IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Digital Object Identifier

Quantum Deep Reinforcement Learning for Robot Navigation Tasks

HANS HOHENFELD[1], DIRK HEIMANN[1], FELIX WIEBE[2], and FRANK KIRCHNER[1,2]
[1]Robotics Research Group, University of Bremen, Robert-Hooke-Straße 1, 28359 Bremen, Germany (e-mail: {hans.hohenfeld, dirk.heimann, frank.kirchner}@uni-bremen.de)
[2]Robotics Innovation Center (RIC), German Research Center for Artificial Intelligence (DFKI), Robert-Hooke-Straße 1, 28359 Bremen, Germany (e-mail: {felix.wiebe, frank.kirchner}@dfki.de)

Corresponding author: Hans Hohenfeld (e-mail: hans.hohenfeld@uni-bremen.de).

This work was funded by the German Federal Ministry of Economic Affairs and Climate Action (BMWK) and German Aerospace Center e.V. (DLR e.V.) through the project QINROS under project numbers 50RA2033 (DFKI) and 50RA2032 (University of Bremen).

ABSTRACT We utilize hybrid quantum deep reinforcement learning to learn navigation tasks for a simple, wheeled robot in simulated environments of increasing complexity. For this, we train parameterized quantum circuits (PQCs) with two different encoding strategies in a hybrid quantum-classical setup as well as a classical neural network baseline with the double deep Q network (DDQN) reinforcement learning algorithm. Quantum deep reinforcement learning (QDRL) has previously been studied in several relatively simple benchmark environments, mainly from the OpenAI gym suite. However, scaling behavior and applicability of QDRL to more demanding tasks closer to real-world problems e. g., from the robotics domain, have not been studied previously. Here, we show that quantum circuits in hybrid quantum-classic reinforcement learning setups are capable of learning optimal policies in multiple robotic navigation scenarios with notably fewer trainable parameters compared to a classical baseline. Across a large number of experimental configurations, we find that the employed quantum circuits outperform the classical neural network baselines when equating for the number of trainable parameters. Yet, the classical neural network consistently showed better results concerning training times and stability, with at least one order of magnitude of trainable parameters more than the best-performing quantum circuits. However, validating the robustness of the learning methods in a large and dynamic environment, we find that the classical baseline produces more stable and better performing policies overall. For the two encoding schemes, we observed better results for consecutively encoding the classical state vector on each qubit compared to encoding each component on a separate qubit. Our findings demonstrate that current hybrid quantum machine-learning approaches can be scaled to simple robotic problems while yielding sufficient results, at least in an idealized simulated setting, but there are yet open questions regarding the application to considerably more demanding tasks. We anticipate that our work will contribute to introducing quantum machine learning in general and quantum deep reinforcement learning in particular to more demanding problem domains and emphasize the importance of encoding techniques for classic data in hybrid quantum-classical settings.

INDEX TERMS Reinforcement learning, Autonomous agents, Robotics, Quantum machine learning, Quantum computing

I. INTRODUCTION

Robotics research and applications pose various algorithmic challenges, ranging from large-scale optimization, processing of high-dimensional sensory input, planning the execution of complex tasks in demanding environments, and learning of autonomous, adaptable behaviors. On the latter, deep reinforcement learning is used to produce impressive results in tasks such as learning complex manipulation behaviors [1], reaching, tracking and, navigation [2], manipulation based on visual input [3] as well as dexterous hand movements [4] among many others. It constitutes a central role on the path toward autonomous and life-long learning robots.

Quantum computing algorithms [5] present a novel way of approaching algorithmic problems and offer theoretical advantages over classical algorithms for specific problems like factoring numbers [6], unstructured search [7], and solving systems of linear equations [8]. With more development and further resources, quantum computing and, in particular quantum machine learning [9] may contribute to the development of artificial intelligence in general and the learning of

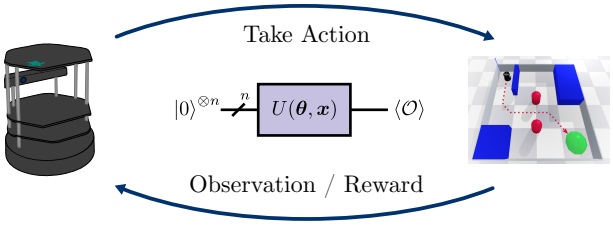


FIGURE 1. Main contribution: We use parameterized quantum circuits (PQCs) as function approximators in the DDQN Deep Reinforcement Learning algorithm to learn optimal policies for a simulated Turtlebot robotic system in several simulated navigation tasks.

autonomous behaviors for robots in particular [10].

The idea of robots controlled by quantum computers, interacting with an environment on the scale of individual quantum states has arguably first been hypothesized and described by quantum computing pioneer Paul Benioff in the late 1990s and early 2000s [11]–[14]. While those envisioned *Quantum Robots* are very different from typical mechanical robotic systems as they can be found in various practical applications today, the idea of a mobile system utilizing quantum computing hardware remains intriguing.

Quantum computing technology has not yet reached the state of mobile, embedded, and potentially battery-powered quantum hardware but has made remarkable progress over the last two decades. Research institutions and companies are building quantum computers with increasing capabilities, and while current Noisy Intermediate-Scale Quantum Computers (NISQ) are limited in the number of qubits, coherence times, and fidelity of operation [15], they already enable exploring solutions for various problems [16].

One potential application for NISQ devices is the hybrid training of parameterized quantum circuits (PQCs) as machine learning models [17]. While this technique has been studied in various domains of machine learning [18], deep reinforcement learning has only recently attracted substantial research interest in this context. Existing works (see Sec. II-D) demonstrate the applicability of hybrid quantum-classical approaches for reinforcement learning tasks, with performances similar to classical algorithms while learning notably more compact models. However, their scope is currently limited to relatively simple benchmark environments, mainly from the OpenAI gym suite [19].

Or main contributions, illustrated in Fig. 1, are as follows. We demonstrate the feasibility of quantum deep reinforcement learning in three simulated robotic navigation tasks of increasing size and difficulty. Thereby, we extend the scope of previously introduced methods to substantially more complex tasks in the robotic domain, as we show by comparative experiments with typical benchmark environments. Furthermore, we compare to different encoding strategies for the classical state of the robot into a quantum circuit and also analyze the scaling behavior of the quantum circuits relative to a classical baseline. To validate the robustness of the presented methods, we additionally demonstrate their application in a substantially larger, more demanding and dynamic environment. In comparison to previous works in the field of quantum deep reinforcement learning, we thereby increase the complexity of considered learning tasks and furthermore provide a systematic evaluation of the scaling behaviour of quantum models in this context. Finally, we discuss various challenges and limitations of quantum deep reinforcement learning in a robotic context, as well as potential areas of research for quantum machine learning to contribute to the future advancements in autonomous robotics.

The rest of this paper is outlined as follows: In Sec. II, we provide an overview of previous works regarding deep reinforcement learning with PQCs. Subsequently, we outline the quantum deep reinforcement learning framework underlying this work in Sec. III. Afterwards, the learning setup with regard to the simulated environments and learning methods is documented in Sec. IV. We present the training results of the suggested methods compared to a classical baseline in Sec. V before summarizing our main findings and discussing their implications and limitations in Sec. VI. Finally, we give an outlook toward potential future research directions in Sec. VII.

II. RELATED WORK

Introducing quantum algorithmic techniques and quantum mechanical effects into reinforcement learning (RL) methods is an active and growing field of research. Meyer *et al.* [20] give an overview over various proposed methods and applications in this area. In the following, we highlight important methods and results from this line of research.

A. QUANTUM RL AND QUANTUM INSPIRED RL

Quantum mechanics and quantum computing were introduced reinforcement learning by Dong *et al.* [21], who proposed Quantum Reinforcement Learning (QRL). In the QRL algorithm, the classical states and actions of the agent are expressed in the orthonormal eigenbasis of a Hermitian observable. Actions are chosen by measuring in that basis from a superposition state, where the amplitudes of that superposition state are modified during learning utilizing amplitude amplification, the essential building block of Grover's algorithm [7]. The authors evaluate the QRL algorithm in a discrete maze world, comparing it to the tabular TD(0) RL algorithm [22], achieving convincing performance. Quantum-inspired Reinforcement Learning (QiRL) [23] is a classical RL algorithm that builds on the ideas of QRL, using a quantum-inspired probabilistic sampling technique to address the exploration vs. exploitation [22] problem in RL and a classical technique inspired by amplitude amplification to control the sampling probabilities. The algorithm is demonstrated on a simulated grid world and real-world robot navigation task with a wheeled MT-R robot. A variant of QiRL with flexible rotation angles in the amplitude amplification step is proposed in Ref. [24], which shows better performance on a UAV navigation problem compared to tabular Q-learning [25] with two different exploration strategies. Hu *et*

al. [26] apply QRL to the `MountainCar` and `CartPole` problems from the OpenAI Gym [19] suite, focusing on the exploration vs. exploitation problem, finding better overall learning performance compared to the classical Q-learning algorithm with an ϵ-greedy policy. Quantum-inspired Experience Replay (QER) [27] is an extension of the concepts of QiRL to the representation of experiences and sampling from the replay buffer in Deep Reinforcement Learning (DRL), which the authors evaluate in several `Atari 2600` game environments [19] and compare to baseline experience replay and prioritized experience replay with several variants of the DQN [26] algorithm.

This line of work with QRL and QiRL emphasizes the expression of states and actions in RL problems in quantum states, efficiently updating measurement probabilities leveraging amplitude amplification, and expressing the same concepts in a classical learning setup. QER extends these ideas to experience replay in DRL. Our contribution is conceptually different, as we focus on substituting classical neural networks in DRL with parameterized quantum circuits while keeping the learning algorithm and representation of all aspects of the learning task unchanged.

B. QUANTUM ENVIRONMENTS

Dunjko *et al.* [28] propose a quantum-enhanced framework that, in principle, covers supervised, unsupervised, and reinforcement learning but, in its formulation, is closest to the latter. In this framework, the agent and the environment exchange actions and percepts by applying completely positive trace-preserving (CPTP) maps to a shared quantum register and their local quantum memory. The authors analyze the conditions under which an agent in this framework can outperform its classical counterpart. They extend this work to a meta-learning scenario in Ref. [29] demonstrating further improvements.

Saggio *et al.* [30] suggest a reinforcement learning setting in which the agent and environment exchange information on a classical and a quantum channel in an alternating way and show how, in such a setting, an agent performs better than with strictly classical information exchange. The authors validate the concept by performing experiments on a photonic quantum processor. Theoretical performance analysis of this framework is provided in Ref. [31], where the authors find a quadratic learning speed-up, which still holds under hardware noise and limited coherence times. Dalla *et al.* [32] successfully demonstrate a classical deep reinforcement learner in a quantized maze environment based on quantum walks, considering potentially noisy dynamics.

Quantum-enhanced learning frameworks, as considered in these works, require some form of quantum information based interaction between agent and environment and are, in that aspect, different from our contribution. We consider an environment from the robotic domain, where agent-environment interaction is strictly classical.

C. PROJECTIVE SIMULATION

Projective simulation [33] is an extension of the RL learning framework by an episodic and compositional memory, which allows the agent to predict potential future events using random walks on that memory. In Ref. [34], the authors propose an extension of this learning method using quantum walk on quantum memory instead to achieve a quadratic speed-up, which was later demonstrated in a proof-of-principle experiment on an ion-trap based quantum system by Sriarunothai *et al.* [35].

The deep reinforcement learning algorithm we use in our work does not utilize any form of episodic memory, hence the suggested techniques in this line of research are not immediately applicable.

D. QUANTUM DEEP REINFORCEMENT LEARNING

In quantum deep reinforcement learning (QDRL), the line of research from which our contribution originates, one or multiple classical neural networks are replaced or extended by parameterized quantum circuits. In contrast, agent-environment interaction and as the learning procedure are kept classical. We give a detailed account of the underlying theory in Sec. III. The focus in this relatively new field so far has mostly been on showing the feasibility of the methods, understanding their capabilities and limitations, as well as finding quantum-classical separations in learning tasks.

In several works the Q-function approximation in the DQN algorithm is implemented by a PQC. Chen *et al.* [36] use basis encoding [37] followed by CNOT entanglements and parameterized Pauli rotations without a data re-uploading structure for the `FrozenLake` [19] and a `CognitiveRadio` task [38] with discrete state and action spaces. Lockwood *et al.* [39] use a different encoding technique and combine the parameterized circuit with quantum pooling operations [40] and classical neural network layers without data re-upload. This setup is able to learn a `Blackjack` environment but do not successful learn `Cartpole-v0` [19]. In Ref. [41] the circuit layout and encoding scheme is similar to the one that we employ in this work. The architecture also includes data re-uploading and enables learning on `FrozenLake` and `Cartpole-v0`.

In addition, PQCs have also been used in the policy gradient methods REINFORCE [22]. In Ref. [42] the PQC architecture also includes data re-upload scheme. Included in the REINFORCE algorithm, the setup is able to solve `Cartpole-v1`, `Mountaincar-v0` and `Acrobot-v1`. Additionally, the authors demonstrate experimentally and formally that hybrid quantum deep reinforcement learning can solve environments based on the discrete logarithm problem [43] which are intractable for classical learning methods. A variant of the REINFORCE algorithm is used in Ref. [44] to optimize PQCs, which replace the classical attention head layers originally introduced in Ref. [45], to solve the vehicle routing problem and achieve similar results as the classical counterpart.

Furthermore, actor critic methods such as proximal policy optimization (PPO) [46] and soft actor-critic (SAC) [47] have also been adapted with PQCs. In Ref. [48] the PPO algorithm is augmented with PQCs by exchanging the actor approximation network. The PQC has no data re-uploading scheme and is trained on `Cartpole-v0` without completely solving it. In Ref. [49] unentangled PQCs with a fully connected classical layer as post processing unit replace the classical estimator for the actor and critic. This setup solves OpenAI Gym environments `Cartpole-v1`, `Acrobot-v1` and `LunarLander-v2`.

Nagy *et al.* [50] simulate a hybrid quantum version of PPO on a photonic processor which demonstrates that PQC equivalences on photonic quantum computers can be used for reinforcement learning as well. In Ref. [51] the author demonstrates that the critic network in SAC can be exchanged with a PQC followed by a classical neural network and still solve the `Pendulum-v0` problem from OpenAI gym with continuous state and action spaces.

Several works introduce parameterized quantum circuits into a hybrid quantum-classical learning setup, without strictly falling into the category of deep reinforcement learning. Cherrat *et al.* [52] implement a quantum version of policy iteration to solve `FrozenLake` and the `InvertedPendulum` environment. Franken *et al.* [53] implement a gradient-free method based on evolutionary methods to optimize a PQC that receives input data encoded by a tensor network. This setup is able to solve `MiniGrid` worlds [54] with discrete state space.

Or contribution extends upon these previous works in the following way:

- We extend the scope of QDRL to considerably more complex learning tasks from the robotic domain. We establish that increase in complexity by comparative experiments (see Appendix A).
- We systematically evaluate the scaling behaviour of parameterized quantum circuits in QDRl across task complexity as well as model size, which has previously not been done.
- We compare different encoding strategies suggested in the literature for re-uploading circuits with regards to their performance in a QDRL scenario.

Thereby we extend the understanding of the feasibility of QDRL from very simple benchmark environments towards more realistic application scenarios from the robotics domain and contribute to the understanding of the model scaling behaviour in this context.

III. QUANTUM DEEP REINFORCEMENT LEARNING

A. DOUBLE DEEP Q-NETWORKS

For all our experiments, we used the Double Deep Q-Network (DDQN) [55] algorithm as it performed slightly better on average compared to e. g., the basic Deep Q-Network algorithm (DQN) [56]. DDQN is a model-free, off-policy deep RL algorithm that uses a neural network to approximate the Q-function from the basic Q-learning algorithm [25].

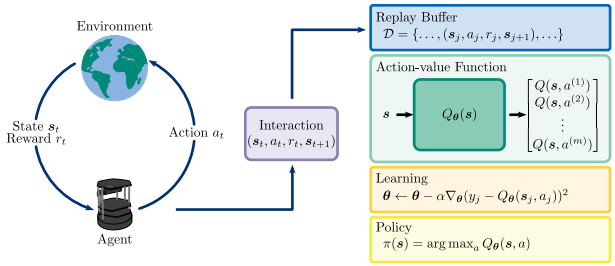


FIGURE 2. Reinforcement Learning setup (left) and main parts of the DDQN algorithm (right). An agent interacts with an environment by performing action a_t after observing a state of the environment s_t, causing a transition to state s_{t+1} and receiving a reward r_t. For the DDQN algorithm, these interactions are stored in a replay buffer, from which regularly random mini-batches are sampled to train an artificial neural network Q_θ approximating the action-value function.

RL is used to solve Markov Decision Processes (MDPs), that is, discrete-time, stochastic processes (S, A, T, r, p_0) with

- S: The state space, a set of all possible states of an environment
- A: The action space, a set of all possible actions for an agent
- $T : S \times A \times S \to [0, 1]$: The possibly stochastic transition function with $T(s, a, s') = p(s'|s, a)$ being the probability of transitioning to state s' after taking action a in state s.
- $r : S \times A \times S \to \mathbb{R}$: A reward function with $r(s, a, s')$ denoting a numeric reward for taking action a in state s and transitioning to state s' and
- p_0: A probability for each state to be a starting state of the MDP.

The general scheme of interaction for an agent in an environment governed by an MDP is illustrated on the left side of Fig. 2. At each time step t, the agent observes a state s_t of the environment, takes an action a_t, which causes a transition to state s_{t+1} and the agent to receive a reward r_t. The agent's action selection is governed by a policy $\pi : S \to A$ and the goal is to maximize the total cumulative reward R for a possibly infinite time horizon, given by

$$R = \sum_{t=0}^{\infty} \gamma^t r_t \tag{1}$$

with $\gamma \in [0, 1]$, called discount factor, encoding the preference for immediate over long-term rewards.

In the DDQN algorithm, this is achieved by learning an optimal action-value function $Q : S \times A \to \mathbb{R}$. The action-value function $Q(s, a)$ expresses the expected total cumulative reward for taking action a in state s

$$Q(s, a) := \langle R \rangle_{s,a,\pi}, \tag{2}$$

and the greedy policy can be expressed in terms of $Q(s, a)$ by

$$\pi(s) := \operatorname*{argmax}_a Q(s, a). \tag{3}$$

The action-value function, also referred to as Q-function, is approximated by an artificial neural network $Q_{\boldsymbol{\theta}}$ with parameters $\boldsymbol{\theta}$. The neural network takes a state $s \in S$ as input and computes $Q(s, a^{(i)})$ for all $a^{(i)} \in A$ as output. During learning, an ϵ-greedy policy is employed by the agent, that is at each time step t with probability $\epsilon \in [0, 1]$, the agent takes a random action from A to further explore the environment and with probability $1 - \epsilon$, it follows the greedy policy (3) to exploit its current knowledge. At the beginning of training, ϵ is commonly chosen with a value close to 1 and gradually reduced toward 0 as learning progresses.

Interactions $(s_t, a_t, r_{t+1}, s_{t+1})$ are stored in a replay buffer [57] from which at a predefined interval e. g., at each time step, a mini-batch is sampled to update $Q_{\boldsymbol{\theta}}$ with stochastic gradient descent, minimizing the loss

$$\mathcal{L}(\boldsymbol{\theta}) = (y_t - Q_\theta(s_t, a_t))^2, \tag{4}$$

with y_t given by

$$y_t = r_{t+1} + \gamma Q_{\boldsymbol{\theta}'}(s_{t+1}, \underset{a'}{\operatorname{argmax}}\, Q_{\boldsymbol{\theta}}(s_{t+1}, a')). \tag{5}$$

The target network $Q_{\boldsymbol{\theta}'}$ is used to stabilize the training process [56]. It has the identical structure as $Q_{\boldsymbol{\theta}}$ and is periodically updated with $\boldsymbol{\theta}' \leftarrow \boldsymbol{\theta}$.

B. QUANTUM COMPUTING

We give a short introduction to the common notation of quantum computing and refer the interested reader to [58] for a comprehensive explanation of basic and advanced concepts of this topic. The fundamental objects in quantum computing are qubits, analogous to bits in classical computing. Unlike classical bits, which can be in one of the two states, 0 and 1, a qubit can be in a state, which is a linear combination of those states. Using the bra-ket notation, a qubit state $|\Psi\rangle$ can be written as

$$|\Psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle\,, \text{with } \alpha, \beta \in \mathbb{C}, |\alpha|^2 + |\beta|^2 = 1, \tag{6}$$

where $|0\rangle$ and $|1\rangle$ are basis states of the underlying single-qubit Hilbert Space. During the probabilistic measurement process, the qubit will collapse to one of the two basis states, and $|\alpha|^2$ and $|\beta|^2$ can be interpreted as the probabilities for the respective basis states. Before the measurement, the state can be modified by applying quantum gates U

$$|\Psi'\rangle = U\,|\Psi\rangle\,, \tag{7}$$

formally described by unitary operators U. This formulation can be extended to multi-qubit systems by preparing an n-qubit quantum register. For quantum computers, this is commonly initialized in its computational basis state $|0\rangle^{\otimes n}$.

C. PARAMETERIZED QUANTUM CIRCUITS FOR DEEP REINFORCEMENT LEARNING

Variational quantum algorithms are a promising method to implement algorithms on current and near-term quantum computers as they are well suited for systems with a relatively small number of qubits, noisy operations, and limited coherence times [59]. Their basic principle of operation is the combination of a parameterized quantum circuit whose parameters are adjusted by a classical optimizer toward the desired outcome while evaluating the quantum circuit with adjusted parameters at each optimization step [60]. First introduced in the context of variational quantum eigensolvers [61], they became a major research area in quantum machine learning [59].

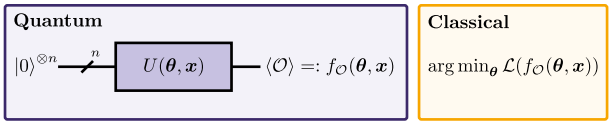


FIGURE 3. Basic principle of a parameterized quantum circuit as function approximator. A unitary $U(\theta, x)$, which may be composed of any number of quantum gates, is applied to an n qubit register initialized in its basis state. The unitary is parameterized by trainable parameters θ and input data x. Thereby, the expectation value of an observable $\langle\mathcal{O}\rangle$ can be defined as a parameterized function $f_{\mathcal{O}}(\theta, x)$. The parameters θ are optimized toward a desired outcome, by minimizing a task specific loss $\mathcal{L}$ using a classical optimization technique e. g., gradient descent.

A parameterized quantum circuit (PQC) is a series of unitary quantum gates $U(\boldsymbol{\theta}, x)$, which is applied to the computational basis of n qubits $|0\rangle^{\otimes n}$. These gates are parameterized by variational parameters $\boldsymbol{\theta}$ and classical input data x. Fig. 3 shows this general ansatz for a machine learning application.

The PQC's quantum state $|U(\boldsymbol{\theta}, x)\rangle = U(\boldsymbol{\theta}, x)\,|0\rangle^{\otimes n}$ is computed and measured for many repeated iterations to gather sufficient statistics for the expectation value

$$\langle\mathcal{O}\rangle = \langle U(x, \boldsymbol{\theta})|\ \mathcal{O}\ |U(\boldsymbol{\theta}, x)\rangle \tag{8}$$

for an observable $\mathcal{O}$. $\langle U(x, \boldsymbol{\theta})|$ denotes the conjugate transpose of $|U(\boldsymbol{\theta}, x)\rangle$. The measured expectation value of the quantum computation can be interpreted as the computation of a parameterized function $f_{\mathcal{O}}(\boldsymbol{\theta}, x)$ depending on the observable, circuit parameters, and input.

The parameters $\boldsymbol{\theta}$ are tuned with an appropriate method to fit a target function. E.g., in the domain of supervised machine learning, a loss function $\mathcal{L}$ can be minimized by performing gradient descent. Several analytic and numeric methods enable calculating gradients of quantum circuits with respect to their parameters, such as the finite difference methid [62], the parameter shift rule [63] and adjoint differentiation [64].

Various encoding methods for quantum machine learning tasks have been suggested [37], but recent results on the expressiveness of quantum circuits emphasize the advantages of repeated encodings, also referred to as data re-upload [65], [66]. Such an ansatz enables the circuit to compute functions of the form

$$f(\boldsymbol{\theta}, x) = \sum_{\omega \in \Omega} c_\omega(\boldsymbol{\theta}) e^{i\omega x}, \tag{9}$$

which is a partial Fourier series with frequency spectrum Ω depending on the data encoding and coefficients $c_\omega(\boldsymbol{\theta})$ determined by trained variational parameters $\boldsymbol{\theta}$ and the entanglement gates [66].

PQCs with data re-upload have L layers, which consist of data encoding unitaries $U_{\text{in}}(x_l)$ followed by parameterized

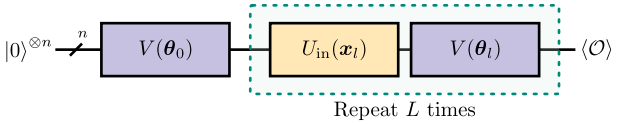


FIGURE 4. Data re-upload in parameterized quantum circuits: After an initial parameterized unitary $V(\theta_0)$, L layers of data encoding unitaries $U_{\text{in}}(x_l)$ and parameterized unitaries $V(\theta_l)$ are applied to the n qubit quantum register.

unitaries $V(\boldsymbol{\theta}_l)$ in each layer l. As introduced in [66], the circuit starts with parameterized unitaries $V(\boldsymbol{\theta}_0)$ applied on the n-qubit register $|0\rangle^{\otimes n}$ followed by a sequential implementation of the layers. Fig. 4 depicts the circuit layout for such an ansatz.

We consider two different re-upload strategies resulting in two different implementations of $U_{\text{in}}(x_l)$. In the first case, we follow [41] by rescaling each continuous classical feature s_i of the state s with trainable parameters ξ_{li} for each layer l using the function $x_{li} = \arctan(\xi_{li} s_i) \in [-\pi, \pi]$. The index sets of the trainable input parameters are given by $i \in \{1, \ldots, n_s\}$ and $l \in \{1, \ldots, L\}$ resulting in Ln_s parameters. In the following, x denotes the set of all encoded and rescaled input data and x_l a subset of all encoded, and rescaled input data for layer l. In this encoding style, each state feature s_i is encoded on one qubit:

$$s \mapsto U_{\text{in},1}(x_l(s)) = \bigotimes_{q=1}^{n} U_{\text{in},1}^{(q)}(x_{lq}), \tag{10}$$

where $U_{\text{in},1}^{(q)}$ is one of the Pauli rotations R_x, R_y, R_z with rotation angle x_{lq} depending on state feature s_q acting on qubit q. For this type of encoding the number of qubits n has to be equal to the number of input features n_s.

In the second case, we encode, in line with [65], three features of the rescaled state s in a universal, single qubit gate $U_{\text{in},3}^{(q)}$ composed of three parameterized Pauli rotation gates. Any combination of rotation gates capable of representing a general single qubit rotation, e. g. $R_x R_y R_x$, suffices for $U_{\text{in},3}^{(q)}$. The state features s_i are encoded as the rotation angles. Therefore, the rescaling is done by using different trainable variables ξ_{li}^q for each qubit $q \in \{1, \ldots, n\}$, which formally can be written as: $x_{li}^q = \arctan(\xi_{li}^q s_i) \in [-\pi, \pi]$. This encoding uses nLn_s trainable input parameters. Similar to the first encoding, all trainable parameters used for layer l are denoted by x_l. A state s is encoded as:

$$s \mapsto U_{\text{in},3}(x_l(s)) = \bigotimes_{q=1}^{n} U_{\text{in},3}^{(q)}(x_{l1}^q, x_{l2}^q, x_{l3}^q). \tag{11}$$

For a state space with more than three features, $U_{\text{in},3}$ is repeated until all features are encoded. Each $U_{\text{in},3}$ then encodes a subset of three features, potentially padding the state space with features set to zero to make its dimensionality divisible by three [65]. We perform experiments with both encoding styles, $U_{\text{in},1}(x)$ and $U_{\text{in},3}(x)$, and unify the notation by referring to both with $U_{\text{in}}(x)$.

The parameterized part of the PQC, $V(\boldsymbol{\theta})$, consists of two parts. One part includes universal, single qubit gates:

$$U_{\text{par}}(\boldsymbol{\theta}_l) = \bigotimes_{q=1}^{n} U_{\text{par}}^{(q)}(\theta_{l1}^q, \theta_{l2}^q, \theta_{l3}^q), \tag{12}$$

which can be implemented by any general, parameterized rotation with three Pauli-rotation gates contributing $3nL$ trainable circuit parameters. The second part contains fixed entangling gates U_{ent} to create entanglement by acting on all n qubits. We choose controlled Z gates on all neighboring pairs of qubits and between the last and the first qubit.

Combining all segments, the PQC ansatz with data re-upload is constructed by applying the parameterized part $V(\boldsymbol{\theta}_0)$ to the initial register, followed by a layer of data encoding $U_{\text{in}}(x_l)$ and another parameterized part $V(\boldsymbol{\theta}_l)$, which are repeated L times. The entire circuit is given by:

$$U(\boldsymbol{\theta}, x) = \prod_{l=1}^{L} \Big(U_{\text{ent}} U_{\text{par}}(\boldsymbol{\theta}_l) U_{\text{in}}(x_l) \Big) U_{\text{ent}} U_{\text{par}}(\boldsymbol{\theta}_0). \tag{13}$$

This operator is applied to the initial state $|0\rangle^{\otimes n}$ leading to the final state $|U(\boldsymbol{\theta}, x)\rangle$, and the expectation value of an observable $\mathcal{O}$:

$$f_{\mathcal{O}}(\boldsymbol{\theta}, s) := \langle\mathcal{O}\rangle_{\boldsymbol{\theta},s} = \big\langle U\big(x(s), \boldsymbol{\theta}\big)\big|\ \mathcal{O}\ \big|U\big(\boldsymbol{\theta}, x(s)\big)\big\rangle. \tag{14}$$

As observables, we choose Pauli-Z gates $\sigma_z^{(1)} \otimes \ldots \otimes \sigma_z^{(n)}$, each acting on another qubit to obtain n different output values. The output values can either be directly interpreted as values for $Q(s, a)$ in the reinforcement learning scenario or combined, scaled, or further post-processed by any classical means including additional classical neural network layers.

Let a_j be one action of the action space $A = \{a_0, \ldots a_{n_a}\}$ with $n_a \leq n_s$. If $n_a < n_s$, the PQC output values can either be combined, e. g., by multiplying some of them [41], to reduce the number of output values to the number of possible actions n_a, or the first n_a qubits are measured. Four our comparative experiments with the `Cartpole-v0` environment we use the former strategy, for the dynamic robot navigation environment the latter.

In our learning scenario, the Q-values range exceeds the interval $[-1, 1]$ and thus needs additional post-processing. The authors of [41] suggest rescaling each output value with an additional trainable parameter ω_j:

$$Q(s, a_j) = \langle\sigma_z^{(j)}\rangle_{\boldsymbol{\theta},s} \cdot w_j, \tag{15}$$

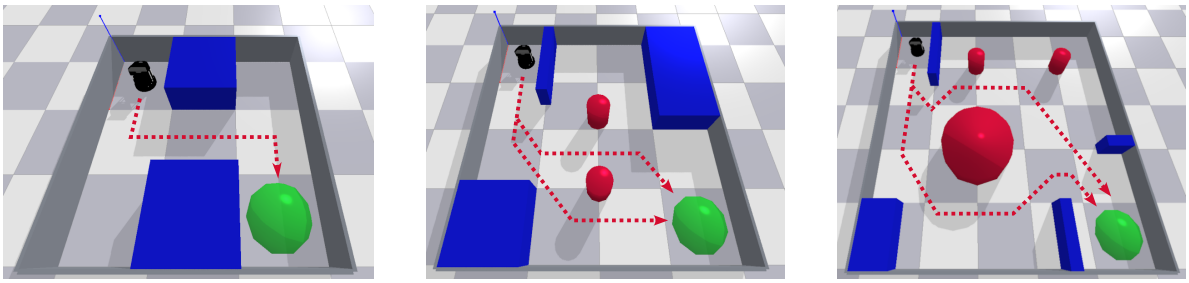
which adds n_a trainable output variables to the model.

IV. METHOD

A. ENVIRONMENTS

In our experiments, we use four environments based on a simulated Turtlebot 2 robot[1]. We chose this robotic system as it enables relatively simple yet realistic navigation tasks while being a readily available and extensible system we can build upon in future work. The robot is controlled via two

[1] https://www.turtlebot.com/turtlebot2/



(a) 3×3 Environment (b) 4×4 Environment (c) 5×5 Environment

FIGURE 5. The three simulated static navigation environments for the Turtlebot 2 robot. In each, the robot has to navigate from its starting position in the upper left corner to the position marked with a green circle in the lower right while avoiding collisions with the enclosing walls and any obstacles. With the configured control scheme, this takes about 20 steps in the 3×3 environment (left), 30 in the 4×4 (center), and 45 steps in the 5×5 environment (right) for a (near) optimal trajectory. Possible paths the robot can take to solve each environment are marked with a red dotted line.

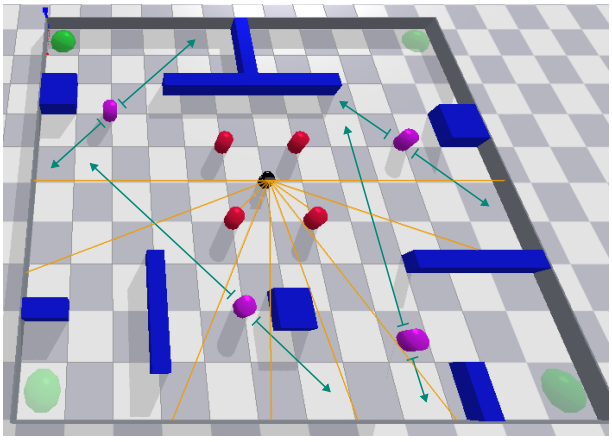


FIGURE 6. Dynamic environment in which the robot is equipped with a front facing lidar, depicted with orange rays. The robot starts in the center of the environment, the goal position is sampled at random from either of the four corners at the start. While navigating to the goal, the robot has to avoid several static and moving obstacles. The trajectories of the moving obstacles is indicated by green arrows. Solving the environment takes between 60 and 70 individual steps, depending on the sampled goal, position of dynamic obstacles and path the robot takes.

independent motors by setting target velocities for its two wheels.

The first three environments are static navigation tasks depicted in Fig 5. The 3×3 environment shown on the left is the smallest, the 4×4 environment (center) is of medium size, and the 5×5 environment (right) is the largest. In each environment, the robot starts at a fixed position in the upper left corner and has to navigate to a fixed goal position marked with a green sphere while avoiding collisions with the outer walls and the obstacles within the environment. The robot has a state space with three components for these tasks. The first two are its position in the plain in s_x and s_y coordinates, and the third is its orientation s_φ around the z-axis in radians. We use these environments to understand the scaling behavior of parameterized quantum circuits in the learning task, assess their behavior and performance for trajectories of increasing length and complexity and evaluate the effect of an increasing exploration demand.

We furthermore created a considerably more demanding environment to validate the robustness of the presented method with a higher dimensional state space and dynamic components in the learning task. In this environment, shown in Fig. 6, the robot is equipped with a simple, front-facing lidar that covers a range of 180° in the plane. The robot's state space contains ten distance measures in 20° intervals as well as the current distance and orientation to the goal. The robot starts in the center of the environment and has to navigate to a goal position, which is sampled at random at the beginning of each episode to be in either of the four corners.

We created all environments with the pybullet [67] real-time physics engine and set a control frequency of 100 Hz for collision detection and calculating forward dynamics.

The robot has three actions available (forward, turn left, turn right) to move in the environment. To move forward, the same target velocity is applied to both wheels, whereas for turning left and right, equal velocities but with opposing directions are set. Turning left or right causes a change in orientation between 40° and 50° depending on the current forward and angular velocity of the robot. Similarly, the robot moves between 0.15 and 0.2 units in the direction of its current orientation, where one unit corresponds to the length of one square on the environment floor. An action is chosen every 50 simulation steps, corresponding to an execution time of 0.5 seconds. With this control scheme, the robot needs about 20 consecutive actions to reach the goal in the 3×3 environment on a near-optimal trajectory, 30 steps in the 4×4 environment, and 45 steps in the large 5×5 environment. Possible paths the robot can take to solve the static environments are marked with red dotted lines in Fig. 5. In the dynamic environment, where the robot is equipped with a lidar, a typical trajectory leading to the goal takes about 60 to 70 steps, depending on the current goal, position of dynamic objects and path taken by the robot.

We use the same simple yet informative reward function to train the robot in all environments. The agent receives a positive reward for decreasing the distance to the goal as well as for reaching it, whereas increasing or maintaining the distance as well as collisions are penalized. The reward

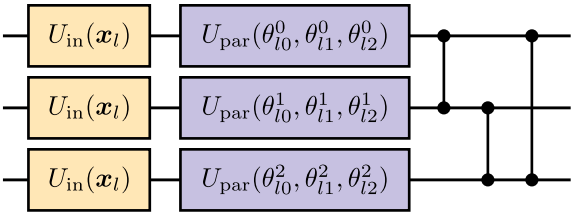


FIGURE 7. Circuit layout for layer l of our PQC ansatz used for the static environments. The encoding unitary U_{in} on each qubit q is given by $U^{(q)}_{\text{in},1}$, resp. $U^{(q)}_{\text{in},3}$. The encoding is followed by a general rotation gate U_{par} with three variational parameters $\theta^q_{l0}, \theta^q_{l1}, \theta^q_{l2}$ for each qubit and a full entanglement among all qubits with three controlled-Z gates. We omit the qubit index on all gates for readability. For the dynamic environment we use the same ansatz extended to 12 qubits.

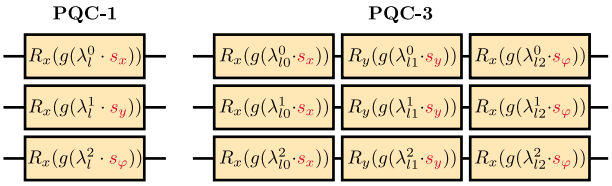


FIGURE 8. The two input encoding strategies used for training a parameterized circuit as action-value function. For the PQC-1 strategy shown on the left, each feature of the state $s = (s_x, s_y, s_\varphi)$ is encoded on an individual qubit in each layer using a single R_x gate. The PQC-3 encoding scheme uses three consecutive parameterized rotation gates $R_xR_yR_x$ to encode the entire state s on each qubit and layer as depicted on the right. For both strategies, before encoding, each feature of the state is scaled by a trainable parameter λ^q_{li} individual to each encoding gate, and an activation function $g : \mathbb{R} \to \mathbb{R}$ is applied. Through all our experiments, we use *arctangent* as activation functions.

function is given by:

$$r(s_t, s_{t+1}) = \begin{cases} 10.0 & \text{if } s_{t+1} \text{ is within the goal area} \\ 0.1 & \text{if } d_{\text{goal}}(s_{t+1}) < d_{goal}(s_t) \\ -1.0 & \text{for any collision} \\ -0.2 & \text{else} \end{cases} \tag{16}$$

where $d_{\text{goal}} : S \to \mathbb{R}$ is the euclidean distance of the robot to the goal area. The penalties in the reward function ensure that shorter trajectories are preferred by the agent. We consider the static environments solved when a total reward of 10.5 (3×3), 11.0 (4×4), and 10.0 (5×5) is reached. Higher rewards are possible, as the two larger environments have more than one possible path to the goal and we furthermore allow some tolerance for the length of the trajectory and the exact route. Therefore, these thresholds are a lower bound based on several manually determined valid trajectories. For the dynamic environment we do not set a threshold and observe the average evaluation reward over the entire training time.

In all environments, an episode ends when the robot reaches the goal, collides with an object, or when a maximum of 200 steps were executed during training.

B. LEARNING

We trained the simulated robot using three different paradigms: A baseline with a classical neural network as approximator for the action-value function and two different parameterized quantum circuits, distinguished by their encoding strategy for the classical input data.

For the classical baseline agent, we employ a three-layer, fully connected neural network with rectified linear unit activation on all but the final layer, which has a linear activation. In the static environments, the network takes the three components of the robot's state $s = (s_x, s_y, s_\varphi)$ as input, followed by two layers with u_1 and u_2 number of hidden units and three outputs corresponding to $Q(s, a_i), i \in \{1, 2, 3\}$. For the dynamic environment, we use the same neural network architecture, albeit with a 12 dimensional input for the ten lidar distance measurements as well as the distance and orientation to the goal. The number of trainable parameters $|\boldsymbol{\theta}_{NN}|$ including weights and biases for the classical neural network is therefore given by:

$$|\boldsymbol{\theta}_{NN}| = \underbrace{|s|u_1 + u_1}_{\text{first layer}} + \overbrace{u_1u_2 + u_2}^{\text{second layer}} + \underbrace{3u_2 + 3}_{\text{third layer}}. \tag{17}$$

Here $|s|$ is the dimensionality of the state space and u_i the units in the i-th layer.

In both quantum cases, we build our circuit on three qubits for the static environments, which aligns well with the dimensionality of the state space and the number of actions available to the agent. Both circuits follow the general approach depicted in Fig. 4 and are only different in their data encoding structure U_{in} and the number of layers L. The circuit layout for a single layer $l > 0$ is illustrated in Fig. 7, whereas the encoding strategies are shown in Fig. 8.

Our first data re-upload PQC model uses the encoding $U^{(q)}_{\text{in},1}$ on each qubit with the rotation gate R_x to encode one state feature on each qubit (PQC-1). For the second model, we use $U^{(q)}_{\text{in},3}$ for each qubit with rotation gates $R_xR_yR_x$ to encode all three state features on each qubit (PQC-3). As introduced in Sec. III, we scale each feature with a trainable parameter that is individual for each encoding gate and furthermore apply an activation function for which we choose the arctangent in all our experiments. The universal rotation $U^{(q)}_{\text{par}}$ on each qubit is composed of three parameterized Pauli rotation gates $R_xR_yR_z$ with trainable parameters.

For the large, dynamic environment with a 12 dimensional state space, we use circuits with 12 qubits. The PQC-1 as described above directly translate to this setting, whereas for the PQC-3 encoding we distribute all 12 features of the state space across four layers, each encoding three of the features, as outlined in Sec. III.

The number of trainable parameters for each quantum circuit $|\boldsymbol{\theta}_{PQC}|$ is the sum of variational parameters in the initial parameterized and the following L layers, the input scaling and the output scaling parameters, in total:

$$|\boldsymbol{\theta}_{PQC}| = \underbrace{3Q(L+1)}_{\text{variational}} + \overbrace{Qn_{\text{enc}}L}^{\text{input}} + \underbrace{3}_{\text{output}}. \tag{18}$$

Here $n_{\text{enc}} = 1$ for the PQC-1 and $n_{\text{enc}} = 3$ for the PQC-3 encoding, Q is the number of qubits in the circuit.

TABLE 1. The configurations used for the classical baseline in all three static environments with the number of units u_1 and u_2 for the first two layers of the neural network and the number of trainable parameters $|\theta_{NN}|$ for each configuration.

Hidden units		Parameters
u_1	u_2	static $\|\theta_{NN}\|$
8	8	131
16	8	227
16	16	387
32	16	707
32	32	1,238
64	32	2,435
64	64	4,611
128	64	8,963
128	128	17,411
258	128	34,307

TABLE 2. The configurations used for both quantum encoding strategies while training the three static environments. The number of layers L as well as the number of trainable parameters $|\theta_{PQC}|$, which include the variational, input, and output scaling parameters, are outlined.

PQC-1	PQC-3	Parameters
L	L	static $\|\theta_{PQC}\|$
12	8	156
15	10	192
18	12	228
21	14	264
24	16	300
27	18	336
30	20	372
33	22	408
36	24	444
39	26	480

Based on these three architectures, two quantum and one classical, we performed experiments with different sizes of each architecture for the static environment. For the classical neural network, we evaluated a total of ten configurations for the number of units u_1 and u_2 in the first and second hidden layers. The configurations and their number of trainable parameters $|\theta_{NN}|$ are outlined in Table 1.

Likewise, we included ten configurations for each quantum encoding strategy with an increasing number of layers L. As the different types of encoding lead to a different amount of trainable parameters for each layer, we arranged the number of layers to have an equal number of parameters between both. The number of layers L for both strategies, as well as the number of trainable parameters, including variational, input, and output scaling parameters are summarized in Table 2.

With regard to the parameter scaling, we emphasize that the number of trainable parameters roughly doubles with each increase of the configuration size for the classical baseline, whereas the scaling for the quantum circuits is only linear. Thus, the largest neural network we employed has about two orders of magnitude (34,307) more parameters than the largest quantum circuits (480). With the dynamic environment, we only perform experiments with a single configuration for each architecture, due to the considerable computational effort involved in simulating very large quantum circuits. The neural network used as baseline has 256 and 128 hidden units (36,611 trainable parameters), the PQC-1 circuit has 24 layers, and the PQC-3 circuit 16 layers (bot 1,191 trainable parameters).

We set a learning rate for the stochastic gradient descent of 10^{-3} for the classical baseline as well as for the variational parameters in both quantum circuit architectures. The input and output scaling parameters were trained with a learning rate of 10^{-2} for both PQC-1 and PQC-3 as encoding.

For the hyper-parameters specific to the DDQN algorithm, we use the same values for all experiments. The replay buffer was set to a capacity of 20,000 experience samples and is initially filled with 5,000 samples from executing a fully random policy in the environment, before each training starts. The agent is trained after each step it executes in the environment with a mini-batch of 64 samples from the replay buffer. Exploration is handled with an ϵ-greedy policy as introduced in Sec. III, starting at $\epsilon = 1.0$ and setting $\epsilon \leftarrow 0.99\epsilon$ every 250 training steps. Total training time is limited to 50,000 steps in all environments, except for the dynamic environment, in which we train 100,000 steps. We evaluate the current performance of the learned policy after every 100 training steps by performing 10 consecutive runs within the environment. Once the average total reward over those 10 runs surpasses the solution criterion for any of the static environment outlined above, the training is stopped early, whereas we do not stop the training early in the dynamic environment.

To gather sufficient data on the robustness and reproducibility of the learning procedure, we repeat the training for each combination of static environment, architecture, and configuration 20 times, each time with a different random seed. We do not set the random seeds to specific values but have them provided by the operating system's randomness source instead. We consider a configuration successful if at least 15 of 20 training runs solve the environment. In the dynamic environment, we repeat each training 10 times and record the evaluation performance to evaluate the robustness of the presented methods in a considerably larger and more challenging environment and large quantum circuits.

All hyper-parameters were determined empirically before the actual experiments. Our main goal was to find a set of parameters that would enable reliable and robust training under mostly identical premises for all three architectures, their respective configurations, and for all three environments, as our main interest is not in absolute performance but in comparison of architectures and scaling behavior. An overview of all hyper-parameters can be found in Table 6 in App. B.

C. HARDWARE, SOFTWARE AND COMPUTATIONAL RESOURCES

All experiments were conducted on a workstation equipped with an AMD Ryzen Threadripper Pro 3975WX 32 core/64 thread CPU, 128 GB of RAM, and an NVIDIA RTX A6000 GPU. On the software side, we used TensorFlow [68] as framework for all general and classical machine learning

tasks, TensorFlow Quantum [69] for quantum machine learning specific tasks, as well as TensorFlow Agents [70] for all components related to Deep Reinforcement Learning and a stable DDQN implementation. TensorFlow Quantum integrates the Cirq [71] quantum computing framework for building and running quantum circuits, as well as the Qsim [72] quantum circuit simulator.

All quantum circuit simulations in Qsim were executed under idealized noise-free conditions. We compute the expected value of observables directly from the system's state vector. If experiments were to be conducted in a shot-based simulation, a large enough number of circuit receptions would need to be chosen to estimate the expected values of observables with sufficient accuracy. Similarly, if experiments were to be reproduced on quantum hardware or with simulated hardware noise, appropriate measures for error mitigation would have to be taken into account, which is outside of the scope of this work.

All simulated robotic environments were built using the PyBullet [67] python bindings to the Bullet real-time physics SDK. For the baseline experiments described in App. A, we furthermore used the OpenAI Gym [19] suite.

We released our robotic environments as well as the entire experimental setup under an Open Source license for interested researchers to reproduce, verify, or build upon our work. Both can be found together with installation and usage instructions under the following addresses:

- Environments: https://github.com/dfki-ric-quantum/qdrl-turtlebot-env
- Experiments: https://github.com/dfki-ric-quantum/qdrl-turtlebot-eval

Concerning the computational resources and wall-clock time needed to conduct our experiments, we observe the following: For the classical baseline, training a single neural network within the range of configurations and across all environments requires 1.5 GB of RAM and 600 MB of VRAM, assuming TensorFlow uses GPU acceleration. Training the network for 1.000 steps takes on average 25 seconds wall-clock time with our hardware setup, which is relatively stable overall environments and network sizes.

For the three static environments, the number of qubits of a quantum circuit, which is the same in both our encodings and across all configurations, primarily determines the memory requirements for its simulation. Simulating the training of each quantum circuit requires about 2.2 GB of system memory and 500 MB of VRAM. The quantum circuit simulator imposes a substantial computational overhead, resulting in much longer execution in terms of wall-clock time and a nearly linear growth with respect to the number of layers. The average wall-clock time for 1.000 training steps in the 5×5 environment with the PQC-1 and PQC-3 encoding are summarized in Table 3. Learning the dynamic environment with either encoding on 12 qubits for the number of layers we use, requires considerably more computational resources. A single run requires about 18 GB of system memory and 1.2 GB of VRAM. Training for 1.000 steps takes on average one hour, hence the training the full 100.000 steps is finished in about four days.

TABLE 3. Average wall-clock time for 1.000 training steps with the PQC-1 and PQC-3encoding in the 5×5 environment. The time necessary to train the model grows nearly linear in the number of layers.

PQC-1		PQC-3	
L	Wall-clock time (s)	L	Wall-clock time (s)
12	393	8	361
15	467	10	442
18	544	12	520
21	629	14	605
24	722	16	661
27	815	18	743
30	878	20	811
33	967	22	892
36	1033	24	954
39	1088	26	994

V. RESULTS

A. OPENAI GYM ENVIRONMENTS

As we work with custom environments, we first compared their complexity to established OpenAI Gym environments, namely `FrozenLake` and `Cartpole-v1`. The results for both environments with classical neural networks and PQCs are described in Appendix A. With these comparative experiments, we can demonstrate that our navigation environments are indeed substantially more difficult to solve for the DDQN algorithm.

B. STATIC ENVIRONMENTS

Performing experiments with the 10 classical neural network configurations and 10 configurations for both PQC input encoding variants provides insight into the scaling behavior and robustness across multiple training runs for each architecture in the given robotic reinforcement learning task. The complete statistics for all experiments in the static environments are outlined in Tables 7 to 9 in Appendix C and visualized in Fig. 9.

The first noteworthy result is that all three architectures, the classical neural network as well as both types of quantum circuits are capable of learning an optimal action-value function in all three environments in 20 out of 20 training runs with a sufficiently large configuration (see column *Solved* in the Tables 7 to 9). More precisely, for the 3×3 and 4×4 environments, all configurations of the architectures solve the environments, whereas in the 5×5, the three smallest neural networks, the two smallest PQC-1, and the smallest PQC-3 configurations were unable to solve the environment in at least 15 out of 20 runs. Also, we find that an increase of the model size in terms of the number of trainable parameters leads to a decreased median and mean in required training steps. This trend converges after the model size reaches a sufficient size. In our case, the two biggest configurations of all three architectures are similar in terms of median and mean of training steps. In addition, in most cases, the range

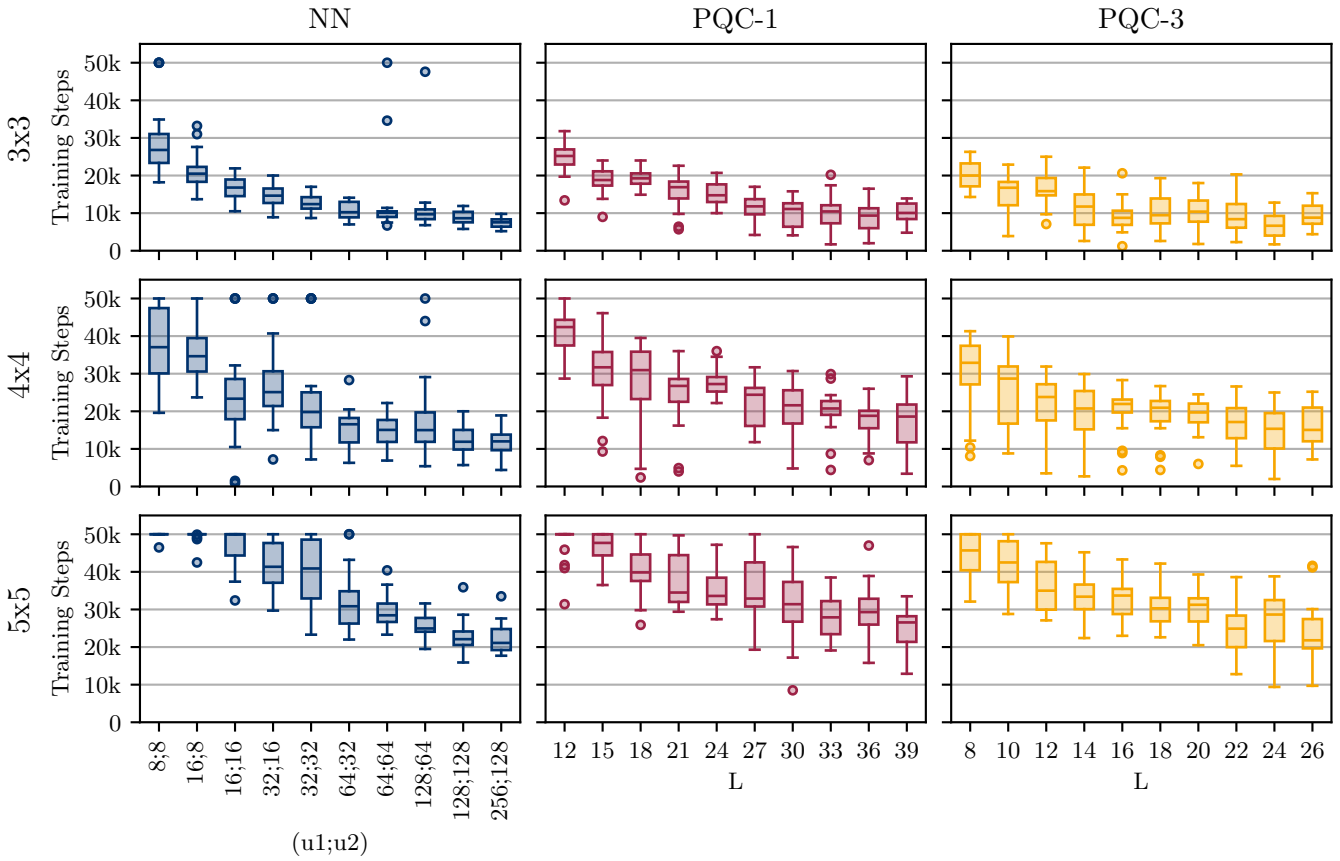


FIGURE 9. **Statistics on training time for all three static environments, architectures, and configurations. The results per environment are shown from the top to bottom row, whereas the classical baseline neural network architecture (NN), as well as both types of quantum circuits (PQC-1 and PQC-3) are arranged from left to right. For each combination of environment and architecture all related configurations, that is number of units (u_1, u_2) for the classical baseline and number of layers L for both quantum encoding strategies, are reported. Each box shows the median training steps over 20 runs for each configuration with different random seeds as well as the lower and upper quartile, range and flier points.**

and standard deviation decreases as the model size increases, resulting in our largest models being the best-performing and most stable configurations.

In the following, we focus on the two best PQC-1 and PQC-3 configurations and compare them with four different neural network configurations. From our data, we select the classical neural networks such that they have a similar number of parameters or one or two orders of magnitude more parameters than the PQC configurations. Table 4 summarizes the mean and standard deviation of the required training steps for these configurations. From this, we observe the following general trends with regard to the number of trainable parameters:

- With about the same order of magnitude of parameters, the quantum circuits perform better and converge to an optimal solution faster. This is especially true for the PQC-3 architecture.
- With about one order of magnitude more parameters for the classical neural network, its performance is about equal compared to the parameterized quantum circuits.
- A further increase in the number of parameters up to two orders of magnitude more for the neural network puts it slightly ahead of both quantum circuit architectures in all observed metrics.

Furthermore, we can compare the results of the two best PQC-1 and PQC-3 configurations in Table 4. For all environments, the best PQC-3 architecture yields faster convergence to an optimal policy compared to the best PQC-1 encoding scheme.

This advantage is relatively stable across all three static environments, suggesting that for the navigation setup considered in this work, a larger number of encoding gates is beneficial. A larger variety of environments with regards to complexity and type of task to learn would need to be evaluated to make more definitive statements on this.

The evaluation performance for the best configuration for each architecture in all three environments is shown in Fig. 10. During training, we observed the agent's performance with the trained policy every 100 steps for 10 consecutive runs and evaluated its mean reward. In all three environments, the classical baseline converges to an optimal policy faster, albeit with two orders of magnitude more trainable parameters. In the 3×3 environment, the PQC-3 architecture reaches a solution notably faster than the PQC-1 architecture. With increasing environment complexity, both types of quantum

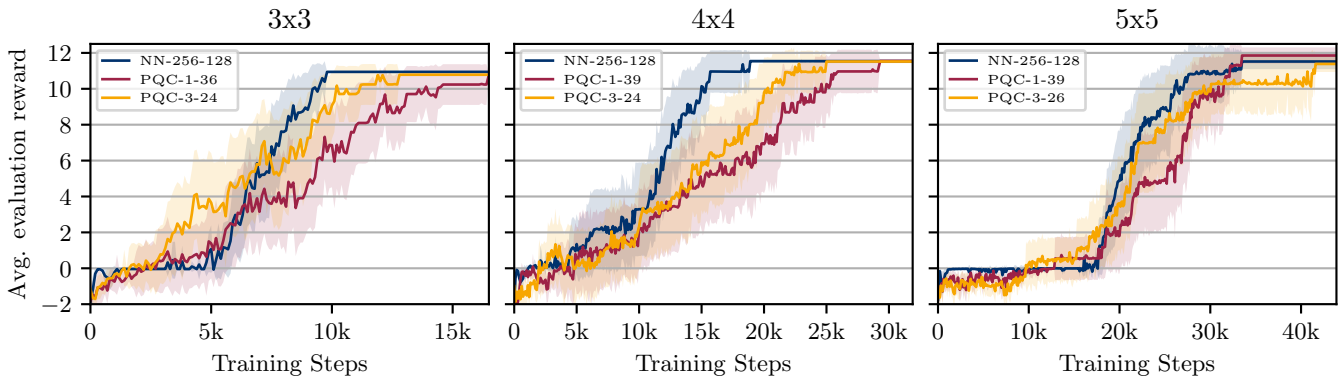


FIGURE 10. Evaluation results for the configurations with the best mean performance for each architecture in all three static environments. From left to right the mean evaluation performance over 20 consecutive runs with the trained policy outlined by the 95% confidence interval for the 3×3, 4×4, and 5×5 environments is shown. Negative rewards are rescaled by a factor of 0.1 to improve readability. When not bound by the number of trainable parameters, the classical baseline models performs slightly better than both quantum architectures in all three environments. Training was stopped once the mean evaluation reward reached the solution threshold in the environment, plots are padded with additional evaluation runs for better comparability.

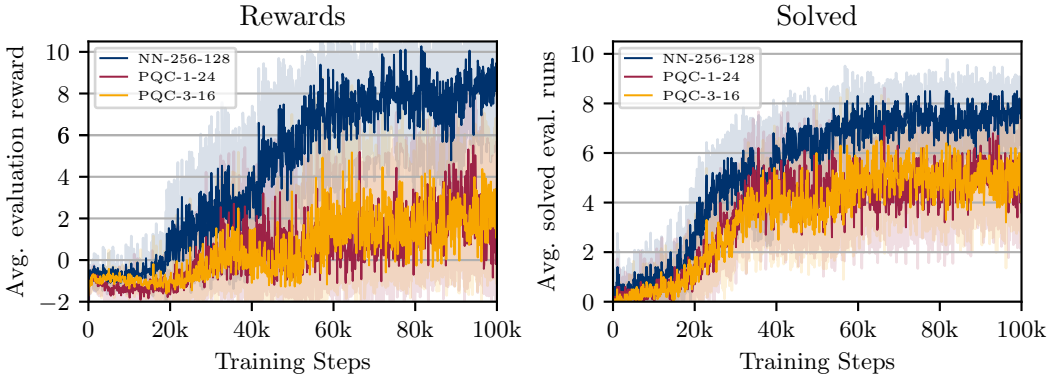


FIGURE 11. Evaluation performance for the three tested configurations in the dynamic environment with regards to the evaluation rewards (left) and number of solved evaluation runs (right). Both metrics are recorded every 100 training steps for 10 consecutive evaluation runs and 10 receptions of the experiment with different random seeds. The outlines mark the 95% confidence interval. Negative rewards are scaled by a factor of 0.1 to improve readability. For both metrics we find, that while there is learning progress for both quantum circuit architectures, the classical baseline provides better and more robust results, albeit with about 30 times the number of trainable parameters.

circuits perform increasingly similarly, whereas the classical neural network remains ahead of both. This finding emphasizes the trends discussed above.

C. DYNAMIC ENVIRONMENT

In the large, dynamic navigation environment, our main interest is the robustness of the presented method in a substantially more demanding task and employing considerably larger quantum circuits. To this end, we trained a classical baseline and two large quantum circuits with the two encoding strategies for 100,000 iterations on the environment. We evaluated the performance in 10 consecutive runs every 100 training steps. Fig. 11 shows the training progress regarding the mean evaluation reward and number of solved evaluation runs, with averages taken over 10 repetitions of the experiment with different random seeds.

The classical baseline neural network performs considerably better in this task than both employed quantum circuits, learns policies that achieve higher mean rewards, solves more evaluation runs on average and is more robust in the dynamic setting. Both quantum architectures perform about the same concerning to both metrics. The fact that the difference between the classical model and quantum circuits regarding the mean reward is larger than for the number of solved evaluation runs is explained by the observation that the environment allows for much larger negative rewards on failed runs than positive rewards on the successful ones. Hence, the negative rewards will dominate the result if several runs fail.

Table 5 summarizes the best results achieved by all three architectures. The classical baseline reaches its best average performance after 81,500 training steps, whereas both quantum circuits require more than 94,000 steps. Additionally, the mean evaluation reward of 10.27 for the classical neural network is considerably larger than 5.50 and 3.87 for the PQC-1 and PQC-3 architecture.

After this training duration, the robot can successfully navigate to the goal on average in 8.5 out of 10 evaluation runs over 10 repeated experiments. Solving 6.7 and 6.3 evaluation runs on average for the quantum architectures shows noteworthy training progress for both, but with considerably

TABLE 4. Mean number of training steps and standard deviation in all three environments for the two largest configurations for both quantum circuit architectures in comparison to two classical baseline models with about the same order of magnitude of trainable parameters as well as two larger neural networks. We find, that with about the same order of magnitude of parameters, the two quantum architectures converge to an optimal solution in fewer training steps. With an order of magnitude more parameters, the classical neural network performs comparable or better and achieves better performance compared to both quantum architectures with further increasing number of trainable parameters.

			No. of training steps					
	Environment:		3×3		4×4		5×5	
Arch.	Config.	$\lvert\theta\rvert$	Mean	Std.	Mean	Std.	Mean	Std.
NN	(16;8)	227	21,075	5,050	35,910	7,672	49,515	1,684
	(16;16)	387	16,405	3,129	24,565	13,736	46,570	5,237
	(64;32)	2,435	10,635	2,290	15,055	4,315	32,135	8,049
	(256;128)	34,307	7,495	1,359	11,480	3,899	22,220	4,122
PQC-1	36	444	9,150	3,746	17,705	4,678	29,260	6,614
	39	480	10,060	2,905	16,880	7,288	25,110	5,309
PQC-3	24	444	6,850	3,146	14,665	6,068	25,757	7,334
	26	480	9,515	3,347	15,875	5,164	23,635	7,535

TABLE 5. Statistics over the training in the dynamic navigation environment. For all three configurations the number of training steps to the best performing evaluation runs, the mean evaluation reward, the mean number of solved evaluation runs as well as their respective standard deviations are reported. Statistics are taken over 10 consecutive evaluation runs executed every 100 steps and 10 repetitions of the experiment with different random seeds.

			Reward		Solved	
Arch.	Config.	Steps	Mean	Std.	Mean	Std.
NN	(256;128)	81,500	10.27	1.37	8.50	0.92
PQC-1	24	94,200	5.50	3.12	6.70	1.79
PQC-3	16	94,500	3.87	3.53	6.30	1.27

worse performance. We furthermore observe larger standard variations on both metrics for the quantum models compared to the classical baseline, suggesting less robust and less reliable training results.

VI. DISCUSSION

In this work, we investigated the potential and scaling of hybrid quantum deep reinforcement learning as a method to learn autonomous robotic behaviors. We systematically evaluated two different quantum circuit architectures in three simulated static environments of increasing difficulty and with increasing circuit sizes. These results were compared to a classical neural network baseline. Additionally we tested the robustness of the presented method in a considerably more demanding learning task, using a dynamic navigation environment.

Both quantum architectures as well as the classical baseline yielded sufficient action-value functions for the simulated robot in all three static environments. Not considering the number of trainable parameters, the classical baseline models outperformed the quantum circuits in terms of training speed and stability. A noteworthy result, which is in line with previous findings from the quantum deep reinforcement learning research is that both best-performing quantum circuits were capable of solving the environments within a similar number of training steps as classical neural networks with about one order of magnitude more trainable parameters. This observation is consistent across all three environments. The best-performing quantum models have 444 and 480 trainable parameters, the classical baseline was sufficient to solve the 3×3 and 4×4 with a similar amount of parameters, albeit with substantially more training steps. At this model size, the neural network was unable to fit an optimal action-value function for the 5×5 environment in most of the 20 training runs within the $50,000$ training step threshold we set, whereas both quantum architectures still succeeded with only 300 parameters.

Comparing both quantum circuit architectures, we find that the PQC-3 embedding performs better than the PQC-1 embedding in all three environments, suggesting that in this context, having more encoding gates for the same data is beneficial, although the difference becomes less pronounced with increasing environment difficulty. Moreover, our experiments show that with increasing environment size, quantum circuits with more layers are needed to solve the tasks consistently, especially for the 5×5 environment. This finding is consistent with the results from [66], as adding more layers increases the expressiveness of the circuit, which makes it possible to approximate more complex action-value functions.

Testing the same learning methods in a more demanding, dynamic navigation environment, we find that both quantum circuit architectures get outperformed by the classical neural network regarding reward, solved evaluations, training duration and robustness. Given the limited scope of this experimental setup, it remains open, if this result can be improved by changes on the training procedure, circuit architecture, encoding strategies or by increasing the circuit size. We consider these questions to be out of scope for this work, but plan to address them in future.

Additionally, our results demonstrate that PQCs of this size are trainable in a quantum circuit simulator for a practical problem class, which does not necessarily follow from previous considerations on the expressiveness of PQCs [65], [66].

Beyond these results, we can confirm, similar to e. g., [73], that training PQCs is fairly unstable regarding changes in the hyperparameters compared to classical neural networks.

Considering the best-performing PQCs architectures in this work, we have to emphasize that this configuration is not to be considered efficient or even viable for current quantum hardware. The largest employed circuit using the PQC-3 architecture has almost 200 gates per qubit, not considering additional gates that could be introduced by transpiling it to a native gate set of any quantum hardware platform. Circuits with long execution times and more gates are more prone to noise on current quantum hardware. Hence, we would not expect meaningful results without substantial error mitigation efforts. Training the circuits directly on quantum hardware was also not a realistic option, given the total number of experiments we conducted and the limited availability and access to quantum computing hardware. Consequently, we limited our study to an idealized environment in a noise-free quantum circuit simulator.

VII. OUTLOOK

Understanding the characteristics of PQCs is an ongoing research topic. For PQCs to offer advantages over classical solutions, there are still some open questions that have to be addressed. Concerning expressiveness, the authors of [66] showed that PQCs with the data-reupload technique can represent real-valued truncated Fourier series. While this could be considered a weak restriction on the expressiveness, it remains unclear if they are rich enough to approximate deep RL algorithm outputs for more complex behaviors. In [74], the authors leverage that PQCs represent truncated Fourier series by showing that classical models can be obtained efficiently from trained PQCs. They also report no advantage in the performance nor trainability of PQCs over classical models for the problems they consider. The trainability of PQCs is analyzed in more detail by Bittel et al. [75], who rigorously prove that classical training is NP-hard, and by the authors of [76], who found many sub-optimal local minima in the gradient landscape. Moreover, barren plateaus [77] are one additional hurdle for trainability. These works and our results indicate that PQCs mark the beginning of quantum machine learning in general and quantum deep reinforcement learning specifically. These methods have to be developed further substantially to yield potential improvements over classical learning techniques.

Our results provide additional insight into the scaling behavior and applicability of hybrid quantum deep reinforcement learning based on PQCs, especially with regard to more demanding problems than previously considered. Our experimental setting is focused on three static environments and two different quantum circuit architectures. Furthermore, we studied the robustness of these methods in a more demanding, dynamic navigation task, although with limited scope. Hence more empirical research is needed to substantiate our findings further, and produce more conclusive results.

The second area is the applicability of quantum machine learning and quantum deep reinforcement learning in real-world applications, especially in the field of robotics. While we have demonstrated quantum deep reinforcement learning in a limited robotic scenario, actual advantages of the presented method over classical deep reinforcement learning have yet to be shown. While previous works demonstrated a quantum advantage for a certain class of problems [42] intractable for classical learning methods, it remains an open question if this advantage can be translated to problems from e. g., robotic domains.

Another crucial topic linked to real-world applications, is the encoding scheme of classical data into the quantum circuit. With the proposed methods, the required number of qubits and the operations per qubit scale linearly in the best case with the dimensionality of the state space. It will be interesting to see how different encoding techniques like e. g., amplitude encoding [78] would impact the learning behavior. Also, we limited our experiments to state spaces of small dimensionality to account for the computational demands of simulating quantum circuits on a classical computer. While this imposed no detriments on our learning scenarios, having high dimensional sensory data, e. g., high-resolution image data, is common in more complex robotic tasks. How to encode classical data with hundreds, thousands, or more dimensions efficiently onto quantum devices with their current limitations is an open question. Investigating classical preprocessing, compression, and dimensionality reduction techniques in this context could potentially enable quantum deep reinforcement learning for such scenarios. Tensor networks, as indicated by Chen et al. [79], are one further promising candidate for encoding more complex robotic data.

Additionally, the scope of our work is limited with regards to actual quantum hardware and its properties. We performed all our experiments with a quantum circuit simulator which, enabled us to employ circuits with a depth beyond what current hardware provides and also removed the necessity to deal with the noise that typically comes with the execution of algorithms on current quantum hardware. The execution of even simpler machine learning tasks on actual quantum hardware would be further limited by their sparse availability, complexity, and high usage cost. To circumvent these issues, research into techniques to combine quantum simulators and quantum hardware in an efficient training setup could be a practical route forward.

We understand our work as a contribution toward application-focused empirical research on quantum algorithms in a robotic context. We see this as a viable route to accelerate the development and understanding of quantum algorithms, quantum machine learning, and the application of quantum techniques in deep reinforcement learning. Looking forward, we believe that quantum algorithms, together with future hardware developments in the field of quantum computing, will contribute to the advancement of autonomous robotics.

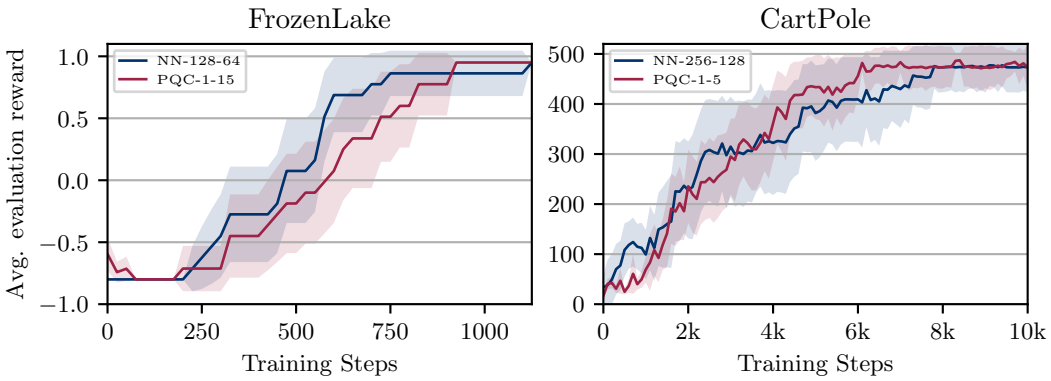


FIGURE 12. **Evaluation results of the FrozenLake and CartPole environment. Each plot shows the mean evaluation performance over 20 consecutive runs with the trained policy. The evaluation was performed every 25, resp. 100, training steps and training stopped, once the environment was solved with the policy. Plots are padded with additional evaluation runs to have equal lengths for better comparability. The outline of each plot is the 95% confidence interval.**

VIII. ACKNOWLEDGMENT

We thank Elie Mounzer, Gunnar Schönhoff, Patrick Draheim, Melvin Laux, and Alexander Fabisch for their valuable feedback on this manuscript. We furthermore thank the anonymous reviewers for their constructive remarks and recommendations to improve this paper.

APPENDIX A
COMPARISON TO BASELINE ENVIRONMENTS

We use our learning setup to solve the benchmark OpenAI gym [19] environments `FrozenLake` and `CartPole-v1`. This way, we underline our argument that the navigation tasks are indeed more complex and difficult to learn.

To learn the `FrozenLake` environment, we use binary encoding for the state features, adapt the circuit to four qubits, and adapt the parameters for epsilon decay and max steps per episode. All other learning hyper-parameters are unchanged. We also use the *arctangent* activation function and trainable parameters on the input features as well as four trainable output parameters. A full list of the hyper-parameters is given in Table 6 in App. B. The left plot of Fig. 12 shows that 20 runs with a classical neuronal networks with $(128, 64)$ hidden units learn an optimal policy in fewer than 1,250 training steps with a mean of 510 steps, a median of 513 steps, and a standard deviation of 204 steps. The classical architecture takes roughly 20 times as long to learn an optimal policy in our simplest navigation task. Similar results hold for PQCs with one input encoding and 15 layers (PQC-1-15). Here, the training finishes on average in 593 steps, with a median of 613 steps and a standard deviation of 205 steps. This result is in alignment (slightly better) with [41]. We want to emphasize that we did not fine-tune the hyper-parameters for the `FrozenLake` environment but were still able to learn the task much faster than for our 3×3 navigation environment.

We obtained similar results for the `Cartpole-v1` environment as depicted in the right plot of Fig. 12. For this environment, we adapted the PQC to 4 qubits and used the measurements $\sigma_z^{(1)}\sigma_z^{(2)}$ and $\sigma_z^{(3)}\sigma_z^{(4)}$ for the post-processing. We adapted the epsilon decay parameters and other hyper-parameters slightly, as shown in Table 6. Averaged over 20 runs, the classical network with $(256, 128)$ hidden units is able to solve `CartPole-v1` with an average of 3,645 training steps (median: 2,350, standard deviation: 2,600) with slightly adapted hyper-parameters. That is approximately twice as fast as the same network architecture learns the 3×3 navigation environment. For the PQCs, the configuration with one input encoding and five layers needed fewer than 10,000 training steps to learn the optimal policy, which is notably faster than reported in the literature (e. g., [41] for `Cartpole-v0`). The PQC-1-5 ansatz solves `CartPole-v0` in a similar time (mean: 4065, median: 4050, standard deviation: 1933) and thus solves it faster than larger PQC-1 configurations solve the 3×3 navigation environment.

Hence, we conclude that our navigation tasks are considerably more challenging than `FrozenLake` and `Cartpole-v1` for the (hybrid quantum) DDQN algorithm.

APPENDIX B
HYPER-PARAMETERS FOR EXPERIMENTS

The hyper-parameters used in all environments and learning setups are outlined in Table 6.

APPENDIX C
RESULT DETAILS

Detailed statistics over all conducted experiments are reported in Tables 7, 8 and 9.

REFERENCES

[1] Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine. Deep Reinforcement Learning for Robotic Manipulation with Asynchronous Off-Policy Updates. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3389–3396, 2017.

[2] A. Rupam Mahmood, Dmytro Korenkevych, Gautham Vasan, William Ma, and James Bergstra. Benchmarking Reinforcement Learning Algorithms on Real-World Robots. In *Proceedings of The 2nd Conference on Robot Learning*, pages 561–591. PMLR, 2018.

[3] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, and Sergey Levine. Scalable Deep Reinforcement

Learning for Vision-Based Robotic Manipulation. In *Proceedings of The 2nd Conference on Robot Learning*, volume 87 of *Proceedings of Machine Learning Research*, pages 651–673. PMLR, 2018.

[4] Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafał Józefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, Jonas Schneider, Szymon Sidor, Josh Tobin, Peter Welinder, Lilian Weng, and Wojciech Zaremba. Learning Dexterous In-Hand Manipulation. *Int. J. Robot. Res.*, 39(1):3–20, 2020.

[5] Ashley Montanaro. Quantum algorithms: an overview. *npj Quantum Information*, 2(1), 2016.

[6] Peter W. Shor. Algorithms for Quantum Computation: Discrete Logarithms and Factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 124–134, 1994.

[7] Lov K. Grover. A Fast Quantum Mechanical Algorithm for Database Search. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, page 212–219, New York, NY, USA, 1996. Association for Computing Machinery.

[8] Aram W. Harrow, Avinatan Hassidim, and Seth Lloyd. Quantum algorithm for linear systems of equations. *Phys. Rev. Lett.*, 103:150502, Oct 2009.

[9] Leonardo Alchieri, Davide Badalotti, Pietro Bonardi, and Simone Bianco. An introduction to quantum machine learning: from quantum logic to quantum deep learning. *Quantum Machine Intelligence*, 3:1–30, 2021.

[10] Frank Kirchner. AI-Perspectives: The Turing Option. *AI Perspectives*, 2(1):1–12, 2020.

[11] Paul Benioff. Quantum robots and quantum computers, 1997.

[12] Paul Benioff. Quantum robots and environments. *Phys. Rev. A*, 58:893–904, 1998.

[13] Paul Benioff. Space searches with a quantum robot, 2000.

[14] Paul Benioff. Quantum robots plus environments. In *Quantum Communication, Computing, and Measurement 2*, pages 3–9. Springer, 2002.

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

[16] Michael Brooks. Beyond Quantum Supremacy: The Hunt for Useful Quantum Computers. *Nature*, 574(7776):19–21, 2019.

[17] Edward Farhi and Hartmut Neven. Classification with Quantum Neural Networks on Near Term Processors, 2018.

[18] Marcello Benedetti, Erika Lloyd, Stefan Sack, and Mattia Fiorentini. Parameterized Quantum Circuits as Machine Learning Models. *Quantum Sci. Technol.*, 4(4):043001, 2019.

[19] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym, 2016.

[20] Nico Meyer, Christian Ufrecht, Maniraman Periyasamy, Daniel D. Scherer, Axel Plinge, and Christopher Mutschler. A survey on quantum reinforcement learning, 2022.

[21] Daoyi Dong, Chunlin Chen, Hanxiong Li, and Tzyh-Jong Tarn. Quantum reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 38(5):1207–1220, oct 2008.

[22] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Adaptive Computation and Machine Learning series. MIT Press, 2018.

[23] Daoyi Dong, Chunlin Chen, Jian Chu, and Tzyh-Jong Tarn. Robust quantum-inspired reinforcement learning for robot navigation. *IEEE/ASME Transactions on Mechatronics*, 17(1):86–97, 2012.

[24] Yuanjian Li, A. Hamid Aghvami, and Daoyi Dong. Intelligent trajectory planning in uav-mounted wireless networks: A quantum-inspired reinforcement learning perspective. *IEEE Wireless Communications Letters*, 10(9):1994–1998, 2021.

[25] Christopher John Cornish Hellaby Watkins. *Learning From Delayed Rewards*. PhD thesis, King's College, Cambridge United Kingdom, 1989.

[26] Yazhou Hu, Fengzhen Tang, Jun Chen, and Wenxue Wang. Quantum-enhanced reinforcement learning for control: a preliminary study. *Control Theory and Technology*, 19(4):455–464, 2021.

[27] Qing Wei, Hailan Ma, Chunlin Chen, and Daoyi Dong. Deep reinforcement learning with quantum-inspired experience replay. *IEEE Transactions on Cybernetics*, 52(9):9326–9338, 2022.

[28] Vedran Dunjko, Jacob M. Taylor, and Hans J. Briegel. Quantum-enhanced machine learning. *Phys. Rev. Lett.*, 117:130501, Sep 2016.

[29] Vedran Dunjko, Jacob M. Taylor, and Hans J. Briegel. Advances in quantum reinforcement learning. In *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE, oct 2017.

[30] Valeria Saggio, Beate E. Asenbeck, Arne Hamann, Teodor Strömberg, Peter Schiansky, Vedran Dunjko, Nicolai Friis, Nicholas C. Harris, Michael Hochberg, Dirk Englund, Sabine Wölk, Hans J. Briegel, and Philip Walther. Experimental quantum speed-up in reinforcement learning agents. *Nature*, 591(7849):229–233, 2021.

[31] Arne Hamann and Sabine Wölk. Performance analysis of a hybrid agent for quantum-accessible reinforcement learning. *New J. Phys.*, 24(3):033044, mar 2022.

[32] Nicola Dalla Pozza, Lorenzo Buffoni, Stefano Martina, and Filippo Caruso. Quantum reinforcement learning: the maze problem. *Quantum Mach. Intell.*, 4(1):1–10, 2022.

[33] Hans J. Briegel and Gemma De las Cuevas. Projective simulation for artificial intelligence. *Scientific reports*, 2(1):1–16, 2012.

[34] Giuseppe Davide Paparo, Vedran Dunjko, Adi Makmal, Miguel Angel Martin-Delgado, and Hans J Briegel. Quantum speedup for active learning agents. *Physical Review X*, 4(3):031002, 2014.

[35] Theeraphot Sriarunothai, Sabine Wölk, Gouri Shankar Giri, Nicolai Friis, Vedran Dunjko, Hans J. Briegel, and Christof Wunderlich. Speeding-up the decision making of a learning agent using an ion trap quantum processor. *Quantum Science and Technology*, 4(1), 2018.

[36] 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. *IEEE Access*, 8:141007–141024, 2020.

[37] Maria Schuld and Francesco Petruccione. *Machine learning with quantum computers*. Springer, 2021.

[38] Piotr Gawłowicz and Anatolij Zubow. ns-3 meets openai gym: The playground for machine learning in networking research. In *Proceedings of the 22nd International ACM Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, MSWIM '19, page 113–120, New York, NY, USA, 2019. Association for Computing Machinery.

[39] Owen Lockwood and Mei Si. Reinforcement Learning with Quantum Variational Circuit. In *AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 16, pages 245–251, 2020.

[40] Iris Cong, Soonwon Choi, and Mikhail D. Lukin. Quantum Convolutional Neural Networks. *Nat. Phys.*, 15(12):1273–1278, 2019.

[41] Andrea Skolik, Sofiene Jerbi, and Vedran Dunjko. Quantum Agents in the Gym: a variational quantum algorithm for deep Q-learning. *Quantum*, 6:720, 2022.

[42] Sofiene Jerbi, Casper Gyurik, Simon C. Marshall, Hans J. Briegel, and Vedran Dunjko. Parametrized Quantum Policies for Reinforcement Learning. In *Advances in Neural Information Processing Systems*, volume 34, 2021.

[43] Yunchao Liu, Srinivasan Arunachalam, and Kristan Temme. A rigorous and robust quantum speed-up in supervised machine learning. *Nature Physics*, 17(9):1013–1017, 2021.

[44] Fabio Sanches, Sean Weinberg, Takanori Ide, and Kazumitsu Kamiya. Short quantum circuits in reinforcement learning policies for the vehicle routing problem. *Phys. Rev. A*, 105(6), jun 2022.

[45] Wouter Kool, Herke van Hoof, and Max Welling. Attention, learn to solve routing problems! In *International Conference on Learning Representations*, 2019.

[46] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[47] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

[48] Yunseok Kwak, Won Joon Yun, Soyi Jung, Jong-Kook Kim, and Joongheon Kim. Introduction to quantum reinforcement learning: Theory and pennylane-based implementation. In *2021 International Conference on Information and Communication Technology Convergence (ICTC)*, pages 416–420, 2021.

[49] Jen-Yueh Hsiao, Yuxuan Du, Wei-Yin Chiang, Min-Hsiu Hsieh, and Hsi-Sheng Goan. Unentangled quantum reinforcement learning agents in the openai gym, 2022.

[50] Dániel Nagy, Zsolt Tabi, Péter Hága, Zsófia Kallus, and Zoltán Zimborás. Photonic quantum policy learning in openai gym. In *2021 IEEE International Conference on Quantum Computing and Engineering (QCE)*, pages 123–129, 2021.

[51] Qingfeng Lan. Variational quantum soft actor-critic, 2021.

[52] El Amine Cherrat, Iordanis Kerenidis, and Anupam Prakash. Quantum reinforcement learning via policy iteration, 2022.

[53] Lukas Franken, Bogdan Georgiev, Sascha Mücke, Moritz Wolter, Raoul Heese, Christian Bauckhage, and Nico Piatkowski. Quantum circuit evolution on nisq devices. In *2022 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8. IEEE, 2022.

[54] Maxime Chevalier-Boisvert, Lucas Willems, and Suman Pal. *Minimalistic Gridworld Environment for OpenAI Gym*, 2018. Accessed 19 Nov. 2022.

[55] Hado van Hasselt, Arthur Guez, and David Silver. Deep Reinforcement Learning with Double Q-Learning. In *AAAI Conf. on Artificial Intelligence*, volume 30, 2016.

[56] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-Level Control through Deep Reinforcement Learning. *Nature*, 518(7540):529–533, 2015.

[57] Long-Ji Lin. Self-Improving Reactive Agents Based On Reinforcement Learning, Planning and Teaching. *Machine learning*, 8(3-4):293–321, 1992.

[58] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge Univ Press, 2010.

[59] Marco Cerezo, Andrew Arrasmith, Ryan Babbush, Simon C. Benjamin, Suguru Endo, Keisuke Fujii, Jarrod R. McClean, Kosuke Mitarai, Xiao Yuan, Lukasz Cincio, and Patrick J. Coles. Variational quantum algorithms. *Nat. Rev. Phys.*, 3(9):625–644, 2021.

[60] Jarrod R. McClean, Jonathan Romero, Ryan Babbush, and Alán Aspuru-Guzik. The Theory of Variational Hybrid Quantum-Classical Algorithms. *New J. Phys.*, 18(2):023023, 2016.

[61] Alberto Peruzzo, Jarrod McClean, Peter Shadbolt, Man-Hong Yung, Xiao-Qi Zhou, Peter J. Love, Alán Aspuru-Guzik, and Jeremy L. O'Brien. A Variational Eigenvalue Solver on a Photonic Quantum Processor. *Nat. Commun.*, 5(1):4213, 2014.

[62] Ishtiaq Rasool Khan and Ryoji Ohba. Closed-Form Expressions for the Finite Difference Approximations of First and Higher Derivatives Based on Taylor Series. *Journal of Computational and Applied Mathematics*, 107(2):179–193, 1999.

[63] Kosuke Mitarai, Makoto Negoro, Masahiro Kitagawa, and Keisuke Fujii. Quantum Circuit Learning. *Phys. Rev. A*, 98(3), 2018.

[64] Tyson Jones and Julien Gacon. Efficient calculation of gradients in classical simulations of variational quantum algorithms, 2020.

[65] Adrián Pérez-Salinas, Alba Cervera-Lierta, Elies Gil-Fuster, and José I. Latorre. Data Re-Uploading for a Universal Quantum Classifier. *Quantum*, 4:226, 2020.

[66] Maria Schuld, Ryan Sweke, and Johannes Jakob Meyer. Effect of Data Encoding on the Expressive Power of Variational Quantum-Machine-Learning Models. *Phys. Rev. A*, 103(3):032430, 2021.

[67] Erwin Coumans and Yunfei Bai. *PyBullet, a Python Module for Physics Simulation for Games, Robotics and Machine Learning*, 2016. Accessed 19 Nov. 2022.

[68] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafał Józefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*, 2015.

[69] 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.

[70] Sergio Guadarrama, Anoop Korattikara, Oscar Ramirez, Pablo Castro, Ethan Holly, Sam Fishman, Ke Wang, Ekaterina Gonina, Neal Wu, Efi Kokiopoulou, Luciano Sbaiz, Jamie Smith, Gábor Bartók, Jesse Berent, Chris Harris, Vincent Vanhoucke, and Eugene Brevdo. *TF-Agents: A library for Reinforcement Learning in TensorFlow*, 2018. Accessed 19 Nov. 2022.

[71] Cirq Developers. *Cirq*, 2022.

[72] Quantum AI team and collaborators. *Qsim*, 2020.

[73] Maja Franz, Lucas Wolf, Maniraman Periyasamy, Christian Ufrecht, Daniel D. Scherer, Axel Plinge, Christopher Mutschler, and Wolfgang Mauerer. Uncovering instabilities in variational-quantum deep q-networks. *Journal of the Franklin Institute*, 2022.

[74] Franz J. Schreiber, Jens Eisert, and Johannes Jakob Meyer. Classical surrogates for quantum learning models. *Phys. Rev. Lett.*, 131, 2023.

[75] Lennart Bittel and Martin Kliesch. Training variational quantum algorithms is np-hard. *Phys. Rev. Lett.*, 127(12):120502, 2021.

[76] Eric R. Anschuetz and Bobak T. Kiani. Quantum variational algorithms are swamped with traps. *Nature Communications*, 13(1):7760, 2022.

[77] Jarrod R. McClean, Sergio Boixo, Vadim N. Smelyanskiy, Ryan Babbush, and Hartmut Neven. Barren plateaus in quantum neural network training landscapes. *Nat. Commun.*, 9(1), 2018.

[78] Ryan LaRose and Brian Coyle. Robust data encodings for quantum classifiers. *Phys. Rev. A*, 102:032420, 2020.

[79] Samuel Yen-Chi Chen, Chih-Min Huang, Chia-Wei Hsing, Hsi-Sheng Goan, and Ying-Jer Kao. Variational quantum reinforcement learning via evolutionary optimization. *Machine Learning: Science and Technology*, 3(1):015025, 2022.

TABLE 6. Summary of all hyper-parameters used in the training of the four simulated robotic environments and the two OpenAI gym environments used for reference and comparison in App. A.

Hyper-Parameter	3×3	4×4	5×5	dynamic	FrozenLake	Cartpole-v1
Max. training steps	50,000	50,000	50,000	100,000	2,000	10,000
Max. steps per episode	200	200	200	200	80	500
Initial random steps	5,000	5,000	5,000	5,000	5,000	5,000
Eval. runs	10	10	10	10	10	100
Eval. every t steps	100	100	100	100	25	100
Eval. threshold	10.5	11	10	-/-	0.95	475
ϵ starting value	1.0	1.0	1.0	1.0	1.0	1.0
ϵ minimum	0.1	0.1	0.1	0.1	0.1	0.05
ϵ decay	0.99	0.99	0.99	0.99	0.9	0.95
Decay every t steps	250	250	250	500	50	100
Replay buffer size	20,000	20,000	20,000	20,000	20,000	20,000
Batch size	64	64	64	64	64	64
Target update	100	100	100	100	100	10
γ	0.99	0.99	0.99	0.99	0.99	0.99
Input learning rate	0.01	0.01	0.01	0.01	0.01	0.001
Circuit learning rate	0.001	0.001	0.001	0.001	0.001	0.005
Output learning rate	0.01	0.01	0.01	0.01	0.01	0.1
Classical learning rate	0.001	0.001	0.001	0.001	0.001	0.01

TABLE 7. Statistics on the experiments executed in the small 3×3 environment for all three architectures and their configurations. The best statistical values (mean, median, minimum, maximum, and standard deviation) in terms of number of training steps for each architecture are marked bold, the best overall configuration for each architecture is marked with a green background.

	NN		No. of training steps				
Units	$\lvert\theta_{NN}\rvert$	Solved	Mean	Median	Min	Max	Std.
(8;8)	131	17/20	29,390	26,800	18,200	50,000	9,558
(16;8)	227	20/20	21,075	20,500	13,700	33,200	4,921
(16;16)	387	20/20	16,405	16,800	10,500	21,900	3,049
(32;16)	707	20/20	14,620	14,650	8,900	20,000	2,853
(32;32)	1,283	20/20	12,625	12,400	8,700	17,000	2,143
(64;32)	2,435	20/20	10,635	10,250	7,000	14,100	2,231
(64;64)	4,611	19/20	12,790	10,100	6,700	50,000	10,200
(128;64)	8,963	20/20	11,535	9,700	6,800	47,600	8,414
(128;128)	17,411	20/20	8,885	8,650	5,800	11,900	1,746
(256;128)	34,307	20/20	**7,495**	**7,550**	**5,200**	**9,800**	**1,324**
	PQC-1		No. of training steps				
Layers	$\lvert\theta_{PQC}\rvert$	Solved	Mean	Median	Min	Max	Std.
12	156	20/20	24,670	25,200	13,400	31,800	4,034
15	192	20/20	18,620	18,800	9,000	24,000	3,402
18	228	20/20	19,345	19,250	14,900	24,000	**2,308**
21	264	20/20	15,905	16,900	5,700	22,600	4,600
24	300	20/20	15,350	14,750	10,000	20,700	3,136
27	336	20/20	11,365	11,800	4,200	17,000	3,692
30	372	20/20	10,245	11,100	4,100	15,800	3,383
33	408	20/20	10,220	10,450	**1,700**	20,200	4,427
36	444	20/20	**9,150**	**9,350**	2,000	16,500	3,746
39	480	20/20	10,060	10,050	4,800	13,900	2,905
	PQC-3		No. of training steps				
Layers	$\lvert\theta_{PQC}\rvert$	Solved	Mean	Median	Min	Max	Std.
8	156	20/20	20,270	20,000	14,300	26,300	3,608
10	192	20/20	14,930	16,750	3,900	22,900	5,231
12	228	20/20	16,530	15,850	7,100	25,000	4,155
14	264	20/20	10,995	11,750	2,600	22,100	5,188
16	300	20/20	9,300	8,750	1,200	20,600	4,175
18	336	20/20	10,385	9,450	2,600	19,300	4,553
20	372	20/20	10,390	10,400	1,800	18,000	4,164
22	408	20/20	9,490	8,400	2,300	20,300	5,181
24	444	20/20	**6,850**	**6,650**	**1,700**	**12,800**	**3,146**
26	480	20/20	9,515	8,800	4,400	15,300	3,347

TABLE 8. Statistics on the experiments executed in the medium sized 4×4 environment for all three architectures and their configurations. The best statistical values (mean, median, minimum, maximum, and standard deviation) in terms of number of training steps for each architecture are marked bold, the best overall configuration for each architecture is marked with a green background.

	NN		No. of training steps				
Units	$\lvert\theta_{NN}\rvert$	Solved	Mean	Median	Min	Max	Std.
(8;8)	131	16/20	37,315	37,050	19,600	50,000	9,329
(16;8)	227	17/20	35,910	34,650	23,700	50,000	7,477
(16;16)	387	17/20	24,565	23,350	900	50,000	13,387
(32;16)	707	17/20	28,060	25,100	7,200	50,000	11,359
(32;32)	1,283	16/20	24,315	19,800	7,200	50,000	13,548
(64;32)	2,435	20/20	15,055	16,550	6,300	28,300	5,543
(64;64)	4,611	20/20	14,740	15,050	6,900	22,200	4,205
(128;64)	8,963	19/20	18,370	15,000	5,400	50,000	11,092
(128;128)	17,411	20/20	12,270	**11,900**	5,700	20,000	3,930
(256;128)	34,307	20/20	**11,480**	12,000	**4,400**	**18,900**	**3,800**
	PQC-1		**No. of training steps**				
Layers	$\lvert\theta_{PQC}\rvert$	**Solved**	**Mean**	**Median**	**Min**	**Max**	**Std.**
12	156	18/20	41,395	42,400	28,700	50,000	5,260
15	192	20/20	29,905	31,700	9,300	46,100	9,254
18	228	20/20	27,055	30,950	2,400	39,500	11,105
21	264	20/20	24,815	26,750	4,000	36,000	8,303
24	300	20/20	27,795	27,250	22,200	36,000	**3,913**
27	336	20/20	22,355	24,400	11,800	31,700	6,131
30	372	20/20	20,560	21,600	4,800	30,700	6,542
33	408	20/20	20,135	20,750	4,400	29,900	5,640
36	444	20/20	17,705	18,800	7,000	**26,000**	4,678
39	480	20/20	**16,880**	**18,600**	**3,400**	29,300	7,288
	PQC-3		**No. of training steps**				
Layers	$\lvert\theta_{PQC}\rvert$	**Solved**	**Mean**	**Median**	**Min**	**Max**	**Std.**
8	156	20/20	30,155	32,900	8,100	41,300	9,654
10	192	20/20	25,990	28,700	8,800	39,900	8,469
12	228	20/20	21,525	23,800	3,500	31,900	8,074
14	264	20/20	19,695	20,750	2,700	29,900	7,103
16	300	20/20	20,170	21,950	4,300	28,300	5,984
18	336	20/20	19,015	20,950	4,400	26,700	5,891
20	372	20/20	18,795	19,700	6,000	24,500	**4,346**
22	408	20/20	17,065	17,150	5,500	26,600	5,652
24	444	20/20	**14,665**	15,350	**2,000**	**25,000**	6,068
26	480	20/20	15,875	**15,050**	7,200	25,200	5,164

TABLE 9. Statistics on the experiments executed in the large 5×5 environment for all three architectures and their configurations. The best statistical values (mean, median, minimum, maximum, and standard deviation) in terms of number of training steps for each architecture are marked bold, the best overall configuration for each architecture is marked with a green background. Configurations for which fewer than 15 runs succeeded are considered insufficient and are marked with a red background.

	NN		No. of training steps				
Units	$\lvert\theta_{NN}\rvert$	Solved	Mean	Median	Min	Max	Std.
(8;8)	131	1/20	49,825	50,000	46,500	50,000	762
(16;8)	227	4/20	49,515	50,000	42,500	50,000	1,641
(16;16)	387	8/20	46,570	50,000	32,400	50,000	5,104
(32;16)	707	15/20	42,040	41,350	29,700	50,000	6,291
(32;32)	1,283	17/20	40,055	40,900	23,300	50,000	8,465
(64;32)	2,435	18/20	32,135	30,850	22,000	50,000	7,845
(64;64)	4,611	20/20	29,475	28,450	23,300	40,400	4,218
(128;64)	8,963	20/20	25,540	24,950	19,500	**31,600**	**3,132**
(128;128)	17,411	20/20	22,870	22,100	**15,900**	35,900	4,301
(256;128)	34,307	20/20	**22,220**	**21,100**	17,700	33,500	4,017
	PQC-1		**No. of training steps**				
Layers	$\lvert\theta_{PQC}\rvert$	**Solved**	**Mean**	**Median**	**Min**	**Max**	**Std.**
12	156	4/20	48,005	50,000	31,400	50,000	4,644
15	192	12/20	46,390	47,700	36,500	50,000	4,021
18	228	18/20	40,410	39,850	25,900	50,000	6,182
21	264	20/20	37,375	34,500	29,400	49,700	6,546
24	300	20/20	35,190	33,600	27,400	47,200	**5,216**
27	336	19/20	35,685	32,900	19,300	50,000	7,881
30	372	20/20	30,995	31,400	8,500	46,600	9,123
33	408	20/20	28,155	27,900	19,100	38,500	5,752
36	444	20/20	29,260	29,300	15,800	47,000	6,614
39	480	20/20	**25,110**	**26,550**	**12,900**	**33,500**	5,309
	PQC-3		**No. of training steps**				
Layers	$\lvert\theta_{PQC}\rvert$	**Solved**	**Mean**	**Median**	**Min**	**Max**	**Std.**
8	156	13/20	43,925	45,700	32,100	50,000	6,303
10	192	18/20	41,645	42,500	28,800	50,000	6,744
12	228	20/20	35,675	35,000	27,100	47,600	6,372
14	264	20/20	33,885	33,400	22,400	45,200	5,778
16	300	20/20	32,625	33,700	23,000	43,300	5,588
18	336	20/20	30,765	30,300	22,600	42,200	5,212
20	372	20/20	29,895	31,250	20,500	39,300	**5,037**
22	408	20/20	24,530	24,900	12,800	**38,600**	5,762
24	444	20/20	27,575	28,650	**9,400**	38,800	7,334
26	480	20/20	**23,635**	**21,800**	9,700	41,600	7,535

HANS HOHENFELD received his B. Sc. and M. Sc. degrees in computer science from the FernUniversität in Hagen in 2014 and 2017. Before joining the Robotics Research Group at University of Bremen in 2017, he worked as software engineer since 2008. Currently he is pursuing a Ph. D. degree at the University of Bremen. His research interests include hardware efficient quantum algorithms for machine learning as well as quantum information and quantum algorithms in learning and decision problems for autonomous agents.

DIRK HEIMANN received his B. Sc. and M. Sc. degrees in physics from the Leibniz Universität Hannover in 2014 and 2017, respectively. Before joining the Robotics Research Group at University of Bremen in 2020, he worked as software consultant in the industry. He is currently pursuing a Ph. D. degree at the University of Bremen. His research interests include quantum algorithms for machine learning as well as quantum optimal control.

FELIX WIEBE received his B. Sc. and M. Sc. degrees in physics from the Georg August University in Göttingen in 2014 and 2018, respectively. In 2018, he joined the Mechanics and Control team of the German Research Center for Artificial Intelligence (DFKI) in the Robotics Innovation Center (RIC) in Bremen. Since 2020, he is pursuing a Ph. D. in the field of optimal control for underactuated robots. His research interests include robotics, optimal control, model predictive control, optimization algorithms, and quantum optimal control.

FRANK KIRCHNER received the degree in computer science and neurobiology and the Dr. rer. nat. degree in computer science from the University of Bonn, in 1994 and 1999, respectively. Since 1994, he has been a Senior Scientist with the Gesellschaft für Mathematik und Datenverarbeitung (GMD), Sankt Augustin. He has been a Senior Scientist with the Department for Electrical Engineering, Northeastern University, Boston, MA, USA, since 1998, where he was appointed as an Adjunct and then a Tenure Track Assistant Professor, in 1999. Since 2002, he has been a Full Professor with the University of Bremen. Since December 2005, he has been the Director of the Robotics Innovation Center (RIC), Bremen.