied to a quantum state has an inverse, meaning that any unitary operation on a quantum state is reversible.

2.1.4 Quantum measurements

As mentioned in 2.1.1, a qubit can be in a superposition of states until it is measured. The moment a quantum state is measured, it collapses into one of its basis states. The probability of observing a distinct basis state is determined by the square magnitude of the amplitude of that particular basis state, as prescribed by the Born rule. For each quantum system, we can define a set of measurement operators M_m, where m is the index of the possible measurement outcome. The probability of measuring the value M_m is given by:

$$p(m) = \langle\psi|M_m^\dagger M_m|\psi\rangle \tag{4}$$

M_m must satisfy the completeness equation: $\sum_m M_m^\dagger M_m = I$ for the probabilities to sum to 1, i.e., $1 = \sum_m p(m) = \sum_m \langle\psi|M_m^\dagger M_m|\psi\rangle$. By extending the constraints on measurement operators M_m and requiring them to be orthogonal projectors — that is, M_m is Hermitian and $M_m M_{m'} = \delta_{m,m'} M_m$ — we reduce it to a projector for a projective measurement P_m.

Projective measurements can be described by an observable, M, which is a Hermitian operator (i.e., a matrix with real eigenvalues) on the state space of the system being observed. We refer to these as projective measurements because we project a state space onto the eigenvalues of our measurement operator. An observable has a spectral decomposition of projectors onto the eigenspace of M, defined as:

$$M = \sum_m m P_m \tag{5}$$

Here, m is the eigenvalue of the projector P_m and a possible measurement outcome. The probability of getting result m is given by: $p(m) = \langle\psi|P_m|\psi\rangle$. The expectation value of the observable, often denoted as $\langle M\rangle$, is expressed as:

$$\langle M\rangle = \sum_m m p(m) = \sum_m m\langle\psi|P_m|\psi\rangle = \langle\psi|M|\psi\rangle \tag{6}$$

This value is a weighted average of the eigenvalues of the observable M, with the weights being the probability of observing the corresponding eigenvalue. An important property of projective measurements is that they are non-destructive. After a projective measurement, the state of the quantum system becomes:

$$|\psi\rangle \to \frac{P_m|\psi\rangle}{\sqrt{\langle\psi|P_m|\psi\rangle}} \tag{7}$$

Let's call this the measured state $|\psi_m\rangle$. When the state is remeasured, the probability of obtaining the same result is $\langle\psi_m|P_m|\psi_m\rangle = 1$. Note that, unlike unitary evolutions, measurements are irreversible. After a state is measured, the wave-function collapses onto a basis state, and any information in the amplitudes of other basis states is lost.

2.2 Quantum gates

Bits would be useless if their value was fixed. Classical computers allow us to flip the value of a bit with a `NOT` gate. Analogous to classical gates, quantum computers have quantum gates which allow manipulation of the state of a qubit. A Quantum gate is a unitary operator described as an $(2^n \times 2^n)$ dimensional unitary matrix, where n is the number of qubits on which it acts. The quantum equivalent of a `NOT` gate is the Pauli-X gate. It is a single qubit gate with matrix representation: $X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$. When applied to a qubit in the ground state, it will result in $X|0\rangle = |1\rangle$. The Pauli-X gate, along with the Pauli-Y and Pauli-Z gates, form a group of gates that allow us to rotate the state vector of a qubit around the x, y, and z axes of the Bloch sphere. The Bloch sphere is a useful way to visualize the state of a single qubit.

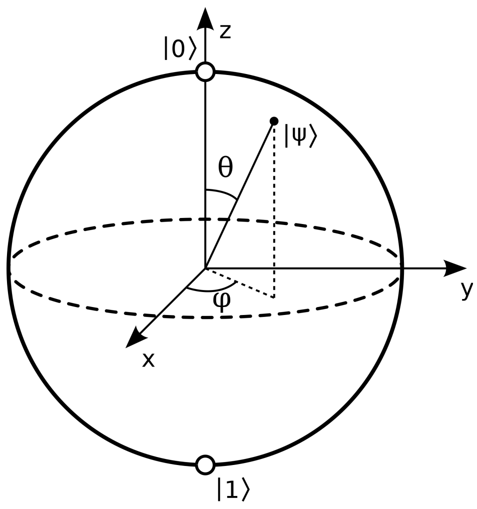

Figure 1: The Bloch sphere of a qubit. The north and south poles represent the basis states $|0\rangle$ and $|1\rangle$ respectively. $|\phi\rangle$ is the state vector representation in the Bloch sphere. The x, y, and z axes represent the axes on which Pauli-X, Pauli-Y, and Pauli-Z gates rotate the state.

Each Pauli gate will result in a 180-degree rotation around its corresponding axis. Exponentiating the Pauli gates gives the rotation gates $R_x(\theta), R_y(\theta)$ and $R_z(\theta)$ which rotate the state vector around the x, y, and z axes of the Bloch sphere by an angle θ. Another important single qubit gate is the Hadamard gate. It has the matrix representation: $H = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$ and when applied to a qubit in the ground state will result in $H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$, i.e., a superposition of the basis states.

2.2.1 Multi-qubit gates and entanglement

Gates can also be applied to multiple qubits. Let's consider a class of gates called controlled operators. Controlled operators act on multiple qubits and have a control qubit and a target qubit. The control qubit decides whether or not the target qubit is acted upon. A popular controlled operator, the `CNOT`, is commonly used to entangle qubits. To illustrate this point, let's take the Hadamard applied qubit from section 2.2 and add a second qubit to the system. This will give us: $\frac{1}{\sqrt{2}}(|00\rangle + |10\rangle)$. Applying a `CNOT` gate with the first qubit as control and the second qubit as target will result in: $\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$.

$$|\psi\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) \tag{8}$$

This state, also called an EPR pair, is interesting because it cannot be factored out into a tensor product of separate states. As a result, their measurement outcomes are correlated; the outcome of measuring the first qubit will fix the outcome of measuring the second qubit, regardless of the physical distance between them. Exploiting this mysterious property of quantum mechanics, labeled by Einstein as "spooky actions at a distance", is what enables quantum computers to do things that classical computers cannot.

2.3 Quantum circuits

What might have become apparent from the previous section is that it is often required to apply quantum gates to a system of qubits in a specific order to obtain a desired transformation. For example, to create the EPR pair mentioned in 2.2.1, one has to first apply a Hadamard gate to the first qubit followed by a $CNOT_{1,2}$ to achieve the required state. While this sequence is simple, one can imagine things could quickly escalate when dealing with high-dimensional systems with many operations. In order to describe and visualize operations on a quantum system, we use quantum circuits.

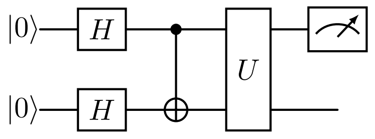

Figure 2: Example of a 2-qubit quantum circuit. Each wire in the circuit represents the passage of time of a single qubit. The qubits are initialized in the ground state $|0^{\otimes 2}\rangle$ and are acted upon by quantum gates, which are labeled boxes. U is an arbitrary unitary operator acting on 2 qubits. The black dot represents the control qubit of a controlled operation. A vertical line is drawn between the control and the $\oplus$, which represents the target of a CNOT operation. The final operation is a measurement operator on the first qubit.

It's important to note that quantum wires are not able to `fanout`, i.e., it is not possible to copy a qubit. This is a consequence of the no-cloning theorem, which states that it is impossible to create a copy of a quantum state.

2.3.1 Circuit cutting

Circuit cutting, also called circuit partitioning, is the process of dividing a quantum circuit into a set of smaller subcircuits. Each of these smaller subcircuits can be evaluated separately on smaller quantum machines; the results are then recombined on classical hardware to obtain the outcome of the original circuit. In the current NISQ era of quantum computing, where the number of qubits is limited, circuit cutting is a promising method to perform large computations that exceed the number of qubits available on a quantum device. Additionally, circuit cutting has proven useful as an error reduction scheme for quantum circuits even when the number of qubits is not a limiting factor [3].
There have been several proposals on cutting schemes. Our approach aligns with [4, 15]. Consider a quantum circuit U that acts on $n + k$ qubits, where n is the number of physical qubits and k represents the number of virtual qubits. In U, each qubit interacts with at most d two-qubit gates; we refer to these

circuits as d-sparse. By partitioning U into two subcircuits A and B such that $|A| = n$ and $|B| = k$, we can evaluate A on a quantum computer with $n+1$ qubits. This allows us to express U as:

$$U = \sum_{\alpha=1}^{\mathcal{X}} c_\alpha V_\alpha \otimes W_\alpha \tag{9}$$

1: Where c_α are coefficients such that $\sum_\alpha^{\mathcal{X}} |c_\alpha| = 1$ and $\mathcal{X} \equiv 2^{4kd}$.

Here, U can be expressed as a sum of the product of unitaries $V_\alpha \otimes W_\alpha$, where V_α acts on n qubits and W_α acts on k qubits. To show that the terms in the sum are bounded by 2^{4kd}, consider that a d-sparse circuit contains at most kd two-qubit gates connecting a qubit from partition A to partition B. Consider the set of gates in unitary U that link the two partitions, which we'll call $G_1, ..., G_m$ with $m \leqslant kd$. Each two-qubit gate $G[i,j]$, where $i \in A$ and $j \in B$, can be decomposed as $G[i,j] = \sum_\alpha^{16} c_\alpha P_\alpha[i] \otimes P_\alpha[j]$, with $P_\alpha \in \{I, X, Y, Z\}$. Applying this decomposition to every $G_1, ..., G_m$ and substituting U into the expectation value for some observable M (see Eq. (6)), we get:

$$\begin{aligned}
\mathbb{E}[M] &= \langle 0^{\otimes n} | U^\dagger M U | 0^{\otimes n} \rangle \\
&= \langle 0^{\otimes n} | \sum_\alpha^\chi \overline{c}_\alpha (V_\alpha \otimes W_\alpha)^\dagger (M_v \otimes M_w) \sum_\beta^\chi c_\beta (V_\beta \otimes W_\beta) | 0^{\otimes n} \rangle \\
&= \sum_{\alpha,\beta}^\chi \overline{c}_\alpha c_\beta \langle 0^{\otimes n} | V_\alpha^\dagger M_v V_\beta \otimes W_\alpha^\dagger M_w W_\beta | 0^{\otimes n} \rangle \\
&= \sum_{\alpha,\beta}^\chi \overline{c}_\alpha c_\beta \langle 0^{\otimes n} | V_\alpha^\dagger M_v V_\beta | 0^{\otimes n} \rangle \langle 0^{\otimes n} | W_\alpha^\dagger M_w W_\beta | 0^{\otimes n} \rangle
\end{aligned}$$

2: Substitution of our decomposed unitary U into the expectation value with some observable M.

Here, $\chi = 16^m$. For an arbitrary circuit U, it is essential to define a partition where the number of two-qubit gates connecting the two is minimal. The partitioned unitaries in the inner product differ only from the original circuit by the replaced two-qubit gates. Generalizing the expression above to T terms and K partitions yields:

$$\mathbb{E}[M] = \sum_{i=1}^{T} c_i \prod_{k=1}^{K} \langle 0^{\otimes n} | U_{i,k}^{'\dagger} M_k U_{i,k} | 0^{\otimes n} \rangle \tag{10}$$

3: Here, $M = \bigotimes_{k\in[K]} M_k$, $c_i \in \{c_i\}_{i\in[T]}$ and $\mathbb{U} = \{U_{i,k}^{'}, U_{i,k}\}_{i\in[T],k\in[K]}$ is a set of unitary operators acting on n_k qubits. K is the number of partitions and T is the number of terms in the sum.

The terms decompose into a product of inner products local to their partition, which can be evaluated on small quantum devices. For arbitrary two-qubit gates, the number of terms in the sum scales as 16^m where m is the number of gates that span across the partition. Storing duplicate terms can reduce the number of circuit evaluations to 6^m. For gates with a known Schmidt number s, the number of terms T scales as $\prod_{i=1}^{M} {s_i}^2$ where s_i is the Schmidt number of gate G_i.

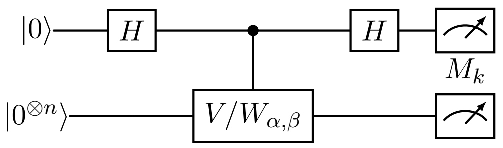

Figure 3: SWAP test of a qubit system

In order to evaluate the real component of the inner products, we have to perform a SWAP test. The controlled V/W gate implements V if the control is 0 and W if the control is 1. Fortunately, the unitaries $V/W_{\alpha,\beta}$ in each inner product only differ by the replaced gates, which limits the required controlled gates to a small set.

2.4 Noisy intermediate-scale quantum

The current era of quantum computing is referred to as the NISQ era, dubbed by Scott Aaronson as: the "Vacuum tube era" of quantum computing. This can be compared to the pre-transistor days of classical computing [20, 19]. The NISQ era, which stands for "Noisy intermediate-scale quantum," is characterized by quantum computers ranging from a few hundred to a thousand qubits with noisy quantum gates. Noise is caused by decoherence, which occurs when physical qubits leak information to their environment. In an ideal world, a quantum state would only reveal itself to its environment upon measurement, but in practice, this is difficult since we also need to perform non-invasive operations on the state. To remedy quantum leakage, researchers discovered quantum error correction, which entangles many physical qubits to create logical fault-tolerant qubits that are less prone to leakage. However, this comes with an approximate 1,000-fold increase in qubit overhead. Until quantum machines become bigger and more reliable, we are stuck with imperfect systems of limited size, forcing us to put more effort into reducing the number of qubits and limiting the depth of our circuits. As of this time of writing, the largest quantum computer claims to have 433 physical qubits [11].

3 Reinforcement Learning

3.1 Agents and Environments

In this section, we will introduce the basic concepts of reinforcement learning. Our discussion will primarily align with the terms and explanations provided in [26]. Reinforcement learning is a machine learning framework that revolves around learning to act. The entity that learns to act is called an Agent. Each timestep $t = 0, 1, 2, 3, 4...$, agents are given the ability to interact with an environment and observe its subsequent effects. Alongside this new observation, the agent experiences a reward R_t. This will result in a sequence of states S_t, actions A_t, and rewards R_t which we call a trajectory, often depicted as:

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3... \tag{11}$$

The interaction between an agent and its environment is often modeled as a Markov Decision Process, meaning that the current state and reward only depend on the immediate previous state S_{t-1} and action A_{t-1}.

The reward R_t is a numerical representation that describes how good of a state the agent is currently in and is defined by the Reward function. The main goal of an agent is to pick the action that will maximize its cumulative reward G_t.
A capable enough agent has to be able to balance short-term rewards for long-term success. Taking a pawn with a queen might increase your immediate score, but will likely lose you the game in the long run. Balancing short- and long-term rewards introduces the concept of discounted returns.

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} = \sum_{k=0}^{T} \gamma^k R_{t+k+1} \quad \gamma \in [0, 1) \tag{12}$$

Our discounted return at timestep G_t is calculated as the sum of all future rewards, each one decayed by a factor of γ. For each step, it represents the returns an agent is expected to receive. The discount rate γ can be tuned depending on how much value is put on immediate or long-term rewards. If we have a high degree of confidence we can predict future rewards, we set $\gamma \approx 1$. Conversely, if we are not certain that the current reward implies high future rewards, we set $\gamma \approx 0$. Each step, the agent tries to maximize its discounted returns. Therefore, the return can be seen as the target of our model;

3.2 Action value and policy gradient methods

Agents use value functions to estimate how good a state they are in, with foresight to possible future states. Value functions estimate how much reward can be expected in the future. However, these future reward depend on future actions. To make estimations we have to judge the value of our current state with respect to our current behavior. We introduce a mapping between actions and states. This mapping is called a policy $\pi(a|s)$, which represents the probability that $A_t = a$ if $S_t = s$.

$$v_\pi(s) = \mathbb{E}_\pi \left[G_t \mid S_t = s \right] \tag{13}$$

$$q_\pi(s, a) = \mathbb{E}_\pi \left[G_t \mid S_t = s, A_t = a \right] \tag{14}$$

The state value function, $v_\pi(s)$, is the expected return when starting in state s and following policy π. An extended version of the state value function is the action value function, which is equivalent to $v_\pi(s)$ conditioned on an action a and thereafter following policy π. Action-value method-based agents learn the action value function and then cleverly pick the next action based on the action value. We are going to focus on policy gradient methods which parameterize an approximation function of the policy and optimize the parameters $\boldsymbol{\theta}$ directly, without needing to learn the action-value function.

$$\pi_{\boldsymbol{\theta}}(a|s) = \Pr\left[A_t = a | S_t = s, \boldsymbol{\theta}_t = \boldsymbol{\theta}\right] \tag{15}$$

We introduce $\boldsymbol{\theta}$, which is a vector that parameterizes the policy. A common way to parametrize the policy in discrete action spaces is to use the softmax function, which provides a probabilistic selection over actions:

$$\pi_{\boldsymbol{\theta}}(a|s) = \frac{e^{f_{\boldsymbol{\theta}}(s,a)}}{\sum_{a'} e^{f_{\boldsymbol{\theta}}(s,a')}} \tag{16}$$

$f_{\boldsymbol{\theta}}(s, a)$ can be any function that is differentiable and expressive enough to encapsulate the relationship between the state and action space. The sum in the denominator is over all possible actions a'. The Softmax function ensures that the probabilities are positive and sum to 1, making it a suitable choice for policy representation in policy gradient methods. We also introduce $J(\boldsymbol{\theta})$ as a performance measure of the policy $\pi_{\boldsymbol{\theta}}$. Policy gradient methods often set $J(\boldsymbol{\theta}) = v_{\pi_{\boldsymbol{\theta}}}(s_0)$. Since our goal is to maximize $J(\boldsymbol{\theta})$, we perform gradient ascent on $J(\boldsymbol{\theta})$ with respect to $\boldsymbol{\theta}$.

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \tag{17}$$

Notice that we need a numerical expression for the policy gradient, $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$, to perform a parameter update on $\boldsymbol{\theta}$. We can derive this expression using the log-kickback trick, which results in:

$$\nabla_{\boldsymbol{\theta}} J\left(\pi_{\boldsymbol{\theta}}\right) = \underset{\tau \sim \pi_{\boldsymbol{\theta}}}{\mathrm{E}}\left[\sum_{t=0}^{T} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}\left(a_t \mid s_t\right) R(\tau)\right] \tag{18}$$

Here, we are dealing with an expectation value, which we can estimate by sampling trajectories $\tau \sim \pi_{\boldsymbol{\theta}}$ directly from our environment. This approach is analogous to the Monte Carlo method where we train on full trajectories. Let the batch $\mathcal{B} = \{\tau_i\}_{i=1,\ldots,N}$. Note that we update our policy only based on the batch from the current policy, not past ones. We refer to this class of RL algorithms as "on-policy". Algorithms that update their policy based on experiences from both the current and older policies are termed "off-policy". Sampling provides the following estimation for the policy gradient:

$$\hat{g} = \frac{1}{|\mathcal{B}|} \sum_{\tau \in \mathcal{B}} \sum_{t=0}^{T} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}\left(a_t \mid s_t\right) R(\tau) \tag{19}$$

The batch size, $|\mathcal{B}|$, can be used as a hyperparameter in our learning algorithm. Larger batch sizes lead to a more accurate estimation of the policy gradient, resulting in more stable learning. In environments where generating trajectories is costly or memory-intensive, one might consider reducing the batch size to expedite learning. For simplicity, we use $R(\tau)$ to weigh the log probability of each action a_t. However, in practice, it is not logical to weigh the gradient of an action a_t with the rewards of the entire trajectory since the returns of previous actions don't reflect the quality of the current action. To address this, we utilize the reward-to-go policy gradient, which replaces $R(\tau)$ with $\sum_{t'=t}^{T} R(s_{t'}, a_{t'}, s_{t'+1})$. Now, the gradient of an action a_t is influenced by rewards received after the action was taken.

3.3 Policy gradient learning algorithm

Put it together and we can define the fundamental policy gradient learning algorithm: REINFORCE, introduced by Ronald J Williams in 1992 [27].

Algorithm 1 REINFORCE

```
Require: A differentiable policy $\pi_{\boldsymbol{\theta}}(a|s)$
1: Initialize $\boldsymbol{\theta}$
2: while True do
3:     Collect batch $\mathcal{B} = \{\tau_i\}_{i=1,\ldots,N}$ following policy $\pi_{\boldsymbol{\theta}}$
4:     for Each $\tau_i$ in $\mathcal{B}$ do
5:         for Each time step $t$ in trajectory $\tau$ do
6:             Compute returns $G_t^i \leftarrow \sum_{t'=t}^{T} \gamma^{t'-t} r_t^i$
7:             Compute gradients $\nabla_{\boldsymbol{\theta}} log \pi_{\boldsymbol{\theta}}(a_t^i|s_t^i)$
8:         end for
9:     end for
10:    $\nabla\boldsymbol{\theta} = \frac{1}{|\mathcal{B}|} \sum_{i=1}^{N} \sum_{t=0}^{T} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}\left(a_t^i \mid s_t^i\right)\left(G_t^i - b_t^i\right)$
11:    Policy update $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \nabla\boldsymbol{\theta}$
12: end while
```

Note that the baseline b_t^i is subtracted from the discounted returns. b_t^i in the presented approach represents the mean of the discounted returns over the entire trajectory. This subtraction serves to reduce the variance in the policy gradient. Intuitively, one could say that agents are less influenced in states where they receive average rewards and are more attentive to states where future rewards deviate from the mean.

3.3.1 Evaluating performance of RL agents

Evaluating the performance of agents and their models is crucial. In supervised learning settings, learning performance is typically evaluated using the loss and accuracy metrics. While in RL, models do have a loss function that is minimized, this loss metric isn't usually an accurate measure of the agent's performance. The reason is that in supervised learning, the loss function is computed with fixed data independent of the model parameters. However, consider our on-policy algorithm: It calculates the loss in relation to trajectory data strongly correlated with its current parameters. Another distinction is that the loss function isn't the actual $J(\boldsymbol{\theta})$ we aim to maximize; in reality, the loss function is just a method to obtain the estimated negative gradient of performance. When we utilize these estimated gradients, we expect them to increase the expected sum of discounted rewards associated with that policy. However, if we were to minimize this loss function beyond the initial gradient step, there would be no assurance of improving $J(\boldsymbol{\theta})$. To genuinely assess the performance of our agent, we should observe the trend of average returns during training.

4 Quantum Reinforcement Learning

Quantum reinforcement learning (QRL) is a subfield of quantum machine learning (QML). There are various ways in which quantum processing can be beneficial in reinforcement learning settings. [10] proposes a schema that distinguishes between four categories: CC, CQ, QC, and QQ. The first symbol represents the internal structure of the agent, and the second represents its environment. (C) stands for classical and (Q) for quantum. These categories can be broadened to categorize not only QRL but also QML as a whole. In this paper, we will concentrate on the QC category, which entails a quantum-enhanced agent operating in a classical environment.

4.1 4.1 Hybrid RL agents

In our research, we focus on hybrid quantum-classical reinforcement learning agents. Hybrid agents typically quantize the model part of their learning algorithm while keeping other subroutines, like preprocessing, and optimization, on classical hardware. In our case, we will substitute the classical parameterized policy with a parameterized quantum circuit (PQC), which is also referred to as variational circuits or quantum neural networks. PQCs have been demonstrated to be successful in supervised learning tasks, such as classification and clustering, and recently, also in reinforcement learning settings. We largely follow the approach of Jerbi et al[12]., who have demonstrated that PQCs can be trained successfully to solve classical reinforcement learning environments with efficiency comparable to their classical counterparts using policy gradient-based methods.

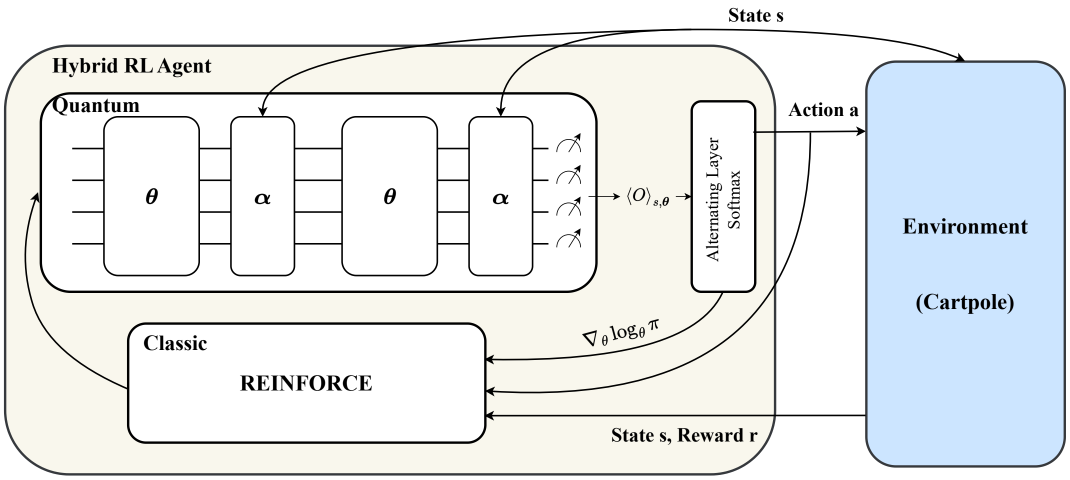

Figure 4: Architectural layout of the hybrid QC policy gradient RL agent

4.2 PQCs for quantum machine learning

PQCs are quantum circuits that can be tuned using paramatrized gates. We define our PQC as a unitary $U(\boldsymbol{s}, \boldsymbol{\theta})$ where $\boldsymbol{\theta}$ represents a set of trainable parameters which are optimized on classical hardware. We apply this unitary U to an n-qubit circuit in the ground state $|0^{\otimes n}\rangle$. The unitary encodes the input state $\boldsymbol{s} \in \mathbb{R}^d$, where d denotes the dimensionality of our input vector, by parameterizing gates in the circuit. If we measure the state using a specific observable M, it yields the following function:

$$f_{\boldsymbol{\theta},M,|\psi\rangle}(\boldsymbol{s}) = \left\langle 0^{\otimes n} \middle| U^\dagger(\boldsymbol{\theta}, \boldsymbol{s}) M U(\boldsymbol{\theta}, \boldsymbol{s}) \middle| 0^{\otimes n} \right\rangle \tag{20}$$

Given the relationship between the dimensions of the input vector $\boldsymbol{s}$ and the number of qubits in the PQC, it's a common practice to allocate a unique qubit for each dimension in the state vector $\boldsymbol{s}$. In the context of a 4-dimensional CartPole environment, this requires the instantiation of a 4-qubit PQC. Subsequent schematics in this thesis will adopt this 4-qubit PQC configuration. It's important to note that the actual configuration of the PQC, in terms of the number of qubits, is inherently determined by the dimensionality of the problem-specific input vector.

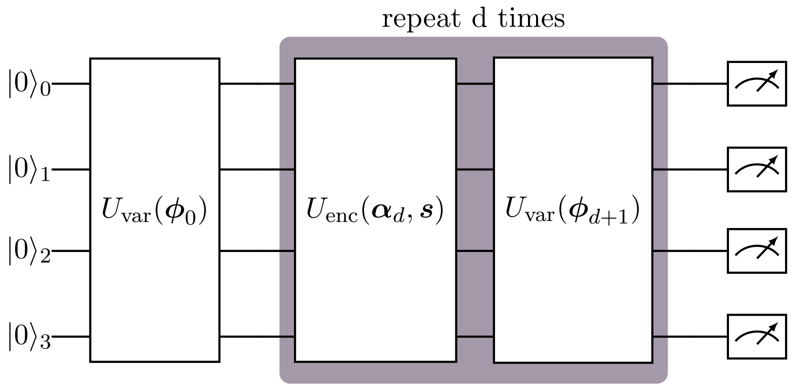
repeat d times
$|0\rangle_0$
$|0\rangle_1$
$|0\rangle_2$
$|0\rangle_3$
$U_{\text{var}}(\boldsymbol{\phi}_0)$
$U_{\text{enc}}(\boldsymbol{\alpha}_d, \boldsymbol{s})$
$U_{\text{var}}(\boldsymbol{\phi}_{d+1})$

Figure 5: Full layout of our PQC architecture using alternating layers

There is a substantial body of literature regarding the specific architecture of PQCs. In our implementation, we adopt the "alternating layer" approach. This scheme comprises a variational layer U_{var} followed by a depth d layer repeated both by U_{enc} and U_{var} unitaries, represented as $U(\boldsymbol{\theta}, \boldsymbol{s}) = \prod_{i=d}^{1}(U_{\text{var}}(\boldsymbol{\phi}_{i+1})U_{\text{enc}}(\boldsymbol{\alpha}_i, \boldsymbol{s}))\times U_{\text{var}}(\boldsymbol{\phi}_0)$ U_{enc} functions as an encoding layer for the input vector $\boldsymbol{s}$, while U_{var} is a circuit block for our trainable parameters $\boldsymbol{\phi}$. We can further subdivide U_{var} into two separate components. The first encompasses a series of single-qubit parameterized rotation gates, and the second consists of controlled-Z gates that entangle the qubits in the circuit.

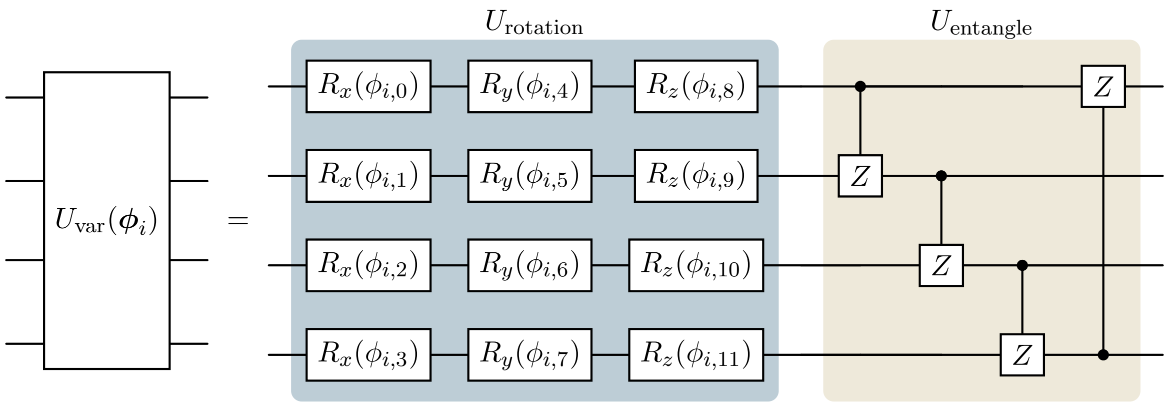

Figure 6: Subdivision of the U_{var} block of the alternating layer PQC. The U_{rotation} block consists of parameterized rotation gates, and U_{entangle} is a block of entangling gates in a ring-configuration.

The first segment of $U_{\text{var}}(\boldsymbol{\phi}_i)$, where $\boldsymbol{\phi} \in [0, 2\pi]^{|\boldsymbol{\phi}|}$, is U_{rotation}. This segment consists of single variational unitaries applied to each qubit. For each layer in our PQC, we apply variational X, Y, and Z rotation gates to the qubits. These gates apply rotations around each axis on the Bloch sphere, parameterized by $\phi_{i,j}$. The set of parameters $\boldsymbol{\phi}$ can be compared to the weights between hidden layers in a classical neural network. The second segment, U_{entangle}, consists of controlled-Z gates applied to each qubit in the PQC, entangling the qubits. We chose a ring configuration in which only neighboring qubits are connected. There are various possible configurations, including the full configuration that connects all possible pairs of qubits, but this requires significantly more controlled-Z gates. Limiting the number of controlled-Z gates is advantageous during partitioning.

The second block of our layer, U_{enc}, is where we encode the classical states into the quantum circuit. We do this by applying the product of a state vector $\boldsymbol{s} = (s_0, ..., s_d)$ and a set of trainable input scale paramaters $\boldsymbol{\alpha}_i = (\alpha_1, ..., \alpha_d)$, which is part of a vector $\boldsymbol{\alpha} \in [0, 2\pi]^{|\boldsymbol{\alpha}|}$ to single-qubit rotational R_x gates. This is shown in figure 7.

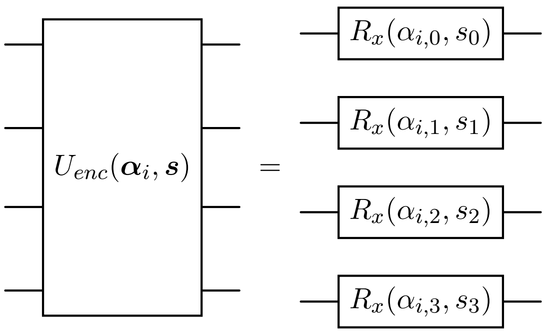

Figure 7: U_{enc} block of the alternating layer PQC.

We repeat this encoding segment for each layer in our model. We refer to these types of circuits as "data re-uploading" since they re-encode the data during the processing part of the computation [17]. Data re-uploading of the input encodings is primarily employed to circumvent the "no cloning theorem" in quantum mechanics. This theorem states that quantum information cannot be copied and this becomes problematic when we draw the analogy between quantum and classical neural nets. In the latter, inputs are reused many times between the nodes inside a hidden layer. With data re-uploading, we create duplicates of the input data on classical hardware and encode it into the PQC for each layer in the model.

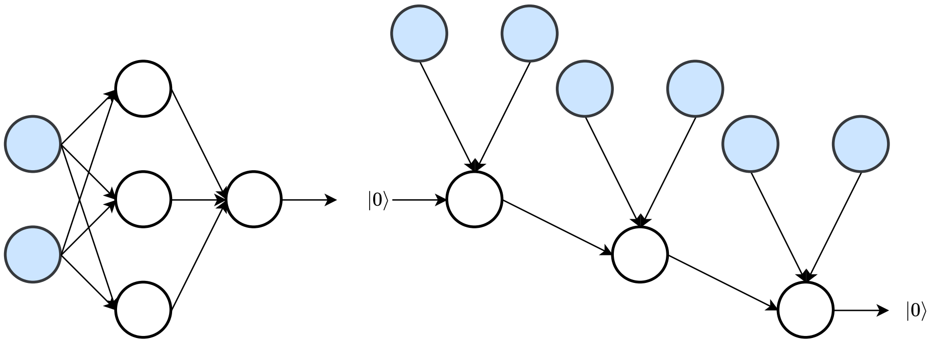

Figure 8: Left: Classical neural network process, Right: Quantum NN process

Consequently, we replicate the ability for each hidden layer node to access the input data, positively impacting the expressivity of the model. Another noticeable analogy between quantum and classical neural nets is that the increasing depth d of the PQC dictates the number of rotation gates in the circuit, effectively increasing the number of trainable parameters and thereby enhancing the expressivity of the PQC. This makes depth a powerful hyperparameter during training, comparable to the number of hidden layers in a classical neural network.

4.3 Quantum policies

There are two main approaches for associating a quantum state with a policy. The first one, called RAW-PQC, leverages the fundamental probabilistic nature of quantum measurements to retrieve a stochastic policy. In this particular approach, we partition the Hilbert space $\mathcal{H}$ into $|A|$ distinct subspaces spanned by the computational basis states and define a projection operator P_a for each of these subspaces. Using the projection operators, we can construct the observable $O = \sum_a aP_a$. Then, we can define a raw PQC policy associated with the state $|\psi_{s,\boldsymbol{\theta}}\rangle = U(\boldsymbol{s}, \boldsymbol{\theta})|0^{\otimes n}\rangle$ as:

$$\pi(a|s) = \langle P_a \rangle_{s,\boldsymbol{\theta}} = \langle \psi_{s,\boldsymbol{\theta}} | P_a | \psi_{s,\boldsymbol{\theta}} \rangle. \tag{21}$$

Where $\sum_a P_a = I$ and $P_m P_m' = \delta_{mm'} P_m$, i.e., the set of projection operators P_a form a complete set of orthogonal projectors in the Hilbert space. We will extend the aforementioned RAW-PQC to a SOFTMAX-PQC policy by making some generalizations to the projection operators. Firstly, we replace the projection operators P_a with arbitrary Hermitian operators O_a and assign trainable weights $w_{a,i}$ to them, resulting in action-specific observables $O_a = \sum_i w_{a,i} H_{a,i}$. This generalization allows us to create very complex trainable observables. The authors of [12] suggest using only tensor products of Pauli operators or high-rank projections on the computational basis states as Hermitian components $H_{a,i}$. They found that this approach prevents the observable from being too expressive and effectively overpowering the trainability of the PQC itself. For our implementation, we chose $H_1 = H_2 = Z_1 Z_2 Z_3 Z_4$ paired with weights $\boldsymbol{w} = [1, -1]$. The generalization from projection operators to arbitrary Hermitian operators O_a also results in arbitrary expectation values. To transform these expectation values into a required probability distribution $[-1, 1] \to [0, 1]$ for a valid policy $\pi_{\boldsymbol{\theta}}(a|s)$, we need to normalize the outputs using the softmax function:

$$\pi_{\boldsymbol{\theta}}(a \mid s) = \frac{e^{\beta \langle O_a \rangle_{s,\boldsymbol{\theta}}}}{\sum_{a'} e^{\beta \langle O_{a'} \rangle_{s,\boldsymbol{\theta}}}} \tag{22}$$

Here, $\langle O_a \rangle = \langle \psi_{s,\boldsymbol{\theta}} | \sum_i w_{a,i} H_{a,i} | \psi_{s,\boldsymbol{\theta}} \rangle$ is the expectation value of the operator H_a associated with state a. Note that we also introduce a parameter β in the softmax function. This parameter is called the inverse-temperature parameter and controls the stochasticity of the policy. A higher β will result in a high exploration rate, while a lower beta will lead to more exploitation of the current best action. This is a common hyperparameter in reinforcement learning since it prevents the policy from getting stuck in local optima.

It is important to note that $\langle O_a \rangle_{s,\boldsymbol{\theta}}$ is an analytical expression of the expectation value which classical simulators of quantum devices are able to compute. However, on real quantum machines, one has to approximate this expectation value, denoted by $\widetilde{\langle O_a \rangle}_{s,\boldsymbol{\theta}}$, by repeatedly sampling circuits from the quantum machine. Fortunately, these quantum expectation values can be efficiently sampled up to some additive error[1]ε which [12] has used to show that an efficient estimation $\widetilde{\pi}_{\boldsymbol{\theta}} = \text{softmax}_\beta(\widetilde{\langle O_a \rangle}_{s,\boldsymbol{\theta}})$ can be made with a total variational distance from the true policy $\pi_{\boldsymbol{\theta}}$ of $\mathcal{O}(\beta\varepsilon)$, where $|\widetilde{\langle O_a \rangle}_{s,\boldsymbol{\theta}} - \langle O_a \rangle_{s,\boldsymbol{\theta}}| \leqslant \varepsilon$.

4.3.1 Quantum policy gradient

As stated in 3.2 the gradient ascent step requires an expression for the gradient of the log policy $\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a|s)$. We can derive this expression for our softmax PQC policy as follows:

[1]Additive error is an error added to the true value which does not depend on the true value itself.

$$\begin{aligned}
\nabla_\theta \log \pi_\theta(a|s) &= \nabla_\theta \log \frac{e^{\beta\langle O_a\rangle_{s,\boldsymbol{\theta}}}}{\sum_{a'} e^{\beta\langle O_{a'}\rangle_{s,\boldsymbol{\theta}}}} \\
&= \nabla_{\boldsymbol{\theta}} \log e^{\beta\langle O_a\rangle_{s,\boldsymbol{\theta}}} - \nabla_{\boldsymbol{\theta}} \log \sum_{a'} e^{\beta\langle O_{a'}\rangle_{s,\boldsymbol{\theta}}} \\
&= \beta \nabla_{\boldsymbol{\theta}} \langle O_a\rangle_{s,\boldsymbol{\theta}} - \sum_{a'} \frac{e^{\beta\langle O_{a'}\rangle_{s,\boldsymbol{\theta}}} \beta \nabla_{\boldsymbol{\theta}} \langle O_{a'}\rangle_{s,\boldsymbol{\theta}}}{\sum_{a''} e^{\beta\langle O_{a''}\rangle_{s,\boldsymbol{\theta}}}} \\
&= \beta \nabla_{\boldsymbol{\theta}} \langle O_a\rangle_{s,\boldsymbol{\theta}} - \beta \sum_{a'} \pi_{\boldsymbol{\theta}} \left(a' \mid s\right) \nabla_{\boldsymbol{\theta}} \langle O_{a'}\rangle_{s,\boldsymbol{\theta}}
\end{aligned}$$

4: Derivation of the log-policy gradient.

Here $\nabla_\theta \langle O_a\rangle_{s,\boldsymbol{\theta}}$ requires the partial derivatives with respect to the observable weights $\boldsymbol{w}$, the rotational angles $\boldsymbol{\phi}$ and the input paramaters $\boldsymbol{\alpha}$. Finding the gradient of paramatrized gates is done using the paramater shift rule [21]. This trick requires us to calculate the expectation value twice with one circuits parameters shifted by a fixed value. The difference between the two is used to calculate the gradient:

$$\partial_i \langle O_a\rangle_{s,\boldsymbol{\theta}} = \frac{1}{2} \left(\langle O_a\rangle_{s,\boldsymbol{\theta}+\frac{\pi}{2}\boldsymbol{e_i}} - \langle O_a\rangle_{s,\boldsymbol{\theta}-\frac{\pi}{2}\boldsymbol{e_i}} \right). \tag{23}$$

4.4 PQC partitioning

Partitioning our PQC into smaller subcircuits presents a set of intriguing challenges. Consider a unitary following the architecture of the PQC defined in 5. Each layer of $U(\boldsymbol{\theta}, \boldsymbol{x})$ contains U_{entangle}, which houses all of the two-qubit controlled-Z gates. Decomposing the n-qubit $U(\boldsymbol{\theta}, \boldsymbol{x})$ into two partitions A and B would necessitate the expansion of 2 controlled-Z operators for each depth d of the circuit layer. If $\{G_1, ..., G_m\}$ represents the set of Controlled-Z gates connecting partitions A and B, then $m = 2d$. Controlled-Z gates can be decomposed into the following sum of single-qubit operators:

$$\text{Controlled} - Z = \frac{1}{1+i} \left(S \otimes S + i S^\dagger \otimes S^\dagger \right) \tag{24}$$

Combining this fact with observations from 2.3.1 suggests that the number of terms to be evaluated in a 2-partitioned PQC scales as 4^{2d}. Building on our previous definition of the expectation value of a partitioned unitary from 1, we obtain:

$$\tilde{f}_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sum_{i=1}^{4^{2d}} c_i \prod_{k=1}^{K} \left\langle 0 \left| U'^{i,k\dagger} (\boldsymbol{\theta}, \boldsymbol{x}) M_k U^{i,k} (\boldsymbol{\theta}, \boldsymbol{x}) \right| 0 \right\rangle \tag{25}$$

Where $\tilde{f}_{\boldsymbol{\theta}}(\boldsymbol{x}) = f_{\boldsymbol{\theta}}(\boldsymbol{x})$ as given by Eq. (20). Using a circuit architecture of equal dimensions as the one in [12] with 5 layers means that the number of subcircuit evaluation for a single passthrough is six magnitudes higher than the circuit evaluations required for an unpartitioned circuit, rendering it infeasible to use a term-complete partitioned PQC in practice. Fortunately, [15] arrived at the same conclusion and proposed a scheme that significantly reduces the number of terms that need to be evaluated in the partitioned PQC.

4.4.1 Reduced PQC partitioning

Reducing the number of terms in our PQC is acceptable under the assumption that we do not require the exact output of our circuit to solve the learning problem. Instead, we intend to make a reasonable approximation of the full PQC. Under this premise, we can limit the number of terms T in the sum to a fixed number L. This hyperparameter L will serve as our runtime budget, and setting it to an appropriate value will reduce the required circuit evaluations to a computationally feasible amount.

$$\tilde{f}^I_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sum_{i \in I} \lambda_i \prod_{k=1}^{K} \left\langle 0 \left| U'^{i,k\dagger}(\boldsymbol{\theta}, \boldsymbol{x}) M_k U^{i,k}(\boldsymbol{\theta}, \boldsymbol{x}) \right| 0 \right\rangle \tag{26}$$

Here $I \subseteq \mathbb{U}$ is the optimal subset of L small circuits in $\mathbb{U}$ that approximates the full circuit. We have introduced a new trainable parameter $\boldsymbol{\lambda}$ which can be optimized during training for the best approximation of $\tilde{f}_{\boldsymbol{\theta}}(\boldsymbol{x})$. This model introduces an exponential reduction in the number of circuits required to approximate $\tilde{f}_{\boldsymbol{\theta}}(\boldsymbol{x})$; however, it requires us to somehow find the optimal subset of small circuits $I \subseteq \mathbb{U}$. This boils down to a combinatorial optimization problem which is impractical to solve during training. In order to circumvent this we use the fact that the smaller circuits only differentiate by their partitioned gates 2.3.1. Parameterizing all of the partitioned gates with a trainable parameter reduces the challenge of finding the optimal subset of small circuits to finding the optimal parameters of the partitioned gates.

$$\bar{f}_{\boldsymbol{\theta},\boldsymbol{\zeta},\boldsymbol{\lambda}}(\boldsymbol{x}) = \sum_{i \in [L]} \lambda_i \prod_{k \in [K]} \left\langle 0 \left| U^{k\dagger}\left(\boldsymbol{\theta}, \boldsymbol{x}, \zeta_{i,k}\right) M_k U^{k}\left(\boldsymbol{\theta}, \boldsymbol{x}, \zeta_{i,k+K}\right) \right| 0 \right\rangle \tag{27}$$

Such that $\tilde{f}_{\boldsymbol{\theta}}(\boldsymbol{x}) = \overline{f}_{\boldsymbol{\theta},\boldsymbol{\lambda},\boldsymbol{\zeta}}(\boldsymbol{x})$ for every $\boldsymbol{\theta}$ and $\boldsymbol{x}$. By adjusting the trainable $\boldsymbol{\lambda_i}$ and $\boldsymbol{\zeta_{i,k}}$ parameters, the model can represent each of the blocks of smaller circuits in the sum. This reduces the number of unitaries in the set $\tilde{\mathbb{U}}$ from $\{U^{i,k}(\boldsymbol{\theta}, \boldsymbol{x}), U^{i,k}(\boldsymbol{\theta}, \boldsymbol{x})\}$ to $\{U^{k}(\boldsymbol{\theta}, \boldsymbol{x}, \boldsymbol{\zeta})\}_{k \in [K]}$, i.e., one unitary per partition. Note that this scheme introduces a total of L $\boldsymbol{\lambda}$ and $4Ld$ $\boldsymbol{\zeta}$ trainable parameters to the model.

Gradient-based methods involve calculating the derivative of a partitioned PQC with respect to some trainable parameter in the circuit. For our partitioned circuit, this requires the application of the chain rule to each of the inner products in the individual terms. Since the parameter usually sits on only one side of the partition, most of the terms will evaluate to 0. However, since we do need to evaluate the partial derivative with respect to each term, the number of calculations required to compute the gradient scales with L times the number of evaluations needed for the gradient of a single smaller circuit.

5 Related Work

In recent years, several works have spurred the motivation of this research. Prominent among these works, is a novel hybrid reinforcement learning approach where the authors successfully used a hybrid quantum-classical policy gradient agent to solve several classical, but also quantum-based RL-environments [12]. The authors introduced a novel Softmax-PQC model, which greatly improved the expressibility and flexibility of previous PQC policies, thereby achieving comparable learning performances to fully classical deep neural networks, both using gradient-based methods. Furthermore, they demonstrated an empirical advantage of PQC-agents over DNN agents in environments which are generated by PQCs. It is worth noting that several other works have attempted a hybrid agent comparable to Jerbi et al. [7] used a similar PQC-agent but using a value-based approach benchmarked on the frozen lake environment, notably going beyond numerical simulations and successfully running it on a real quantum machine. [25] has successfully used value-based methods to solve the CartPole environment. Both Jerbi and Skolik et al. used a 4-qubit quantum machine in their attempts to solve CartPole. [28] has implemented a combination of the two methods mentioned above, the so-called actor-critic method (DDPG); however, this approach is fundamentally different from the one we use since the data input from the environment is quantum.

There is a sizable body of research related to circuit partitioning schemes. [4], discussed more thoroughly in 2.3.1, showed that any $n+k$ qubit circuit can be run on n qubits with a $2^{O(k)}poly(n)$ simulation overhead. [18] showed we can reduce te computational overhead of gatecutting if we allow classical communication between the partitions. These circuit cutting methods suffer from exponential overhead making them intractable for most machine learning problems. The approach this thesis will focus on is the work from [15], where a novel approach of reduced circuit partitioning was introduced. The main deviation from other methods is that they relax the requirement of requiring the exact output of the original circuit and instead make a trainable model that learns an optimal apporximation within a computational budget, as we have already laid out more in depth in 4.4.1. [15] managed to successfully train the fundamental supervised learning task, MNIST, on a 8-qubit quantum machine, where the original implementation of an 8x8 handwriting grid would require 64 qubits. They have also shown that you can get an acceptable approximation of a random unpartitioned circuit using the reduced partitioned model. However, they left open the question as to whether their approach would be useful on other learning tasks.

6 Problem Statement

The primary goal of this thesis is to determine whether the reduced partitioned models obtained by applying cutting methods from [15] are also useful in a reinforcement learning setting, and if so, how they perform in comparison to unpartitioned circuits. For this, we will consider the MNIST equivalent of reinforcement learning, Cartpole, which is a classic control problem used to benchmark RL algorithms. If we can successfully solve Cartpole, the scheme could be extended to more challenging and complex reinforcement learning environments. Moreover, [15] found that in using a reduced partition model, there is a varying correlation between the number of terms and the accuracy of the model. More specifically, solving MNIST saw significantly less benefit in the loss/terms ratio than using the reduced partitioned model to approximate a random unpartitioned instance of itself. Therefore, a suitable follow-up question is how strong the relationship is between the number of terms and the accuracy of the reduced model in a reinforcement learning environment like CartPole. Intuitively, MNIST, which is a classification problem, is a more static problem than reinforcement learning. That is, the training data is static, and the associated labels are deterministic (a well-trained model should give the same classification for the same digit). Reinforcement learning, on the other hand, is fundamentally more dynamic and stochastic. The data on which the model is optimized is a function of its own parameters, and we optimize on a moving target. It is unclear how the partition learning process will behave in such environment. Intuitively, MNIST would benefit more from an accurate representation of the underlying model than CartPole and would, therefore, see a stronger relationship between the number of terms and model accuracy. However, as mentioned in 3.3.1, the accuracy of our model does not necessarily tell us how well our agent performs. To truly compare the performance of our reduced partition model with respect to the number of evaluated terms, we must examine the rewards received by the agent. Determining the strength of this relationship in CartPole can be accomplished in an experimental setting where we train the reduced model on a range of terms and compare the learning performance.

7 Methodology

In this section, we describe the methods used to investigate whether circuit partitioning can be used in reinforcement learning. First, we describe how we implemented the reduced partitioned model inside a policy gradient agent. Then, we show the environment in which we will benchmark our agent, and finally, we describe the experiments we will run to test our hypothesis.

7.1 Implementation of the quantum model

Our implementation of the reinforcement learning agent itself, is largely in line with the approach taken in [12]. The authors have thoroughly benchmarked an unpartitioned implementation of the model on commonly used reinforcement learning environments. This approach and implementation will therefore function as the baseline of our implementation, while we essentially substitute the quantum model for a partitioned one.
To do this, we will use Tensorflow-Quantum(TFQ) [6], which is a quantum extension for Tensorflow that functions as an ML-purposed abstraction layer for quantum computing programming libraries, in particular, Cirq [9], which is used to define and manipulate quantum circuits. As stated in 4.3, classically simulated circuit evaluations have the benefit of having an analytical solution, which can be used for the required expectation values, removing the need for multiple measurement shots. Since these qubits are simulated classically, they can be done so without the burden of decoherence or other noise-related errors. While it is possible to simulate noise in TFQ, we will not do so in any of our experiments. TFQ will also simplify the gradient calculations of our model. Under the hood, it will create a computational graph of our model, which will help it keep track of the operations we perform during a forward pass. Then, during the backward pass, it will traverse this list to compute the gradients of each operation using automatic differentiation. In 4, we mentioned that we use a parameter-shift rule to compute the gradients of our parametrized gates. This method is also attainable in TFQ, but by default, it will use the more efficient adjoint differentiation method, which requires only one forward pass and two backward passes for the gradient of an arbitrary rotation gate. This method, however, is not possible on real quantum machines.

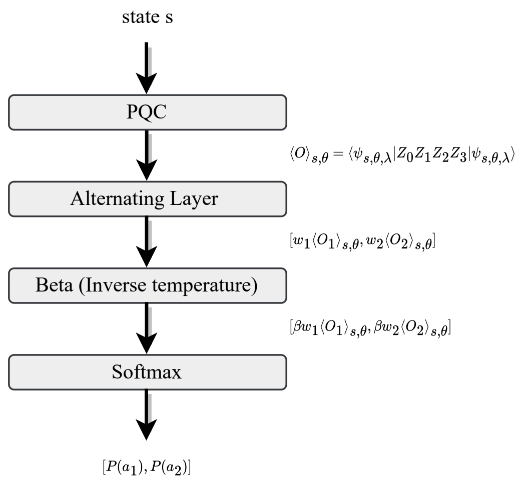
state s
PQC
$\langle O\rangle_{s,\theta} = \langle\psi_{s,\theta,\lambda}|Z_0Z_1Z_2Z_3|\psi_{s,\theta,\lambda}\rangle$
Alternating Layer
$[w_1\langle O_1\rangle_{s,\theta}, w_2\langle O_2\rangle_{s,\theta}]$
Beta (Inverse temperature)
$[\beta w_1\langle O_1\rangle_{s,\theta}, \beta w_2\langle O_2\rangle_{s,\theta}]$
Softmax
$[P(a_1), P(a_2)]$

Figure 9: Overview of each layer in our custom Tensorflow model.

Our model consists of four custom Tensorflow layers, starting with the PQC. For our cartpole environment, we will measure this PQC with the $Z_0Z_1Z_2Z_3$ observable, which will result in some expectation value $\langle O\rangle_{s,\boldsymbol{\theta}}$. We will pass this expectation value through an alternating layer, which consists of a simple dense layer of action-specific observation weights $\boldsymbol{w} = (w_0, w_1)$, initialized to $[1, -1]$. These trainable weights will be used to obtain the action-specific expectation values from our measurement. Subsequently, these action-specific expectation values are passed through a Beta layer, followed by the softmax layer to obtain the action probabilities.

For the implementation of the PQC, we draw heavily from the implementation by [15]. Remember from 4.4.1 that we reduced the set of required unitaries to just two unitaries local to their partition. Furthermore, since our full PQC is mostly symmetrical across the cuts, we can get away with defining just one single circuit and associating to each partition and term the correct parameters. This is illustrated in 10. The circuit is defined as follows:

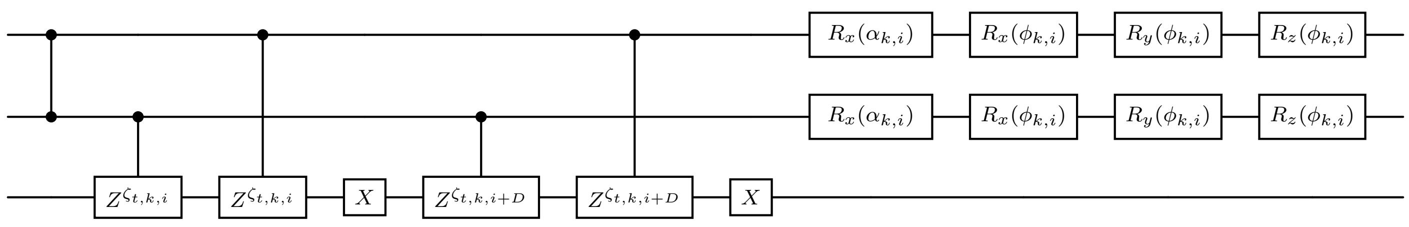

Figure 10: Single layer $i \in [D]$, partition $k \in [K]$ of our circuit implementation representing a single inner product of term $t \in [L]$ from 1.

Parameter	Size
$\boldsymbol{w}$	$\|A\|$
$\boldsymbol{\phi}$	$(3 \times N_{\text{qubits}}) \times (d+1)$
$\boldsymbol{\alpha}$	$N_{\text{qubits}} \times d$
$\boldsymbol{\lambda}$	L
$\boldsymbol{\zeta}$	$4 \times L \times K \times d$

Table 1: Overview of defined trainable parameters where $|A|$ is the size of the action space, d is the depth of the circuit, L is our runtime budget and K is the number of partitions in our circuit.

The controlledPQC operator from TFQ allows us to pass the inner product specific parameters alongside the circuit evaluation in order to obtain the correct expectation value of the subcircuit. Each call to our model, we iterate through the terms and their respective inner products, evaluate the relevant subcircuit, store the intermediate results, and sum them up to retrieve the final PQC output. An unfortunate yet important caveat of this approach is that we are not using the optimized Tensorflow parallelization methods, which implies that individual terms are evaluated sequentially. This severely impacts the time efficiency of our model, which, to exacerbate matters, does not leverage GPU compute for its matrix multiplications unlike regular TF models. This implies that, despite all efforts of reducing the number of terms, it is still quite time-consuming to run the scheme in a reinforcement learning setting on regular consumer-grade hardware, particularly once we scale up the number of terms. For future implementations, one would ideally want to use optimized Tensorflow methods to speed up the evaluation of the PQC.
Furthermore, we assign to each trainable parameter in the model its own optimizer so we can select different learning rates for each parameter. This will prove itself useful in subsequent sections. Each trainable variable in our model uses the Adam optimizer.

7.2 The CartPole Environment

To demonstrate the learning capabilities of our partitioned PQC, we use it to solve the classic CartPole environment. CartPole is often used as a benchmark for RL algorithms because it is intuitive to understand yet complex enough to illustrate the key challenges in reinforcement learning [5]. In essence, what MNIST is to supervised learning, Cartpole is to Reinforcement learning: a very well-suited testbed for both simple and complex implementations, as shown by [15], where they successfully used the partition scheme to solve MNIST.

The objective is to keep the stick upright as long as possible. We can represent its action space a as:

$$a = \{0, 1\}$$

where 0 = "Move cart Left" and 1 = "Move cart Right". The observation space can be described as:

$$s = [x, \dot{x}, \theta, \dot{\theta}] \in \mathbb{R}^4$$

Here x is the position of the cart, $\dot{x}$ its velocity, θ the pole angle, and $\dot{\theta}$ the pole angle velocity.
At the start of each trial, the pole is initialized in a random position and angle. The reward function r_t is one every time-step the pole stays upright. Whenever the pole reaches a certain angle or the cart a certain position, the state becomes terminal and the episode ends. The termination state is also given when we reach a reward $r_t = 500$. We consider an environment solved whenever an agent can consistently reach the max reward across multiple trials.

7.3 Inspecting the Circuit Implementation

To verify the accuracy of the circuit's implementation, we utilize a supervised learning setting for training. Initially, we construct a 4-qubit circuit of depth 5, mirroring the dimensions in the study by [12]. This full circuit is then used to generate 1000 random outputs, and we reformat its $\boldsymbol{\theta}$ weights to the 2-qubit partitioned PQC. During the training process, we retain these parameters in a fixed state to solely focus on training the gates used in the partitioning scheme. A near-zero loss implies that our partitioned circuit can approximate an unpartitioned version of itself with high accuracy, thereby validating the correctness of our implementation.

7.4 Hyperparameter Search on Trainable Parameters

Identifying the optimal hyperparameters is an maticolous process in machine learning, particularly in on-policy reinforcement learning where data is sampled from a non-stationary distribution due to consistent policy updates. High learning rates often result in divergence and instability, while low learning rates lead to slower training and inadequate regularization. To empirically demonstrate the feasibility of using reduced partitioned schemes in reinforcement learning, we must start by obtaining a learning curve that reveals even a slight indication of the agent's learning ability. Initial test runs of the implementation, with all newly introduced learning rates set within the same order of magnitude as [12], showed promising initial learning followed by swift plateauing. Such a pattern is indicative of potential hyperparameter issues. To eliminate any hyperparameter-related concerns, we decided to conduct a sweep of all PQC-specific hyperparameters. Detailed configurations of this sweep can be found in appendix A.

7.5 Definition of lambda paramters

Initially, some agents either got trapped in a min-reward situation, signifying an uninterrupted left or right movement leading to immediate termination, or if they surpassed this initial phase, they succumbed to catastrophic forgetting. This phenomenon refers to the unlearning of previously acquired behavior, resulting in a collapse of the reward function. Agents getting trapped in min-reward is peculiar behavior, likely indicative of a model-related issue. The implementation we adopted from [15] used $exp(\boldsymbol{\lambda})$ for the λ parameters in the sum. While this approach proved successful in the MNIST and synthetic data experiments, it might have been a source of issues in our implementation. This is primarily because the lambda parameters at the beginning of the sum considerably influence our circuit output as all other model-related parameters in each term are directly scaled by this single parameter. To ascertain whether this parameter was the root of the problem, we decided to compare three distinct definitions of our lambda parameters.

1. $exp(\boldsymbol{\lambda})$, i.e., the original implementation.
2. $\boldsymbol{\lambda}$, i.e., no exponent, initializing each term to 1.
3. $\boldsymbol{\lambda} \times \frac{1}{L}$, i.e., no exponent, initializing each term to $\frac{1}{L}$.

We test each method on agents with 3 and 6 terms and compare their learning curves (Appendix: B). In method 1, the lambdas are automatically constrained to be positive. For methods 2 and 3, we add a non-negative constraint to the trainable parameter, ensuring positive values. However, we must be cautious when setting lambda to a small value, as the output of the inner product is inherently small. Initializing lambda to an already small value might cause some terms to vanish during training. Consequently, we also analyze the lambda distribution of the factoring method.

7.6 Term Comparisons

To evaluate the impact of the number of terms on training capabilities, we train agents with 1, 3, 6, 10, and 20 terms and compare the learning curves. Each agent will have identical hyperparameters, and each average is the temporal average of 10 episodes for 5 concurrently running agent (Appendix: C). However, this average does not convey the complete picture of the learned policy. Agents could have similar average performance while the actual learned policies significantly differ. To delve deeper into potential differences, we conduct a comprehensive sweep of individual policies to determine if there are any visible changes. Due to the multidimensionality, we execute a full sweep of two dimensions while setting the remaining two to zero. For each point, we calculate the pre-softmax probabilities of taking the 'left' action. Additionally, we compute the mean squared error of the partitioned outputs relative to the unpartitioned one to ascertain whether the inclusion of more terms improves the approximation of the partitioned circuit with respect to an unpartitioned one. This ofcourse is no strong evidence, but rather a rough indication, of model accuracy since agents learn slightly different policies each training run.

7.7 Reshuffed Inputs

In 7.2, we laid out the observation space received from the cartpole environment. These individual data points are allocated to a single qubit in our circuit. By cutting the circuit in half, we effectively create a separation between these data points; more specifically, we place $(x, \dot{x})$ on one partition of the model and $(\theta, \dot{\theta})$ on the other. This means that in our default experiment setting, we retain the uncut circuit gates between the two derivative pairs. An experiment was conducted to check whether previously obtained results could be replicated by placing both derivative pairs on each side of the partition. Results showed that the default pair exhibited slightly more initial divergence from the mean. Both methods converged to each other in later training stages. These results suggest that previously obtained results will generalize to other permutations of the inputs (Appendix: E).

7.8 Training

As stated in 7.1, training these models, particularly those with a high number of terms, is time-intensive. This is due to the significant computational overhead incurred by the use of a custom TensorFlow layer for our circuit implementation. The only viable strategy for conducting multi-trial experiments is in parallel on a CPU cluster. For job scheduling and cluster management, we used Ray Tune, a popular ML hyperparameter tuning library that abstracts many cluster management tasks[13].

8 Results

8.1 Initial Test Results

The results from our initial implementation test are presented below. The findings suggest that our reduced, partitioned circuit can accurately approximate the unpartitioned circuit using five terms. These results confirm that the circuit in our agent is capable of approximating an unpartitioned version of itself by adjusting its $\boldsymbol{\lambda}_{\text{term}}$ and $\boldsymbol{\zeta}$ parameters.

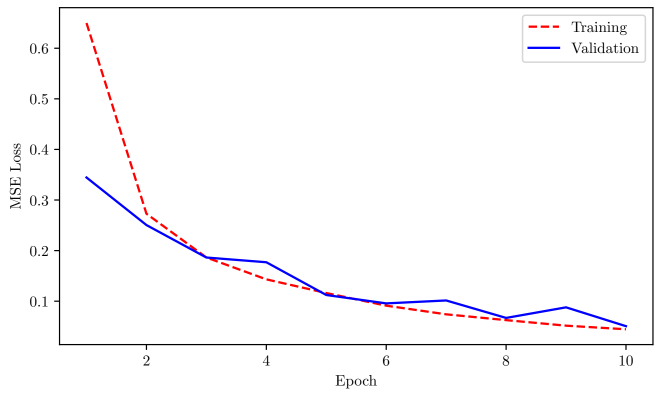

Figure 11: Testing the implementation of our partitioned PQC by approximating a 4-qubit unpartitioned PQC on two 2-qubit partitions. The $\boldsymbol{\theta}$ parameters were fixed, and the training focused only on the parameters intended to replicate the partition.

8.2 Optimal Hyperparameter Search

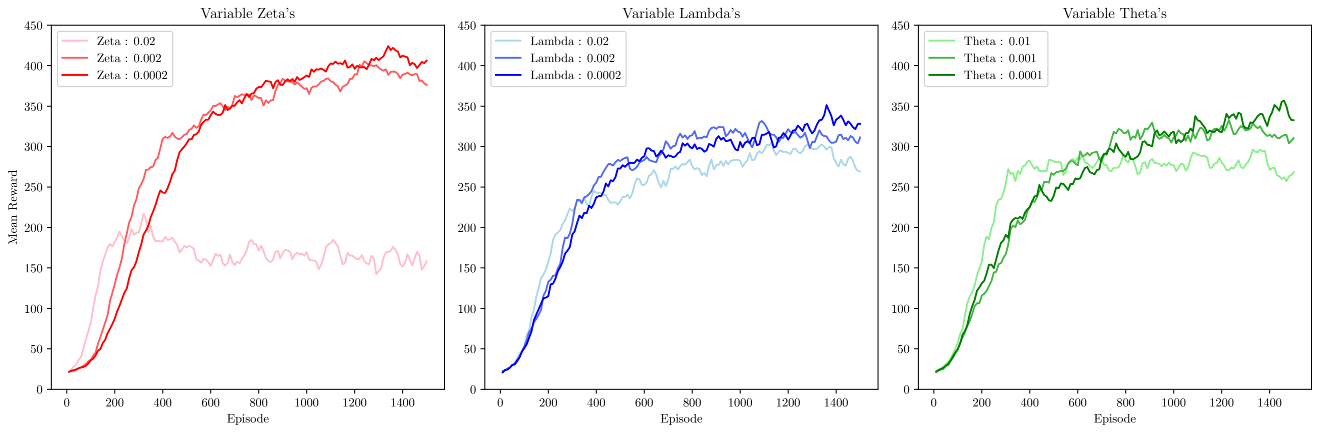

Figure 12: Side-by-side results of the hyperparameter grid search, organized by single parameters.

The hyperparameter search reveals a slight preference for lower learning rates in the case of both the $\boldsymbol{\lambda}$ and $\boldsymbol{\theta}$ parameters. We observe that most of the benefits from lower learning rates occur in more experienced agents, without a noticeable trade-off in the initial learning take-off. Regarding the $\boldsymbol{\zeta}$ parameters, there appears to be a stronger correlation between learning rate and initial learning speed. Furthermore, the agents display a strong aversion to high $\boldsymbol{\zeta}$ learning rates, where the agents get stuck in a local optimum.

8.3 Initializing Lambda Parameters

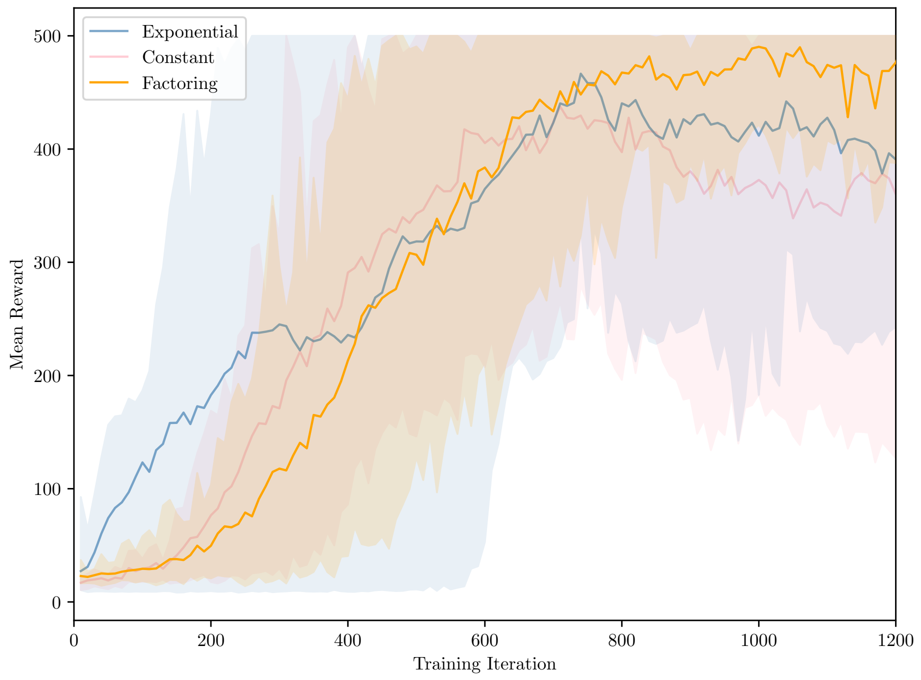

Figure 13: Comparison of multiple lambda initialization and computation methods. Each method represents the 10-episode temporal average of 10 agents.

The benefits of exponential lambda's for significantly faster early training is quickly offset by the cost of high divergence. After a few hundred episodes, the methods seem to converge in learning speed. The real difference starts to become apparent after several hundred episodes, where both exponential and constant methods suffer from trials that either become stuck in a local optimum or collapse and fail to recover. Ultimately, we see that initializing all lambdas as $\boldsymbol{\lambda} = 1$ results in the worst performance in terms of divergence and average. The factoring scheme not only produces the highest average agents but is also the most stable one. A concern was that using the factoring method made the initial $\boldsymbol{\lambda}$ values too small, causing some terms to essentially disappear during training. However, from the inspection of the final $\boldsymbol{\lambda}$ distribution, we see that it is reasonably flat, indicating that we're not dealing with vanishing terms in our model.

1. $L = 3$, $\boldsymbol{\lambda}_{\text{final}} =$ [0.3460143 0.33767825 0.32377955]
2. $L = 6$, $\boldsymbol{\lambda}_{\text{final}} =$ [0.1714123 0.16860187 0.17415948 0.17259052 0.1777627 0.1780858]

Using this factoring method allows us to scale up the number of terms in our model for our term comparison in subsequent experiments more confidently, without introducing lambda-related instabilities.

8.4 Effects of the Number of Terms on the Reduced Model

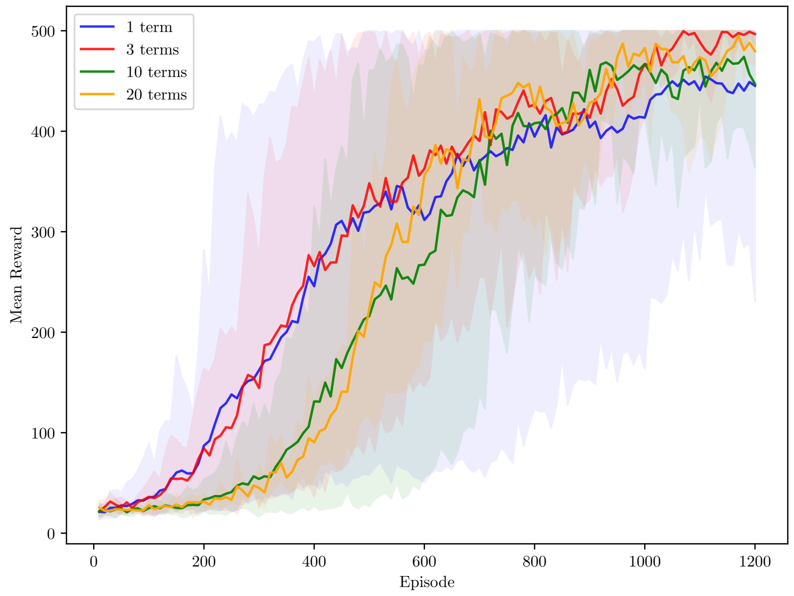

Figure 14: Comparative analysis of the effects of varying term quantities on learning performance. Each trajectory represents the ten-episode temporal average across five distinct agents.

Some apparent observations are that fewer terms lead to quicker early training, but higher divergence. The initial slower training might be related to the larger set of partition process related trainable variables as the number of terms increases. The averages of all the terms seem to converge after a few hundred episodes.

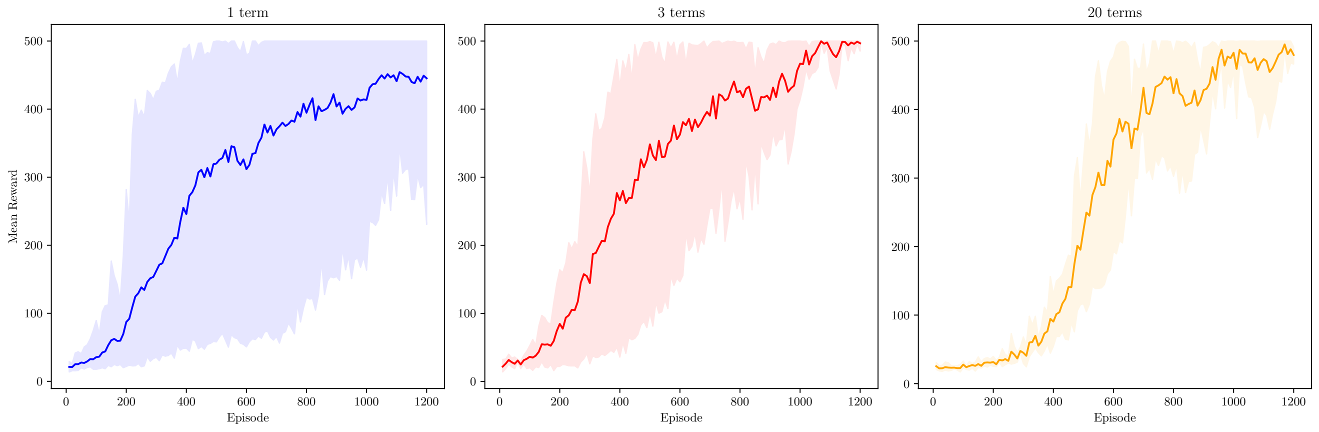

Figure 15: Side-by-side comparison of different term quantities, showing decreasing divergence as the number of terms increases.

Models with a single term are able to solve the cartpole problem, even though we see high variance in these agents. This deviation from the mean progressively decreases when we start to introduce more terms, with some outliers, which could be due to the limited power of our experiment. On average, we observe that all models perform similarly, but when examining their unnormalized outputs, we definitely notice significant differences. From the side-by-side comparison (Appendix: D), we observe that the more terms we introduce, the less oscillating the output of the model tends to become. This effect is very apparent in models with fewer terms but becomes more nuanced in those with more terms. PQCs are known to have a tendency to oscillate in comparison to classical neural networks, which was already observed by [12]. This oscillating nature is a source of instability unique to PQCs. Furthermore, we find that introducing more terms also reduces the mean squared error between the unpartitioned and partitioned model, suggesting that introducing more terms provides a closer approximation to the unpartitioned circuit.

9 Discussion

The results of our experiments show that the reduced partition model can solve CartPole using a variaty of learning rates. There is quite some flexibility for all model-related learning rates with exception to $\boldsymbol{\zeta}_{lr}$ which if set too high will impair learning capabilities. Some speculative reasoning might be that there must be a balance between the rate at which the agent learns CartPole and the rate at which model learns its partition weights. Putting $\boldsymbol{\zeta}$ to high might cause the partition paramaters to overfit on intermediate learning steps. Lower learning rates yield higher long term rewards and more stability while maintaining roughly simliair learning speeds, making the tradeoff quite favorable.

Intuitively, removing lambda from the exponent should make them less sensitive to minor changes, thereby enhancing their stability. This assumption is somewhat supported by the observed results. Both the constant and factoring methods display greater stability early in the training phase. The magnitude difference between them doesn't affect the learning rate since these adjustments are applied to the gradient, not the parameter itself. Agents using the exponential scheme, which become trapped in continuous left or right actions, likely do so because of their larger circuit outputs. As these significantly larger outputs are processed by the softmax layer, they result in skewed action probabilities. Yet, this observation does not completely account for the observed ordering later in training, where the constant and exponential methods clearly underperform. This behavior may relate to the rescaling of the $\boldsymbol{w}$ parameters during training. Recall that these parameters scale the circuit's output before it reaches the softmax layer. Upon analyzing their behavior we observed that in the constant/exponential scheme, $\boldsymbol{w}$ gradually scales down the magnitude of the circuit outputs, while in the factoring scheme, $\boldsymbol{w}$ gently increases them. The latter most likely results in more stable outputs, again, due to the characteristics of the softmax layer, which creates flatter distributions when receiving inputs of reduced magnitude. Slowly increasing $\boldsymbol{w}$ throughout training gives similair effects as decaying-epsilon type methods, which reduce the exploration rate throughout training. Normalizing $\boldsymbol{\lambda}$ on the number of terms is an effective method of preventing any magnitude related instabilities that might occur in subsequent layers.

We have demonstrated that CartPole can be trained successfully using a single term. The slightly reduced learning speed when introducing more terms is probably a consequence of the added complexity of the $\boldsymbol{\zeta}$ and $\boldsymbol{\lambda}$ parameters, suggesting the model needs more time initially to learn the partition gates. When evaluating the impact of introducing more terms on the stability of the agents, several factors come into play. On the surface, one could argue that more terms lead to a more accurate approximation of the unpartitioned model, thereby reducing approximation errors and consequently resulting in a more stable policy. However, certain results hint that this viewpoint may require reconsideration. In a comparative analysis of five random trials between a 20-term and an unpartitioned agent, the latter exhibits very similar, and occasionally slightly better, stability, especially in later training stages (Appendix: F).

This observation could imply that the relationship is more nuanced than simply improving model approximations. For instance, it might be associated with the dampened oscillatory output observed in the partitioned model as compared to the unpartitioned one, which was an observation noticed by comparing unnormalized model outputs. Introducing more terms might give us a more accurate aproximation of the unpartitioned model that is slightly less ossilatory than the full model itself, serendipitously causing more consistent decision-making or better regularization, thus contributing to an observed increase in stability. Nevertheless, this hypothesis needs further exploration and rigorous empirical testing to establish a definitive understanding of the mechanisms at work.

10 Conclusion and further research

To conclude, results have demonstrated that we can efficiently solve CartPole using reduced parametrized quantum circuits, utilizing half the qubits that the algorithm typically demands. Normalizing our models output with respect to the number of evaluated terms results in more stable policies due to its relationship with the softmax layer. Results also suggest that increasing the amount of terms will positively affect training stability and the model's capability to approximate its unpartitioned counterpart. Furthermore, we provide weak evidence suggesting that these approximations might yield more stable policies than the unpartitioned circuit. However, to derive more definitive conclusions, additional experiments should be conducted to add to their statistical power. The insights from this research pave the way for deeper exploration into circuit partitioning in progressively complex hybrid reinforcement learning algorithms and environments. One particularly captivating candidate is a PQC-generated environment. This environment [12] demonstrated separations in performance between PQCs and classical neural networks. It would be compelling to examine the interplay between terms and performance in an environment which is fundamentally more quantum. Another viable research direction is to benchmark agents in higher-dimensional settings with more partitions to gauge how empirical results scale in such contexts.

A Appendix A: Hyperparameter sweep settings

Hyperparameters	**Values**	**Hyperparameters**	**Values**
lr w	0.01	N terms	1, 3, 6
lr λ_{input}	0.01	N layers	5
lr ϕ	0.02, 0.002, 0.0002	Gamma	1
lr ζ	0.02, 0.002, 0.0002	Beta	1
lr λ_{term}	0.02, 0.002, 0.0002	Batch size	10
Environment	CartPole-v1	N Qubits	2
Partitions	2	Max episodes	1500
Trials per agent	5	Lambda method	Fac

Table 2: Grid search style hyperparameter sweep.

B Appendix B: Lambda Initialization Experiment

Hyperparameters	**Values**	**Hyperparameters**	**Values**
lr w	0.01	N terms	3, 6
lr λ_{input}	0.01	N layers	5
lr ϕ	0.002	Gamma	1
lr ζ	0.0002	Beta	1
lr λ_{term}	0.0002	Batch size	10
Environment	CartPole-v1	N Qubits	2
Partitions	2	Max episodes	1200
Trials per agent	5	Lambda method	Exp, Fac, Const

Table 3: Lambda comparison experiment paramaters.

C Appendix C: Term comparison experiment settings

Hyperparameters	**Values**	**Hyperparameters**	**Values**
lr w	0.01	N terms	1, 3, 6, 10, 20
lr λ_{input}	0.01	N layers	5
lr ϕ	0.002	Gamma	1
lr ζ	0.0002	Beta	1
lr λ_{term}	0.0002	Batch size	10
Environment	CartPole-v1	N Qubits	2
Partitions	2	Max episodes	1500
Trials per agent	5	Lambda method	Fac

Table 4: Term comparison experiment paramaters.

D Appendix D: Policy Comparison

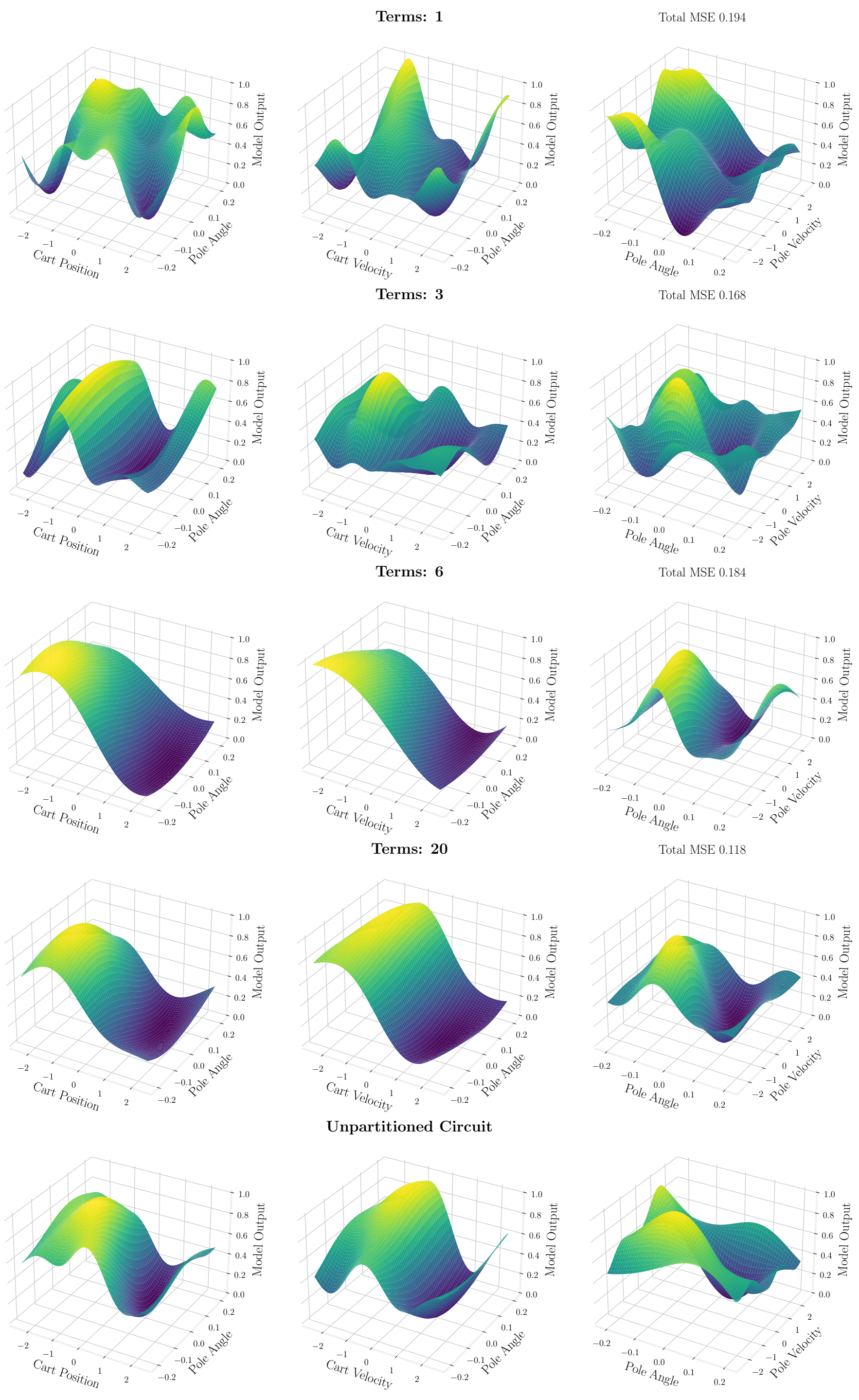

Figure 16: Comparison of all policies with 1, 3, 6, and 20-term partitioned circuits and the unpartitioned circuit. For each set of terms, we show the MinMax-Scaled outputs of the model pre-softmax application. The probability of taking the left action can be obtained by taking the sigmoid of the output. The Mean Squared Error (MSE) of each partitioned circuit is calculated with respect to the outputs of the unpartitioned circuit. Each agent was trained until it reached a maximum average reward, and all agents were trained using the same hyperparameters (if applicable).

E Appendix E: Reshuffled inputs

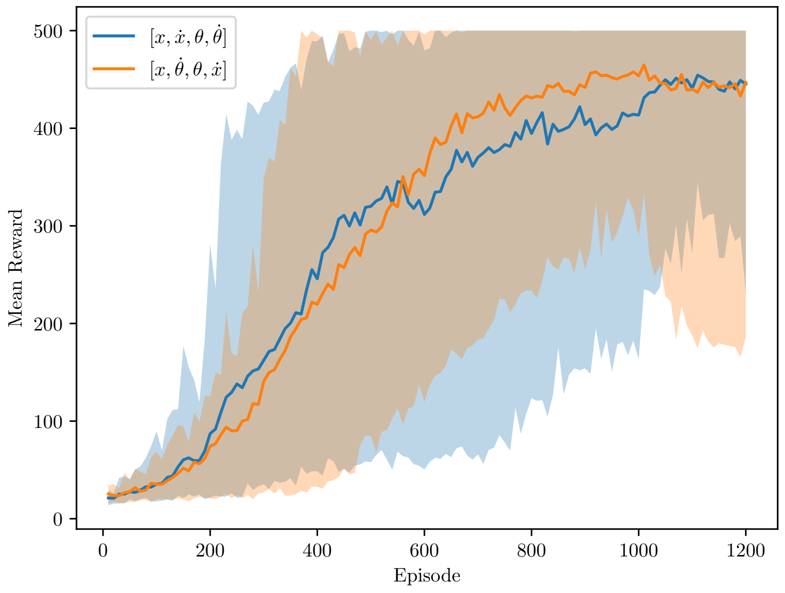

Figure 17: Comparison between different datapoints on each side of the partition. Each permutation is the 10-episode temporal average of 10 agents. Each agent is run on a single term sharing the same hyperparamaters.

F Appendix F: Unpartitioned x Partitioned

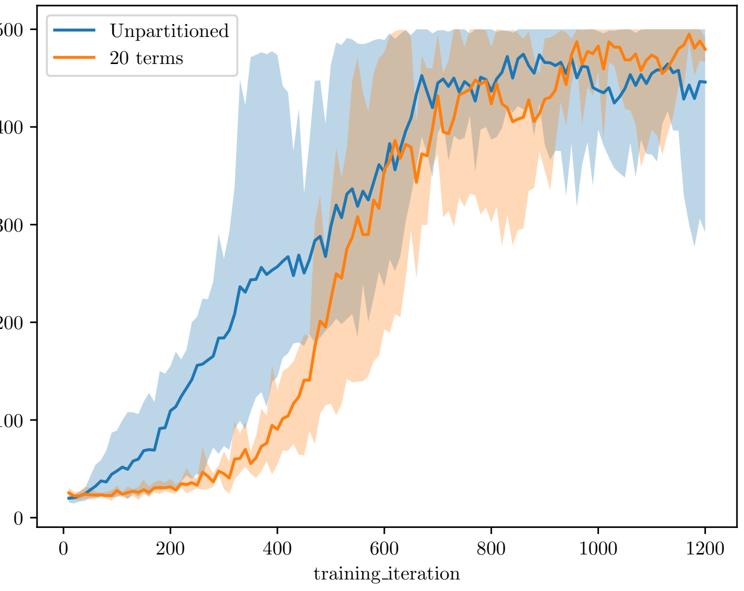

Figure 18: Comparison between 5 random unpartitioned and partitioned circuits.

References

[1] Scott Aaronson. Read the fine print. *Nature Phys.*, 11(4):291–293, 2015.

[2] Scott Aaronson. Introduction to quantum information science lecture notes. *https://www.scottaaronson.com/qclec.pdf*, 2018.

[3] Saikat Basu, Amit Saha, Amlan Chakrabarti, and Susmita Sur-Kolay. I-qer: An intelligent approach towards quantum error reduction. *ACM Transactions on Quantum Computing*, 3(4), jul 2022.

[4] Sergey Bravyi, Graeme Smith, and John A. Smolin. Trading classical and quantum computational resources. *Phys. Rev. X*, 6:021043, Jun 2016.

[5] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[6] Michael Broughton, Guillaume Verdon, Trevor McCourt, Antonio J. Martinez, Jae Hyeon Yoo, Sergei V. Isakov, Philip Massey, Ramin Halavati, Murphy Yuezhen Niu, Alexander Zlokapa, Evan Peters, Owen Lockwood, Andrea Skolik, Sofiene Jerbi, Vedran Dunjko, Martin Leib, Michael Streif, David Von Dollen, Hongxiang Chen, Shuxiang Cao, Roeland Wiersema, Hsin-Yuan Huang, Jarrod R. McClean, Ryan Babbush, Sergio Boixo, Dave Bacon, Alan K. Ho, Hartmut Neven, and Masoud Mohseni. Tensorflow quantum: A software framework for quantum machine learning, 2021.

[7] Samuel Yen-Chi Chen, Chao-Han Huck Yang, Jun Qi, Pin-Yu Chen, Xiaoli Ma, and Hsi-Sheng Goan. Variational quantum circuits for deep reinforcement learning, 2020.

[8] Ronald de Wolf. Quantum computing: Lecture notes, 2023.

[9] Cirq Developers. Cirq, December 2022. See full list of authors on Github: https://github .com/quantumlib/Cirq/graphs/contributors.

[10] Vedran Dunjko, Jacob M. Taylor, and Hans J. Briegel. Quantum-enhanced machine learning. *Physical Review Letters*, 117(13), sep 2016.

[11] IBM. *https://newsroom.ibm.com/2022-11-09-IBM-Unveils-400-Qubit-Plus-Quantum-Processor-and-Next-Generation-IBM-Quantum-System-Two*, 2022.

[12] Sofiene Jerbi, Casper Gyurik, Simon C. Marshall, Hans J. Briegel, and Vedran Dunjko. Parametrized quantum policies for reinforcement learning, 2021.

[13] Richard Liaw, Eric Liang, Robert Nishihara, Philipp Moritz, Joseph E Gonzalez, and Ion Stoica. Tune: A research platform for distributed model selection and training. *arXiv preprint arXiv:1807.05118*, 2018.

[14] Daniel Mankowitz, Andrea Michi, Anton Zhernov, Marco Gelmi, Marco Selvi, Cosmin Paduraru, Edouard Leurent, Shariq Iqbal, Jean-Baptiste Lespiau, Alex Ahern, Thomas Köppe, Kevin Millikin, Stephen Gaffney, Sophie Elster, Jackson Broshear, Chris Gamble, Kieran Milan, Robert Tung, Minjae Hwang, and David Silver. Faster sorting algorithms discovered using deep reinforcement learning. *Nature*, 618:257–263, 06 2023.

[15] Simon C. Marshall, Casper Gyurik, and Vedran Dunjko. High dimensional quantum machine learning with small quantum computers, 2023.

[16] Michael A Nielsen and Isaac L Chuang. *Quantum computation and quantum information.* Cambridge university press, 2010.

[17] Adriá n Pérez-Salinas, Alba Cervera-Lierta, Elies Gil-Fuster, and José I. Latorre. Data re-uploading for a universal quantum classifier. *Quantum*, 4:226, feb 2020.

[18] Christophe Piveteau and David Sutter. Circuit knitting with classical communication, 2023.

[19] Lex Fridman Podcast. Scott aaronson quantum computing lex fridman podcast 72. *"youtubelink"*, 2020.

[20] John Preskill. Quantum Computing in the NISQ era and beyond. *Quantum*, 2:79, August 2018.

[21] Maria Schuld, Ville Bergholm, Christian Gogolin, Josh Izaac, and Nathan Killoran. Evaluating analytic gradients on quantum hardware. *Physical Review A*, 99(3), mar 2019.

[22] Maria Schuld and Nathan Killoran. Is quantum advantage the right goal for quantum machine learning? *PRX Quantum*, 3(3), jul 2022.

[23] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy P. Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *CoRR*, abs/1712.01815, 2017.

[24] David Silver, Satinder Singh, Doina Precup, and Richard S. Sutton. Reward is enough. *Artificial Intelligence*, 299:103535, 2021.

[25] Andrea Skolik, Sofiene Jerbi, and Vedran Dunjko. Quantum agents in the gym: a variational quantum algorithm for deep q-learning. *Quantum*, 6:720, may 2022.

[26] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction.* MIT press, 2018.

[27] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3):229–256, 1992.

[28] Shaojun Wu, Shan Jin, Dingding Wen, Donghong Han, and Xiaoting Wang. Quantum reinforcement learning in continuous action space, 2023.