Quantum Reinforcement Learning for Sensor-Assisted Robot Navigation Tasks

LUND
UNIVERSITY

Author:	Joyce Gerarda Helena Cobussen
External Supervisor:	PD Dr. habil. Jeanette Miriam Lorenz
Internal Supervisor:	Assoc. Prof. Dr. Andreas Walther
Co-supervisor:	MSc. Theodora-Augustina Drăgan
Examiners:	Assoc. Prof. Dr. Claudio Verdozzi
	Assoc. Prof. Dr. Oxana Smirnova
Degree:	Master's Degree General Physics
Project Duration:	30 ECTS / 6 months
Submission:	Munich, October 4, 2023

Abstract

Quantum computing has advanced rapidly throughout the past decade, both from a hardware and software point of view. A variety of algorithms have been developed that are suitable for the current generation of quantum devices, which are referred to as noisy intermediate-scale quantum devices. Amongst them is the variational quantum algorithm, a hybrid quantum-classical algorithm that optimizes the parameters of a parameterized quantum circuit using optimization methods from classical machine learning. Previous research has shown that this approach requires fewer trainable parameters and fewer time steps to find a solution to certain problems compared to various classical machine learning algorithms. This is particularly interesting in the case of reinforcement learning, which is a powerful type of machine learning, although its algorithms are notoriously difficult to train. They often require many training steps to converge to a solution, leading to large computational costs. Therefore, this work investigates the effect of replacing the deep neural networks of the Deep Q-learning algorithm with various parameterized quantum circuits. For the first time, a simulated TurtleBot equipped with a LiDAR sensor is trained to move through three environments containing fixed obstacles, each with a different size and complexity. The influence of different configurations of input data on the performance of both classical and quantum models is investigated. Furthermore, the expressibility, entanglement capability and effective dimension of the quantum circuits were computed to investigate the correlation between these metrics and the performance of the quantum models. Finally, the effect of depolarizing noise on the performance of one of the quantum models was investigated. Results showed that in the smallest environment, the quantum algorithm had a higher chance to converge and required fewer time steps to do so compared to a classical network containing a similar number of trainable parameters. In addition, the best quantum model performed equally well compared to the largest classical model, even though the latter has an order of magnitude more trainable parameters. In general, the data encoding strategy had a strong impact on the performance of the quantum models. Furthermore, there seemed to be no strong correlation between the quantum metrics and the performance of the quantum models. Finally, depolarizing noise at the a relatively low error rate did not influence the performance of the tested model. These results constitute a useful and practical base for further research into training sensor-assisted agents using quantum reinforcement learning. Further investigation into the the performance of the models in larger and more complex environments, as well as their performance in non-static environments, is required. Additionally, the correlation between the quantum metrics, especially the normalized effective dimension, and the quantum model performances requires further analysis. Lastly, noise models incorporating two-qubit gate errors, as well as different types of noise, should be further investigated.

List of Abbreviations

Abbreviation	**Term**
SR	Success Rate
ASTC	Average (number of) Steps To Convergence
CNN	Classical Neural Network
DNN	Deep Neural Network
DQN	Deep Q-Network
ED	Effective Dimension
FIM	Fisher Information Matrix
LiDAR	Light Detection and Ranging
ML	Machine Learning
MSE	Mean Squared Error
NED	Normalized Effective Dimension
NISQ	Noisy Intermediate-Scale Quantum
PQC	Parameterized Quantum Circuit
QML	Quantum Machine Learning
QNN	Quantum Neural Network
QRL	Quantum Reinforcement Learning
RL	Reinforcement Learning
ASQ	Average Solution Quality
VQA	Variational Quantum Algorithm

Contents

1 Introduction

This work investigates the application of quantum reinforcement learning on a robot navigation task. Since the work presented here lies at the intersection of various fields of research, the introduction is structured as follows. Section 1 focuses on the fundamental principles behind quantum computing and the corresponding notations, followed by a brief overview of the current state of the art of quantum computing in Section 2. Section 3 introduces reinforcement learning as one of the three main paradigms in machine learning and discusses the specific algorithms used. Next, Section 4 builds on the previous sections to lay out the fundamentals of quantum reinforcement learning. Section 5 focuses on the specific methods used to conduct and evaluate the performance of the quantum reinforcement learning algorithm, and Section 6 provides a brief literature review of prior quantum reinforcement learning research.

1.1 Quantum Computing

1.1.1 Fundamentals

In a classical computer, the smallest piece of information is called a bit (derived from binary digit), which can either be 0 or 1. The value of a bit is retained in the hardware of the computer using electrical circuits of which the measured voltage indicates which of the two values is represented. Any piece of information stored on a classical computer is described by a sequence of bits, referred to as a bit string. Similarly, the most elementary piece of information on a quantum computer is called a quantum bit, or qubit, for short. In contrast to classical bits, however, qubits are two-level quantum systems that can exist in an infinite number of states, described by a complex two-dimensional unit state vector:

$$|\psi\rangle = a\,|\phi_1\rangle + b\,|\phi_2\rangle \tag{1}$$

where the vectors $|\phi_1\rangle$ and $|\phi_2\rangle$ are two arbitrary, orthonormal vectors that span the Hilbert space $\mathcal{H}\widetilde{=}\mathbb{C}^2$ and $a, b \in \mathbb{C}^2$ are complex numbers, referred to as the amplitudes of the quantum system. The probabilities of measuring the states $|\phi_1\rangle$ and $|\phi_2\rangle$ are $|a|^2$ and $|b|^2$, respectively. As the only two possible measurement outcomes are $|\phi_1\rangle$ and $|\phi_2\rangle$, the normalization condition $|a|^2 + |b|^2 = 1$ must hold. The inner product on $\mathcal{H}$ is defined as $\langle . \,|\, . \rangle$ and the orthonormality of the states imposes that $\langle \phi_m \,|\, \phi_n \rangle = \delta_{mn}$.

The state of a single qubit is usually denoted as:

$$|\psi\rangle = a\,|0\rangle + b\,|1\rangle \tag{2}$$

where the quantum states $|0\rangle := \left(\begin{smallmatrix}1\\0\end{smallmatrix}\right)$ and $|1\rangle := \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)$ highlight the analogy with classical bits [1]. The set $\{|0\rangle, |1\rangle\}$ is referred to as the computational basis.

There exists a convenient parameterization of Eq. 2 in terms of spherical coordinates. The state vector contains two complex numbers a and b, which corresponds to four real numbers. The normalization condition $|a|^2 + |b|^2 = 1$ reduces the four real numbers to three, and the fact that two state vectors that only differ by a global phase effectively represent the same physical system leaves two real numbers, which can be denoted as $\phi \in [0, 2\pi)$ and $\theta \in [0, \pi]$. They can be chosen such that $a = \cos\frac{\theta}{2}$ and $b = \sin\frac{\theta}{2}$. Then, the state vector can be rewritten as:

$$|\psi\rangle = \cos\frac{\theta}{2}|0\rangle + \sin\frac{\theta}{2}e^{i\phi}|1\rangle, \tag{3}$$

which can conveniently be visualized on the Bloch sphere, illustrated in Figure 1. The poles represent the two computational basis states, and any other state vector on the surface of the sphere represents a superposition of these two states. Superposition is one of the phenomena that makes quantum computing interesting and potentially advantageous for solving certain problems, as the number of states that can be represented grows as 2^n, where n describes the number of qubits. In a quantum computer, such superpositions are created by acting on the qubit with qubit gates, which are unitary operators acting on a qubit (single-qubit gates), or multiple qubits simultaneously (multi-qubit or n-qubit gates). A common single-qubit gate to create superposition is the so-called Hadamard gate:

$$H = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{4}$$

which acts on a single qubit as shown below:

$$\begin{aligned} H|0\rangle &= \frac{|0\rangle + |1\rangle}{\sqrt{2}} \\ H|1\rangle &= \frac{|0\rangle - |1\rangle}{\sqrt{2}} \end{aligned} \tag{5}$$

These two orthogonal states form the basis states of the Hadamard basis, also referred to as $|+\rangle$ and $|-\rangle$, respectively. In terms of Bloch sphere representation, they are the vectors pointing along the x-axis of the Bloch sphere.

Another phenomenon that sets quantum computers apart from their classical counterparts is entanglement. This refers to a situation in which two quantum systems are correlated in such a way, that they can no longer be described independently. This correlation remains intact over arbitrarily large distances. More formally, a pure state of multiple qubits is said to be entangled when it cannot be written as a tensor product of the individual qubit states, i.e., when a decomposition of the form $|\psi\rangle = |\phi_1\rangle \otimes |\phi_2\rangle \otimes \cdots \otimes |\phi_n\rangle$ is not possible. This implies that changing or measuring the state of one of the entangled qubits can directly alter the state of the others. For that reason, entanglement is a fundamental building block of many quantum

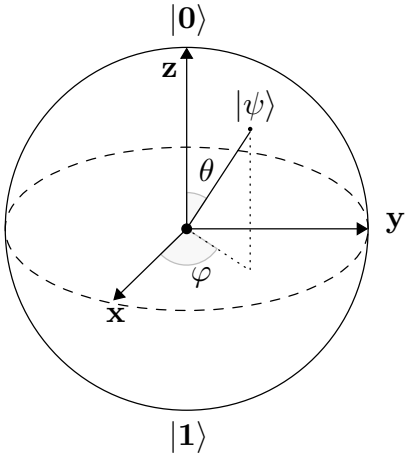


Figure 1: The Bloch sphere. Source: [2]

computing algorithms, for example Grover's algorithm for searching unsorted databases [3]. A common way of entangling two qubits is by applying a Hadamard gate on one qubit, referred to as the control qubit, followed by a CNOT-gate on both qubits. A CNOT-gate is a multi-qubit gate that flips the state of the target qubit(s) if the control qubit is in state $|1\rangle$ and leaves the target qubit(s) unchanged otherwise. The combination of a Hadamard and CNOT gate results in a maximally-entangled Bell state (Eq. 8), as shown below.

$$\text{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{6}$$

$$H\,|00\rangle = \left(\frac{|0\rangle + |1\rangle}{\sqrt{2}}\right)|0\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |10\rangle) \tag{7}$$

$$\text{CNOT}\frac{1}{\sqrt{2}}(|00\rangle + |10\rangle) = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) \tag{8}$$

This is merely one example of how qubits can be entangled. Many different ways exist, leading to different types of entanglement [4, 5]. An overview of commonly used quantum gates, their diagram symbols and matrix forms can be found in Appendix I.

1.2 Quantum Computing State of the Art

This section briefly turns to the physical implementation of quantum computers today, referred to as the noisy intermediate-scale quantum computing (NISQ) devices and the implications of the current status of the technology on the possible applications.

1.2.1 Quantum Computing Hardware

Research in quantum computing hardware progressed rapidly over the last decades. The various hardware implementations of quantum systems are based on different physical phenomena and have their own benefits and drawbacks. Currently, the most easily accessible type of quantum computing is superconducting quantum computing. This technology, provided by IBM [6] and Google [7], amongst others, relies on superconducting circuits, interrupted by Josephson junctions [1], to act as qubits at near-zero temperatures [9]. The quantum gates are microwaves that excite the qubits to higher energy levels. The benefits of this technology are the relatively high gate speed and fidelity, as well as the overlap with current microelectronic processing technology [10]. The cryogenic temperatures, however, make operation more difficult, and qubit connectivity is limited as the qubits are placed in a 2D lattice configuration.

Other types of quantum hardware leverage actual atoms to act as qubits, for example trapped ions. This technology is currently being developed by companies such as Quantinuum [11] and IonQ [12]. The qubits are laser-cooled ions trapped in electromagnetic fields, where the internal electronic states of the ion are leveraged as two-level systems. The quantum gates are implemented through laser pulses [13]. This leads to a technology with high-fidelity gates, long coherence times and no need for cryogenic cooling. However, these benefits come at the cost of relatively slow gate times and the need for an ultra-high vacuum [13].

Neutral atoms technology, currently being developed by Pasqal [14], amongst others, has similar (dis)advantages as trapped ion quantum computers, but uses neutral atoms trapped in optical lattices as qubits instead. The qubit gates are implemented through laser pulses. As the atoms have no charge, they can be moved closely together, which makes it relatively easy to scale the technology.

Another leading technology is photonic quantum computing, currently being developed by companies such as Xanadu [15] and PsiQuantum [16]. They use the properties of photons, such as their polarization, and make use of mirrors, beam splitters and phase shifters to manipulate them. As photons tend to hardly interact with their environment, no special environment such as cryogenics or vacuum are required for operation. However, this also means that two-qubit gates are challenging as photons do not interact directly, in addition to the noise induced by photon-loss [17].

[1] A simple superconducting circuit consists of a capacitor and an inductor, connected by superconducting wires. To create unevenly spaced energy levels of the circuit, the inductor is replaced by a Josephson junction, which consists of a thin piece of an insulating material placed between two superconducting metals. The Cooper pairs in the superconducting circuit can tunnel through the Josephson junction, and as a result, the energy levels of the superconducting circuit become unevenly spaced, allowing gate operations to be executed successfully [8].

The last technology worth mentioning is quantum computing based on solid-state quantum systems, such as NV-centers [2] [18] or defect centers including rare-earth ions [19]. Benefits of this technology include long coherence times and a high qubit density, as well as their capability to interact strongly with light. On the other hand, a drawback is the scalability, as qubit-qubit interactions are limited.

1.2.2 Noise

One of the biggest obstacles for current quantum computing technologies is the severe impact of noise. In this work, noise is defined as any undesired disturbance of the quantum system. There are multiple types of noise, depending on the physical phenomenon that generates it. For example, readout noise leads to a qubit to be measured in the $|1\rangle$ state, while actually being $|0\rangle$, and vice versa [20]. Another prominent source of noise are gate errors. Generally, gate errors can be divided into two categories, coherent and incoherent errors [20]. The latter is a stochastic type of noise, resulting from unwanted entanglement of the quantum system with the environment, and destroys the coherence of the quantum system, i.e., it maps pure states to mixed states. The latter is defined as a statistical ensemble of pure states. In terms of density operators, a mixed state can be written as $\rho_{mixed} = \sum_i p_i \left|\phi_i\right\rangle \left\langle\phi_i\right|$, in contrast to a pure state, which is defined as $\rho_{pure} = \left|\psi\right\rangle \left\langle\psi\right|$. Radiation, light, magnetic fields, sound vibrations etc. are all sources of incoherent noise. In contrast, coherent noise leaves the coherence of the quantum state intact, but introduces slight deviations from the ideal quantum state. For example, in the case of superconducting hardware, miscalibrations, unwanted qubit-qubit interactions and crosstalk are possible sources of coherent noise [21]. Evidently, noise is a complex topic and it would be beyond the scope of this work to do a full analysis on this matter. However, the average noise of large circuits that involve many qubits or many gate operations is fairly well described by depolarizing noise models [22]. Although this type of noise model does not cover coherent errors very well [22], the approximation can nevertheless provide first insights regarding the robustness of a quantum circuit.

1.2.3 Quantum Computing in the NISQ Era

Currently, no fault-tolerant quantum computers exist on any of the hardware platforms mentioned in section 1.2.1. Each implementation suffers, to some degree, from limited qubit coherence times, qubit and gate fidelities, scalability and connectivity, ultimately limiting the

[2] Nitrogen-Vacancy (NV) centers in diamond are atomic-scale defects where a nitrogen atom replaces a carbon atom in the crystal lattice, accompanied by a missing neighboring carbon atom. These defects exhibit electronic spin states that can be manipulated and measured using microwave and optical techniques. These spin states are highly sensitive to their local environment, particularly to magnetic fieldd, which allows for the execution of quantum gates [18].

size of the quantum computers in terms of number of logical qubits. Therefore, these NISQ devices are not suited for fault-tolerant algorithms, such as Shor's algorithm for factorization [23] or Grover's search algorithm [3]. However, that has not stopped research groups from finding algorithms that are NISQ-compatible, such as Variational Quantum Eigensolvers (VQE) [24], the quantum approximate optimization algorithm (QAOA) [25], and variational quantum algorithms (VQA) [26]. In addition, error-mitigation techniques have shown remarkable success in enhancing the performance of NISQ devices [27]. In view of these developments, this work focuses on quantum computing suitable for NISQ devices, which means the approach to the problem at hand should perform well with a limited number of qubits, and deliver desired results even in the presence of imperfect qubits and gate operations.

1.3 Reinforcement Learning

Together with supervised and unsupervised learning, reinforcement learning (RL) is the third main paradigm of machine learning (ML) [28]. The fundamental idea behind RL is to train an agent to make decisions based on interactions with its environment in such a way, that it maximizes a certain reward. One significant difference between RL and other ML paradigms is that the input data is not in a collected data set of the form $\mathbb{X} = \{x_i = f_1, f_2, \ldots, f_m, i \in \{1, 2, \ldots, n\}\}$, with n samples of m features each. Instead, a trial–and–error approach is used, through which the agent learns from its interactions with the environment [28]. The general algorithm follows the procedure outlined in Figure 2.

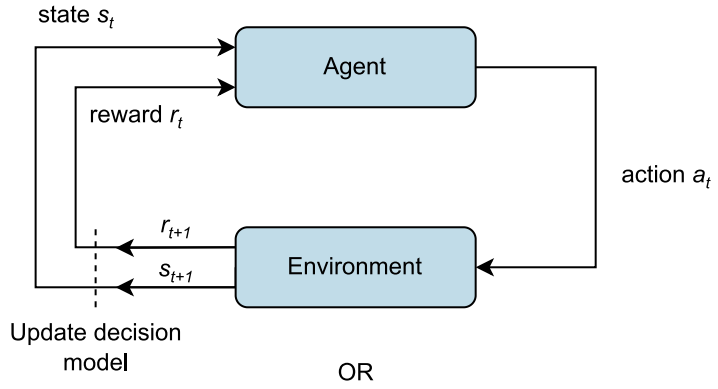


Figure 2: Schematic of the basic reinforcement learning process.

Initially, the agent is placed in an environment at state s_0. At each time step t, the environment is in state s_t, and the agent takes an action a_t. This action changes the environment relative to the agent. As a result, the environment outputs a reward r_{t+1} and then transitions into the new state s_{t+1}. The agent updates its decision model based on the reward and the new state it just observed. After this, the agent decides its next action a_{t+1} according to its policy, and the loop continues. The sequence of multiple time steps is referred to as an episode. The

episodes can theoretically be infinite, but in most implementations end either when the goal is reached, or when the agent exceeded the maximum number of steps it is allowed to take. After an episode ends, the agent and environment are reset to their original state and a new episode starts. This process is repeated until the agent successfully learns to take the best action in any state in order to reach the desired result [28].

1.3.1 Q-learning

There exist many different types of RL algorithms. This work uses Q-learning, which is a model-free, value based approach. Model-free means that the model does not have any prior knowledge about the environment. In computer science jargon, the model does not compute the transition probability for the world to go from one state to the next, based on the action taken. Instead, the model must experience the outcome of all actions purely through trial-and-error. Value-based refers to the quantity that the model aims to estimate, namely the quality-value, or Q-value for short, which is discussed in the next paragraph.

At the core of Q-learning lies the Q-function, which aims to estimate the Q-value. This quantity represents the expected cumulative reward that the agent can collect when it takes an action a_t in a state s_t and follows an epsilon-greedy policy, where R is a uniformly distributed random number between [0, 1]:

$$a_t = \begin{cases} \max(Q(s, a : \theta), & \text{if R} > \epsilon. \\ \text{Random action}, & \text{otherwise.} \end{cases} \tag{9}$$

In the simplest case, the Q-function is updated after every interaction with the environment, based on the obtained reward and the loss, which describes the difference between the predicted Q-value and the actual Q-value obtained at the next step. For discrete and relatively small environments, the Q-function can be realized in the form of a lookup table, or Q-Table, in which all possible combinations of actions and states are listed. Typically, the Q-values are initialized as fixed values, or as values drawn from a random distribution. Through each interaction with the environment, the agent tries to learn the true Q-values for each of these state-action pairs by updating the initial Q-values using equation 10, which is based on the Bellman equation [29]:

$$Q_{t+1}(s, a) = Q_t(s_t, a_t) + \alpha(r_t + \gamma(\max(Q_t(s_{t+1}, a_{t+1})) - Q_t(s_t, a_t)) \tag{10}$$

Here, $Q_t(s, a)$ is the Q-value for taking action a at state s, α is the learning rate, r represents the immediate reward that is acquired after taking action a in state s, γ is the discount factor, and $\max(Q_t(s_{t+1}, a_{t+1}))$ is the maximum Q-value that can be acquired in the next state, which is a prediction made by the model. For example, in the case of a Q-table, the term $\max(Q_t(s_{t+1}, a_{t+1}))$ can be acquired by searching through the table entries corresponding to the

state-action pairs (s_{t+1}, a_{t+1}) and simply selecting the pair with the highest Q-value. Hence, the function updates the current Q-value with the sum of the reward and the highest possible Q-value obtainable in the next state, where the latter is multiplied by the discount factor that decides the importance of future Q-values relative to the immediate reward. To summarize, the Q-function can be understood as the memory of the agent, enabling it to keep track of past interactions and to predict the value of future interactions. As it is a cumulative, discounted and weighted sum of previous rewards, the Q-value predicted by the Q-function is always an approximation to the true Q-value, which remains unknown. However, the approximations approach the true Q-value over time, leading the agent to find a solution to the problem.

1.3.2 Deep Q-network Algorithm

The Q-table is arguably the simplest and perhaps the most transparent form of the Q-function. However, for problems with large or continuous state-spaces, this approach is clearly not suitable, as the Q-table would become infinitely large. Instead, one can use a function approximator to estimate the Q-function. In this case, the Q-function is a parameterized function that takes the current state s_t as an input and predicts the expected reward for each of the possible actions. Typically, the function approximator is a deep neural network (DNN), in which case the algorithm is referred to as a Deep-Q Network (DQN). DQN was first developed by DeepMind in 2015 [30], initially to play Atari 2600. Since then, DQN has found a wide range of real-life applications, such as autonomous driving [31], treatment and diagnosis pathways of various diseases [32], vehicle navigation and planning [33, 34], and robotics [35, 36, 37]. However, despite their wide variety of applications, it should be noted that the success of RL algorithms highly depends on the settings of their hyperparameters, defined as all non-trainable parameters that are set by the user, such as the learning rate, reward, discount factor, and others. This has shown to make deep reinforcement learning algorithms notoriously difficult, and thus computationally expensive, to train [38].

The pseudocode for the DQN algorithm is shown in Algorithm 1. Although its overall structure is similar to that of traditional Q-learning, there are some noteworthy differences. First of all, the DQN does not store nor update any Q-values explicitly. Instead, the DNN predicts the Q-values for each of the available actions, given a certain observation. Once a prediction has been made, the network either selects the action with the highest Q-value with a probability ϵ, which is referred to as the exploration rate, or it selects an action at random with probability $1 - \epsilon$. The exploration rate is a hyperparameter that is usually set to 1 at the beginning of the training and decreases linearly or exponentially throughout the training process. This way, the agent explores its environment at the start of a training by taking actions that it might deem sub-optimal. As time passes, the exploration of the environment decreases and the agent starts exploiting the environment instead, gaining higher rewards by selecting actions based on

previous experiences.

Analogous to Q-learning, the goal of Deep Q-learning is finding parameters that optimize the prediction of the Q-values, such that the policy, which is selecting the action with the highest Q-value at each time step, is optimal. To this end, the parameters of the deep neural network, also referred to as the online network, are updated every n time steps using a specified optimization algorithm. The update requires calculating the gradient of the loss function, in this algorithm defined as the mean squared error (MSE) between the Q-value predictions of the network $Q(\phi_j, a_j; \theta)$ and the 'true' Q-values, y_j:

$$L = (y_i - Q(\phi_j, a_j; \theta))^2 \tag{11}$$

The latter is written between quotation marks because the DQN algorithm does not have access to any labeled data, which means that there are no true Q-values to compare the prediction with. Instead, an approximation of the true Q-value is based on the Bellmann equation (10),

$$y_i := r_t + \gamma \max Q(s_{t+1}, a; \theta^-) \tag{12}$$

where the Q-value $Q(s_{t+1}, a, \theta^-)$ is provided by a second network, also referred to as the target network $\hat{Q}$ with parameters θ^-. The target network is essentially a copy of Q, initialized with the same random weights and producing predictions based on the same samples whenever the online network Q is updated. However, the parameters of $\hat{Q}$ are not updated via gradient descent; rather, the parameters of Q are copied to $\hat{Q}$ every mn number of steps, with $m > n$. This mechanism ensures stabilization of the training process, regulated by the choice of m.

Another important characteristic of the DQN algorithm is that the Q-values are predicted based on so-called transitions. These transitions are small subsets of data that describe a single interaction between the agent and the environment. They contain a state ϕ_t, corresponding to the state ϕ at time step t, the action a_t taken at this time step, the reward r_t obtained as a result from this action, and the next state $\phi_t + 1$. Each transaction is stored in the so-called replay memory D. At every n steps, a fixed number of transitions, also referred to as a mini-batch, is uniformly sampled from D and used to update the network. Using a mini-batch rather than single transition ensures a more efficient use of training experiences and avoids learning unwanted correlations between consecutive steps [30]. D has a limited capacity, so when the maximum capacity has been reached, the oldest transitions are discarded to free up space for the latest transitions.

1.4 Quantum Machine Learning

As mentioned in previous sections, quantum computing has been a fast-growing field of research for several years, both on the hardware and algorithmic side. At present, fault-tolerant

Algorithm 1 Deep Q-learning with experience replay

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights θ

Initialize target action-value function $\hat{Q}$ with random weights $\theta^- = \theta$

For episode=1, M **do**

Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$

For $t = 1$, T **do**

With probability ϵ select a random action a_t

otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$

Execute action a_t in the emulator and observe reward r_t and observation x_{t+1}

Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$

Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D

Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D

$$\text{Set } y_j = \begin{cases} r_j, & \text{if episode terminates at step j+1} \\ r_j + \gamma \text{max}_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$$

Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to θ

Every C steps reset $\hat{Q} = Q$

End For

End For

quantum computers with sufficient logical qubits to fully support algorithms such as Grover's algorithm or the HHL algorithm on are not yet available [39]. Instead, the current generation of quantum computers, referred to as noisy intermediate-scale quantum (NISQ) generation, contains relatively few qubits (in the order of $10^2 - 10^3$ physical qubits) and is subject to noisy operations and limited coherence times. However, this has not stopped both research groups and industry players from finding potential applications for NISQ devices in various fields of research [40]. Quantum machine learning (QML) is one of the research directions that has received substantial attention [41]. In this hybrid quantum-classical approach, a NISQ device is used to run a variational quantum algorithm (VQA), which replaces the mechanisms of the deep neural network within a classical machine learning algorithm. In case of the DQN algorithm, this means that the deep neural network is replaced by a quantum circuit. In the following subsections, the mechanisms behind VQA will be explained and its potential benefits in the context of quantum reinforcement learning will be discussed.

1.4.1 Variational Quantum Algorithms

At the core of a VQA lies its parameterized quantum circuit (PQC), which is the quantum analogue of a classical deep neural network (DNN). A PQC is essentially a sequence of quantum gates $U(\theta, x)$, applied to the computational basis state $|0\rangle^{\otimes n}$ to obtain a final state $|\psi\rangle^{\otimes n}$ that represents the solution to the problem [42]. A schematic overview of a PQC is shown in Figure 3. Generally, the first step is to encode the classical input data onto the quantum circuit. To this end, a set of parameterized gates $U(x)$, where x represents the input data, is applied to a set of qubits that are initialized in the ground state $|0\rangle^{\otimes n}$. For example, in case of angle encoding, the input data is first transformed such that it lies within $[\frac{\pi}{2}, -\frac{\pi}{2}]$. Afterwards, the transformed values are used as input parameters for rotation gates, which encode the transformed data onto the qubits. Next, a trainable layer consisting of several trainable, parameterized, single-qubit gates $U(\theta)$ acts on the qubits, where the parameters θ can be considered as the quantum analogue of the trainable weights in a classical neural network. This layer also includes a qubit entangling scheme. Typically, the encoding and trainable layers are executed several times in alternating fashion, which is referred to as data re-uploading (Figure 3). This technique allows the quantum model, which can be written as a partial Fourier series in the data, to access an increasingly rich frequency of spectra [43] – in other words, it increases the ability of the circuit to approximate any function [44]. Previous experiments have indeed shown data re-uploading to enhance the performance of the algorithm [45].

After executing the quantum circuit, one measures the expectation value of a certain observable, often Pauli-Z, for each of the qubits. The measured outcomes represent the solutions that the PQC predicts. In the case of the DQN algorithm, the expectation values represent the Q-values for each of the available actions. It should be noted that while $\langle \mathrm{Z} \rangle$ takes values between -1 and 1, the problem at hand may deal with Q-values that fall outside this range. It is therefore necessary to post-process the expectation values, either by scaling with a constant, or by implementing a small, fully connected neural network that learns the proper scaling factors [46].

Finally, the post-processed predictions of the PQC are used to evaluate the loss and update the parameters of the PQC using a classical computer. To update the circuit parameters using gradient descent, the gradients of the PQC with respect to its trainable parameters are required. The parameter-shift rule is arguably the simplest method to obtain these gradients [47]. In short, this technique evaluates the loss function at two shifted parameter positions of every parameterized gate, at every training step. The re-scaled difference between the results forms an unbiased estimate of the derivative. Using this derivative and the classical evaluation of the loss function, the gate parameters can be updated, for example via gradient descent [48], in order to minimize the loss. The whole process of encoding the input data, executing the quantum circuit, measuring and classically updating the parameters is repeated until the quantum model

has minimized the loss.

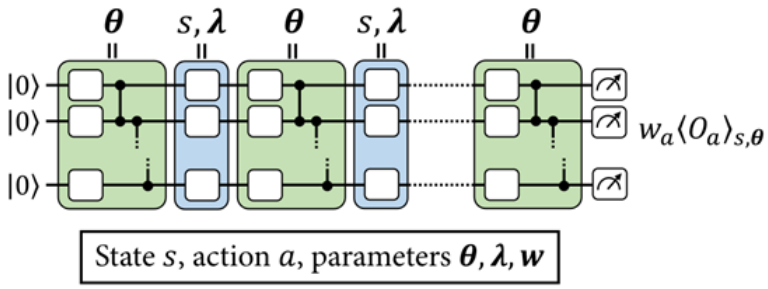


Figure 3: Schematic of the variational quantum circuit. The blue layers encode the data, and the green layers contain the trainable parameters. Source: [49]

1.5 Quantum Circuit Metrics

In the same way that the architecture of a classical neural network is crucial to the performance of any machine learning algorithm, the design of the PQC is known to have a significant impact on the success of a VQA [50]. There are many factors to consider when constructing a PQC, such as the number of qubits, the number and types of gates, the type of data encoding, and the number of layers (circuit depth), which makes it a complex task. Therefore, various metrics have been developed that aim to indicate which circuit architecture could perform better than others. The quantum circuit metrics that will be discussed in this work are the expressibility, the entangling capability and the normalized effective dimension. It should be noted, however, that although these metrics provide a way of comparing different circuit designs, they do not guarantee good solutions and the exact correlation between the metrics and performance of a particular PQC architecture seems to depend on the problem at hand [51, 52].

1.5.1 Expressibility

Expressibility refers to the ability of a quantum circuit to produce states that cover the entirety of the Hilbert space. In the case of a single qubit, this refers to the capability of the qubit to explore the Bloch sphere. To quantify the expressibility of any parameterized quantum circuit, one can compare the distribution of state fidelities obtained from executing the PQC with randomly sampled parameters to a uniform distribution of state fidelities, i.e., state fidelities obtained from an ensemble of Haar-random states [53]. The expressibility can then be defined as the Kullback–Leibler (KL) divergence between the two distributions:

$$Expressibility = D_{KL}(\hat{P}_{PQC}(F;\theta)||P_{Haar}(F)) \tag{13}$$

where $\hat{P}_{PQC}(F;\theta)$ corresponds to the estimated probability density function (PDF) of the fidelity F of the states produced by randomly sampling parameters θ, and $P_{Haar}(F)$ indicates

the PDF of the fidelities of the Haar random states. The analytical form of the latter is known: $P_{Haar}(F) = (N-1)(1-F)^{N-2}$, where N is the dimension of the Hilbert space [53]. A smaller KL divergence indicates a smaller deviation from the fidelity distribution of the Haar random states, and thus a higher expressibility. As the experiments have a finite sample size, the probability distributions in question are approximated with histograms. As a result, the number of bins influences the upper bound of the expressibility as follows: for the least expressive case, where a circuit always outputs the same state (e.g., a circuit containing only the identity gate I), the upper limit for the expressibility is given by $(N-1)\ln(n_{bin})$ [53]. It is worth noting that while expressibility by itself does not constitute a definitive measure for the suitability of a quantum circuit, it may still be used to rule out circuit designs that have (relatively) poor expressibility.

1.5.2 Entangling Capability

The entangling capability of a circuit describes its ability to produce entangled states, which has shown to be advantageous in the context capturing non-trivial correlations in the input data [53]. In addition, a high entangling capability has shown to increase its ability to represent the solution space for various problems, for example classification problems [54].

One way of quantifying the entangling capability of a quantum circuit is to calculate the Meyer-Wallach entanglement measure [55]. This global measure is defined as the average of the bipartite entanglement of one qubit with all others, measured by the purity. In mathematical form, the measure is defined as follows. Consider a linear mapping:

$$\imath_j(b)\left|b_1...b_n\right\rangle = \delta_{bb_j}\left|b_1...\hat{b}_j...b_n\right\rangle \tag{14}$$

where $|b_1...b_n\rangle$ refers to the computational basis of n qubits with $b_j \in \{0,1\}$ and $\hat{}$ denotes the absence of the j-th qubit. The Meyer-Wallach entanglement measure Q is defined as:

$$Q(|\psi\rangle) \equiv \frac{4}{n}\sum_{j=1}^{n} D(\imath_j(0)\left|\psi\right\rangle, \imath_j(1)\left|\psi\right\rangle) \tag{15}$$

where the argument of the sum is given by

$$D(|u\rangle, |v\rangle) = \frac{1}{2}\sum_{i,j}|u_i v_j - u_j v_i|^2 \tag{16}$$

One property of this measure is that $0 \leq Q \leq 1$ and $Q(|\psi\rangle) = 0$ if, and only if, $|\psi\rangle$ is a product state. For example, $Q(|01\rangle) = 0$, whereas $Q(\frac{|00\rangle+|11\rangle}{\sqrt{2}}) = 1$. For any given PQC, the entanglement capability can be computed by randomly sampling the circuit parameters and calculating the average of the measure of the resulting states. As a result, a circuit that

outputs highly entangled states will have a Q close to 1, while a circuit that exclusively produces product states will have Q equal to 0 [53].

1.5.3 Normalized Effective Dimension

The third metric that is analyzed in this work is the effective dimension (ED), which describes how effectively a (quantum) neural network utilizes its parameter space to learn relevant links and patterns that are present in the data [56]. The ED is computed using the Fisher information matrix (FIM). In the context of machine learning, the FIM describes how sensitive the predictions of a neural network are to changes in its parameters, given a certain input. Statistically speaking, the joint relationship between input and output pairs (x, y) can be described as $p(x, y; \theta) = p(y|x; \theta)p(x)$, where $p(x)$ is a prior distribution comprised of the input data, and $p(y|x; \theta)$ a conditional distribution that describes the relation between the input x and output y of the model, given a fixed set of model parameters $\theta \in \Theta$. Given these definitions, the FIM can be calculated as follows:

$$F(\theta) = \mathbb{E}_{(x,y)\sim p}\left[\frac{\partial}{\partial\theta}\log p(x, y; \theta)\frac{\partial}{\partial\theta}\log p(x, y; \theta)^\intercal\right] \tag{17}$$

In practice, however, the exact distribution over the inputs $p(x)$ is not usually known. Instead, one can sample independent and identically distributed (i.i.d.) random variables from an experimentally obtained data distribution $p(x, y; \theta)$, which leads to the definition of the *empirical* Fisher information matrix [57]:

$$\tilde{F}_k(\theta) = \frac{1}{k}\sum_{j=1}^{k}\frac{\partial}{\partial\theta}\log p(x_j, y_j; \theta)\frac{\partial}{\partial\theta}\log p(x_j, y_j; \theta)^\intercal \tag{18}$$

Before moving on to the calculation of the ED, it is worth noting that the FIM itself holds valuable information about the trainability of a neural network. The parameter space of a machine learning model is often flat in most dimensions, except for a few that show clear distortions [58]. This implies that the value of most parameters does not heavily contribute towards finding the global minimum of the loss. Additionally, it means that once a model finds itself in a flat section of the parameter space, it can be difficult, if not impossible, to find a minimum at all, as the gradients of the loss landscape vanish. This is referred to as the barren plateau phenomenon [56]. The FIM reflects this phenomenon through the distribution of its eigenvalues. Large eigenvalues correspond to strong distortions in the loss landscape, whereas relatively flat dimensions are characterized by eigenvalues very close to 0. Hence, having a spectrum that is not entirely concentrated near zero may indicate lower risk of barren plateaus, which is beneficial for training [56].

The effective dimension, as it is defined in [56], incorporates the empirical FIM by integrating over its determinant. This way, the ED estimates the size that a model $\mathcal{M}_\Theta$ occupies in the

space of all possible functions for a given class of models, also referred to as the model space, where the empirical FIM acts as the metric. For the derivation of the formula for the ED, the interested reader may refer to [59]. The definition of the effective dimension presented in [56] is as follows:

$$d_{\gamma,n}(\mathcal{M}_\Theta) := 2\frac{\log\left(\frac{1}{V_\Theta}\int_\Theta \sqrt{\det(Id_d + \frac{\gamma n}{2\pi\log n}\hat{F}(\theta))}d\theta\right)}{\log(\frac{\gamma n}{2\pi\log n})}, \tag{19}$$

where $\mathcal{M}_\Theta := \{p(.,.,;\theta) : \theta \in \Theta\}$ refers to the neural network, here defined as a statistical model with respect to the the $d-$dimensional parameters space $\Theta \subset \mathbb{R}^d$ and data samples $n \in N, n > 1$. Furthermore, $V_\Theta := \int_\Theta d\theta \in \mathbb{R}_+$ refers to the volume of the parameter space and $\hat{F}(\theta)$, which is the normalized FIM defined as:

$$\hat{F}_{ij}(\theta) := d\frac{V_\Theta}{\int_\Theta \mathrm{tr}(F(\theta))d\theta}F_{ij}(\theta) \tag{20}$$

The purpose of the normalization is to ensure that $\frac{1}{V_\Theta}\int_\Theta \mathrm{tr}(\hat{F}(\theta))d\theta = d$, such that the ED is scaling invariant in θ [56]. To ease interpretation, the ED can be normalized by dividing it by the parameter space d. The normalized effective dimension (NED) has values between 0 and 1. If the neural network efficiently uses its capacity to learn features and patterns from the data, i.e., if most of its parameters contribute meaningfully to the network's predictions, the NED will be close to 1. Conversely, a network with an inefficient capacity usage will have a NED close to 0. The latter may indicate that the network is over-parameterized, which may come with a higher risk of overfitting [60]. Furthermore, as the NED directly depends on the number of samples n available to the network, it is a useful tool to see how the NED behaves for different data set sizes.

1.6 Current Status of Quantum Reinforcement Learning

Using VQAs in the context of quantum reinforcement learning (QRL) is a relatively new area of research with many open questions. Previous research has mainly focused on benchmarking QRL performance through Atari games such as Frozen Lake and Cartpole, which is common practice in reinforcement learning [61]. This has shown that QRL algorithms can produce the same results as classical RL algorithms using a comparable number of, or even fewer, trainable parameters [62, 46, 63, 64, 65]. In addition, lower memory consumption has been observed [66], as well as faster convergence [67, 68, 64]. However, it is poorly understood how the choice of quantum circuit architecture affects the learning behavior and overall success of the QRL algorithm. Attempts at understanding have been made in [62], where circuit metrics such as expressibility, entanglement capability and normalised effective dimension have been investigated, but there seemed to be little correlation. The authors of [46] observe that

the increasing the number of trainable parameters in the PQC only improves performance to a certain point, after which the performance stagnates or even decreases. Additionally, they note that hyperparameter tuning and the data encoding strategy have a considerably greater influence on the performance of the algorithm. Furthermore, there is a significant distinction between performances of certain architectures on simulations and real hardware. Each hardware technology comes with its own set of native gates and connectivity, which means that some architectures may need to be converted to fit these requirements, potentially leading to significant overhead in terms of circuit depth. In addition, NISQ-hardware is subject to various types of noise. Therefore, shallow architectures with a limited number total gates, two-qubit gates, and depth, are more likely to be successful on real hardware [69]. This also means that strategies such as data re-uploading may prove less advantageous when implemented on real hardware than simulations have shown in e.g., [45]. Moreover, hardly any research has moved from benchmarking problems to real-life applications. A few examples exists, such as the application of QRL to energy management scenarios [70], hedging problems [71], and medical decision-making [72], but the overall number of investigated applications is limited.

2 Problem statement

2.1 Background Information

This work aims to investigate the potential advantages of using quantum neural networks embedded in reinforcement learning algorithms in the framework of a simple robot navigation task. This learning environment is more complex than most of the Atari games that have been investigated so far and allows the problem size to be scaled up nearly arbitrarily. This use-case is inspired by the work of [42]. The authors simulate the agent, a TurtleBot, in three different environments of different sizes and containing different objects. The objective is to train the robot to walk from its starting position to the goal located diagonally across the environment, using a quantum DQN. The input to the PQC are the x and y coordinates of the robot, as well as its z-orientation, which are first transformed to be within the range $[\frac{1}{2}\pi, -\frac{1}{2}\pi]$ using the function $\theta_{x_i} = \arctan(x_i)$. The measured expectation value $\langle \mathrm{Z} \rangle$ of each qubit corresponds to the Q-value obtained by taking one of the three actions straight, left, or right, respectively. The structure of the PQC used in the paper follows the general structure including data-re-uploading as shown in Figure 3, and can be summarized as follows:

$$U(\theta, x) = \prod_{l=1}^{L} (U_{ent} U_{par}(\theta_l) U_{in}(x_l)) U_{ent} U_{par}(\theta(0)) \tag{21}$$

Here, U_{ent} signifies the entangling operations, U_{par} the layer of trainable parameterized gates and $U_{in}(x_l)$ corresponds to the parameterized gates that are used to encode the input data. The authors employ two variations of this quantum circuit for the PQC. The main difference between them is the way the classical data is embedded in the circuit. The first circuit, PQC-1, uses a single rotation gate $U_x(\theta)$ to encode each of the three inputs, whereas the second circuit, PQC-3, encodes the three elements of the input data on each of the qubits using the rotation gate $U_{rot}(\theta_x, \theta_y, \theta_z)$. The results showed that the QRL achieved results comparable to the classical RL model, while using fewer trainable parameters. In addition, the QRL model managed to successfully train the agent using fewer training steps than the classical models for the larger 4x4 and 5x5 environments. Therefore, one of the suggestions of the paper is further research into the scaling behavior of QRL algorithms for similar tasks, as increased problem size and complexity may reveal the effect and potential benefits of using a PQC to substitute the classical DNN more clearly.

2.2 Research Objectives

The research objective of this work is to see how a QRL algorithm similar to the one used by Heimann et al. [42] performs under larger and more complex problem conditions. To increase the complexity of the problem, the implementation of the TurtleBot will be made more realistic

by adhering to its realistic maximum speed, which implies that the agent will need to learn more steps in order to reach the goal. In addition, an artificial LiDAR sensor will be introduced to provide the distance to the nearest objects in various directions. This data will then be treated as additional input data. The performance of the QRL algorithm will be assessed through comparison with classical models that are arguably comparable in size with respect to the number of parameters. In addition, the expressibility, entanglement capability and normalized effective dimension of the various quantum circuits will be investigated to see whether any correlations between those measures and the model's performance can be observed for this problem. Lastly, a simple depolarizing noise model will be introduced to the sensorless model to investigate its sensitivity to noise.

2.3 Motivation and Contribution

As discussed in Section 1.6, most research so far has focused on benchmarking Atari games and similar environments. These problems and their solutions have been studied over the past decades using classical machine learning techniques, and the available knowledge about nearly all aspects of these problems has made the Atari games a great candidate for benchmarking QRL algorithms in their early stages. Throughout the past decade, research has shown that QRL algorithms yield similar or even better results, such as faster convergence to solutions and requiring fewer trainable parameters, compared to classical RL algorithms. However, it is still unknown whether these benefits hold true for more complex and, more importantly, more realistic problems.

Given this status quo of the field, the motivation for investigating quantum reinforcement learning in the context of robot navigation tasks is two-fold. Firstly, it provides a more complex use-case that may answer questions regarding the scalability of the potential benefits of QRL to more complex problems, as well as the impact of quantum circuit architectures. The investigation of various quantum circuit architectures may provide tangible experimental observations that can be compared to classical results, whereas the analysis of the corresponding quantum metrics (Section 1.5) and architectural patterns of the quantum circuits may provide a better theoretical understanding of how quantum circuit architectures influence the behavior of the agent in more complex settings. Hence, this work contributes to filling the existing knowledge gap in using QRL for more complex problems from both a theoretical and experimental point of view.

Secondly, the specific context of sensor-assisted robot navigation problems enables the application of QRL to a problem that is currently highly relevant in the real world. Robot navigation is a broad field of research and finds applications in transportation, industry, automatization,

and rescue robots [73, 74]. The complexity of the field is evident from the vast number of aspects that should be considered when tackling a problem. Each navigation task can be characterized as online, where the robot has access to sensors to perceive its environment, or as offline, in which case the robot has access to a full model of the room without the ability to sense its surroundings. In addition, one can consider static navigation, which implies that all objects remain in place, as opposed to dynamic navigation, which deals with moving objects and targets [73]. Furthermore, the navigation of the robot can be optimized against many different parameters, such as the path length, path smoothness, object avoidance, etc.. Considering these aspects, the robot navigation problem studied in this work falls into the category of static, online navigation problems, optimized against path length and object avoidance. This type of problem translates to applications such as warehouse navigation, which is a vital aspect of automatizing tasks such as packing, sorting, cleaning, surveiling etc. [75]. In this context, the structure of the environment and the main obstacles inside (counters, elevated floors, scaffoldings, doors) are mostly static, but as some smaller objects may obstruct the way (boxes, forgotten equipment), sensors are still necessary to avoid obstacle collisions. Such a problem has previously been studied using classical reinforcement learning in e.g., [76] and [77], so this work will contribute to the field by introducing quantum reinforcement learning as a novel approach to solving the problem. Exploring this approach may unveil its benefits or drawbacks, both of which would be valuable knowledge for the research community as well as the industry.

There is, however, one important caveat when using quantum reinforcement learning for robot navigation tasks: the trained quantum model should somehow be accessible to the robot once it is deployed in real life. An intuitive idea could be to equip the robot with a small, on-board quantum computer, but this is nearly impossible to achieve with the currently available technology (Section 1.2.1). Alternatively, a more realistic solution could to establish a reliable cloud connection between a classical on-board computer and a remote quantum computer, although this comes with various challenges as well [78]. Unfortunately, it is beyond the scope of this work dive into these practical aspects. Nevertheless, the aim is to explore the potential benefits of QRL in the first place and pave the first steps of the way towards implementation.

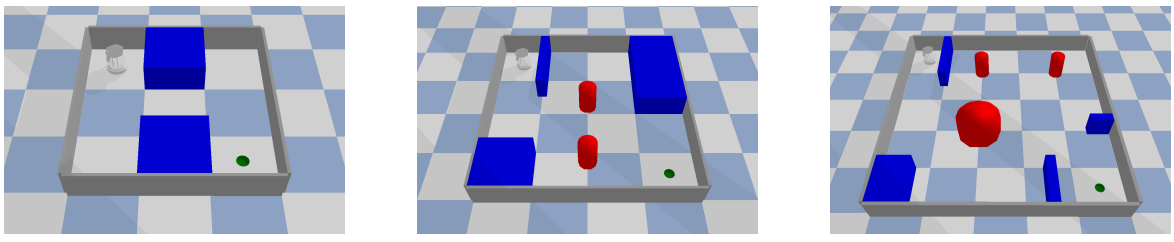
3 Methods

3.1 Simulation

3.1.1 Agent and Environment

The agent and its training environment are simulated using PyBullet, an open-source Python library that is widely used for robotics simulations and smoothly integrates with reinforcement learning tasks [79]. This library is based on Bullet Physics SDK, a physics engine that simulates collision detection as well as soft and rigid body dynamics [80]. In line with the paper by Heimann et al. [42], the agent is a simulated TurtleBot Burger based on a publicly available Unified Robotics Description Format (URDF) file. The TurtleBot can be moved by controlling the velocities of its two wheels. Setting the same wheel velocity for both wheels in the same direction leads to simple forward motion, whereas the same wheel velocity in opposite directions leads to a turn around the Turtlebot's axis. The maximum velocity of the TurtleBot is 0.22 m/s.

There are three different environments of size 3x3, 4x4, and 5x5 squared meters, depicted in Figure 4. Given the maximum speed of the TurtleBot, it takes at least around 45, 55 and 70 time steps to reach the goal, respectively. The objects have fixed base positions, which means that they do not move in case of a collision with the TurtleBot. This was implemented to avoid unwanted interference of objects with the TurtleBot during the training process. Both the TurtleBot and the environment are simulated at a frequency of 100 Hz, which means that 0.5 seconds of real-time simulation are executed at every time step. This simulation frequency allows for an accurate numerical approximation of the physical model within the Pybullet engine, while still being affordable in terms of required computational power. Given the maximum speed of the TurtleBot, these simulation settings mean that the robot either moves 0.11 meters forward, or turns 22 degrees around its axis at each time step.



(a) 3x3 environment (b) 4x4 environment (c) 5x5 environment

Figure 4: The three PyBullet environments used to train the TurtleBot. The starting position of the TurtleBot (top left) is kept constant, as well as the position of the goal, marked by the dark green circle (bottom right).

3.1.2 LiDAR Sensor

As a realistic LiDAR simulation would provide the distance to nearest objects in 360° for every single degree of rotation. This would implies 360 additional data points, as well as measurement uncertainties, which is unsuitable input data for quantum circuits due to the limited circuit depth and number of qubits that can be used. Therefore, we simulate a simplified version of the LiDAR sensor based on built-in PyBullet functions, that yield the distance from some starting point to the first object in a fixed line of sight, without uncertainty. The number of simulated laser beams is eight and nine for the classical and quantum RL algorithms, respectively. The choice of nine for the latter is further motivated in section 3.4.2.

3.1.3 Quantum Circuits

The quantum circuits are simulated using Pennylane, a software for differentiable programming of quantum computers [8]. To maximize the speed of the simulations, the Lightning Qubit device is used for the simulation and the gradients with respect to the parameters are calculated using adjoint differentiation. The latter is an efficient differentiation method, based on the ability to undo quantum operators by applying their adjoint $U^\dagger U |\phi\rangle = |\phi\rangle$. For more detailed description of this method, the interested reader can refer to [81].

3.2 Noise

To simulate the effect of noise, a depolarizing noise model from Pennylane [8] is applied to all qubits in the designated experiments. The noise rate is set to 2.753×10^4, in line with the median SX error rate of the seven-qubit device 'Nairobi' from IBM [82].

3.3 Hyperparameters

The agent is trained using a DQN algorithm, as explained in section 1.3.2. We use four different configurations of the input data, labeled by numbers 1-4 as shown in Table 1. Input configuration 1 is the same as the one used by Heimann et al. [42], employing only x and y coordinates, as well as the z-orientation, while the others also make use of additional LiDAR data. Configuration 2 only considers the three frontal LiDAR lasers, whereas configurations 3 and 4 utilize lasers pointing all around the TurtleBot (Figure 5). Configuration 4 will only be evaluated with classical models due to limited time and computational resources. The number of LiDAR lasers used is eight for the classical neural networks and nine for the quantum neural networks. The reason for having nine instead of eight lasers as input for the quantum model is a matter of encoding convenience, as the PQC is built with three or six qubits (see section 3.4.2).

The DQN algorithm is implemented using Stable Baselines3, an open-source Python library that contains a wide variety of reinforcement learning algorithms [83]. A custom environment according to the Gymnasium format specifies the behavior of the TurtleBot and its environment. The environment is kept constant throughout training, which means that all objects, the starting position of the TurtleBot and the location of the goal are the same for each episode. The behavior of the TurtleBot is guided by the reward function, which is constructed as follows:

$$\text{reward} = \begin{cases} -1.0, & \text{if the agent hits an object,} \\ -0.2, & \text{if the agent increases distance to the goal,} \\ +0.1, & \text{if the agent decreases distance to the goal.} \\ +10.0, & \text{if the agent reaches the goal.} \end{cases} \tag{22}$$

Any state in which the agent collides with an object is regarded as a terminal state that marks the end of an episode. Reaching the goal is also considered a terminal state, but the high positive reward nevertheless makes this the most favorable outcome for maximizing the final cumulative reward. Furthermore, the absolute value of the negative feedback for moving away from the goal is twice as large as the positive reward of getting closer to it, which motivates the agent to take the desired action. If the agent does not reach the goal after 200 time steps, the episode is terminated to prevent the robot from getting stuck in the environment indefinitely without learning. The total number of time steps that the agent is allowed to take in one training is fixed to 50 000 for each input configuration, except configuration 4 which is fixed to 70 000.

During training, the model is evaluated every 100 time steps. If the evaluation reward lies above the reward threshold of 12.5, 13.5, and 14 for each environment respectively, the training is considered successful and training stops early. This avoids overfitting of the models. The classical model is evaluated every 100 steps and only one evaluation is considered, as the simulation is completely deterministic, hence multiple evaluation yield the same outcome. In contrast, the quantum model is evaluated 3 times every 100 steps, as preliminary experiments showed that numerical fluctuations may influence the outcome when small learning rates are used, and training stops early if the mean reward lies above the threshold. All experiments are repeated a fixed number of times to account for randomness that enters the algorithm through the generation of transitions and mini-batch sampling.

3.4 DQN Function Approximators

3.4.1 Classical Network Architecture

Four different sizes of the deep neural networks are tested for the classical models. Each exists of an input layer with one node per observation, two hidden layers, and an output layer with

Label	Input data
1	x and y coordinates, z-orientation
2	x and y coordinates, z-orientation, 3 frontal LiDAR data points
3	x and y coordinates, z-orientation, 9 equidistant LiDAR data points
4	relative euclidian distance to the start and goal, equidistant LiDAR data points

Table 1: Overview of the four input configurations.

three nodes, each of which corresponds to the predicted Q-value of the available actions (left, right, straight). The two hidden layers have 8, 16, 32 or 64 nodes each. An overview of the number of parameters for each classical network configuration to be tested is given in Appendix Table 3. The limited size and depth of the classical networks is chosen to enable a relatively fair comparison with the quantum versions of the algorithm.

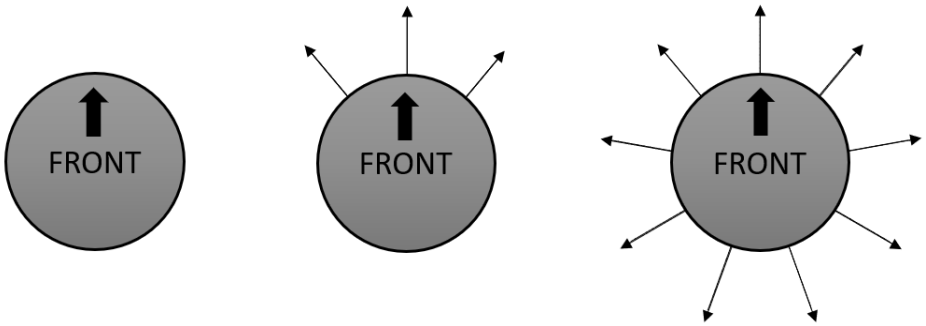


Figure 5: Schematic overview of the lidar lasers used in each of the input data configurations listed in Table 1. The grey circle represents the turtlebot. Configuration 1 uses no LiDAR (left), configuration 2 only employs the frontal lasers (center) and configurations 3 and 4 makes use of 9 equidistant lasers around the TurtleBot (right).

3.4.2 PQC Architecture

The schematics of the PQCs corresponding to each of the input data configurations (Table 1) are shown in Figure 6. The rotational gate U takes three rotational arguments as input. The parameters θ_i refer to the trainable parameters in the initial layer, and θ_{li} to the trainable parameters in layer l. The parameters x, y and z refer to the x, y coordinates and the z-orientation of the TurtleBot in the environment, respectively. For QNN-2a and QNN-2b, only the three frontal LiDAR data points $L_{i,j,k}$ are encoded using a single rotation gate (Figure 6b-c). For QNN2-a, instead of encoding the data points to a fixed qubit, they are encoded in a circular fashion, so the indices i, j, k shift one qubit in every consecutive layer. In QNN-3 (Figure 6d),

the parameters L_i refer to the LiDAR input data. Employing nine input parameters is convenient here, as they exactly fill the three rotational gates (Figure 6).

All input parameters, that is, x, y, z–orientation and LiDAR data, are passed through an arctangent function before being encoded as rotation angles, such that their values are within $[-\frac{\pi}{2}, \frac{\pi}{2}]$. In addition, they are multiplied by trainable parameters ϕ (not shown in the diagram to improve readability). The trainable layer, enclosed between the two barriers (dashed lines) is repeated L times. A small, single layer, fully connected CNN is used to process the measured Pauli-Z expectation values, as discussed in section 1.4.1. This adds 12 resp. 21 classical trainable parameters to the network for the 3-qubit and 6-qubit circuits. An overview of the number of parameters for each of the circuits with various numbers of layers can be found in Appendix II.

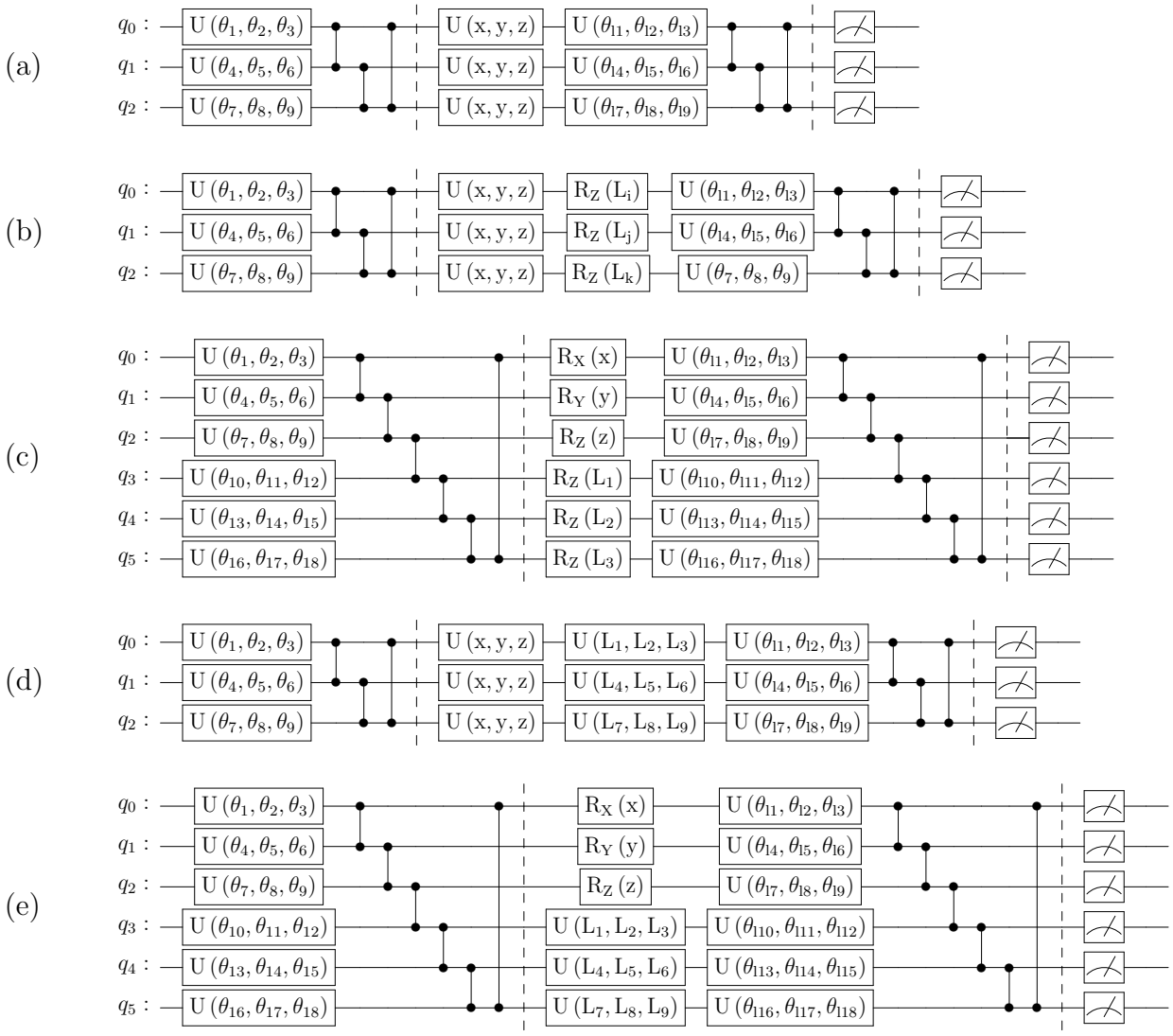


Figure 6: Overview of quantum circuits QNN-1 (a), QNN-2a (b), QNN-2b (c), QNN-3a (d) and QNN-3b (e). QNN-1 is identical to the quantum circuit used in [42], using general rotation gates $U(x_1, x_2, x_3)$ to encode the coordinates of the TurtleBot onto the quantum circuit. The models QNN-2a and QNN-3a follow a similar architecture as QNN-1, but contain an extra layer of general rotation gates to encode the LiDAR data. The circuits QNN-2b and QNN-3b are made of six qubits instead of three and have similar ansatzae and entanglement structures as the other circuits, but use different gates (single rotation gates) to encode the LiDAR data, as well as single rotation gates for the coordinate data in case of circuit QNN-2b.

4 Results

4.1 Classical Model Performance

4.1.1 Overall Training Process

The evaluation rewards of the best five out of ten experiments of classical models CNN-1, CNN-2 and CNN-3, in terms of fewest steps required to converge, are shown in Figures 7–9. Only the best five experiments are plotted to facilitate comparison with the results from Heimann et al. [42], who present the best ten out of twenty experiments. The bold line represents the average reward obtained at a given time step, and the shaded area indicates the standard deviation. All negative rewards have been scaled by 0.1 to improve readability, also in line with the results from Heimann et al. [42]. In addition, the evaluation reward is set to 11 for all subsequent time steps after a model has successfully reached the goal during evaluation. Hence, each steep local increase corresponds to one successful experiment, and a single line with a standard deviation of 0 indicates that all experiments have reached the goal.

The stepwise increase of the evaluation reward visible in nearly all plots for the 3x3 and 4x4 environments confirms that the models can successfully reach the goal. Inspection of the learned paths confirmed that all converged models learned a path that was close to the optimal path, which is shown in Figure 10. In contrast, only a single experiment (CNN-3, 64x64) could solve the 5x5 environment once – all other experiments fail to reach the goal. However, the evaluation rewards for this environment nevertheless increase as time progresses, indicating that the model learns to get closer to the goal despite not fully reaching it.
Figure 10c shows the optimal path and the learned path in the 5x5 environment. The black cross indicates the final point reached by the agent after 50 000 time steps. Reaching the goal via the learned path would take more time steps than the optimal path and requires four to five turns, compared to only two turns in the optimal case. A possible reason for learning this more complex path could be that turning left at the first path bifurcation comes with a higher risk of hitting obstacles in the short term. When the agent is too risk-averse, it will be difficult to learn this path. This indicates that the current reward function may not be optimal, and changing the relative differences between rewards and punishments may lead to better outcomes in the 5x5 environment.

4.1.2 Quantitative Performance Comparison

To enable a quantitative comparison between the various classical models, environments and network architectures, two performance measures are introduced. Firstly, the average number of time steps required to converge (ASTC), i.e., the number of time steps needed to find the goal, is a measure of how fast a model learns and its maximum value is 50 000, corresponding

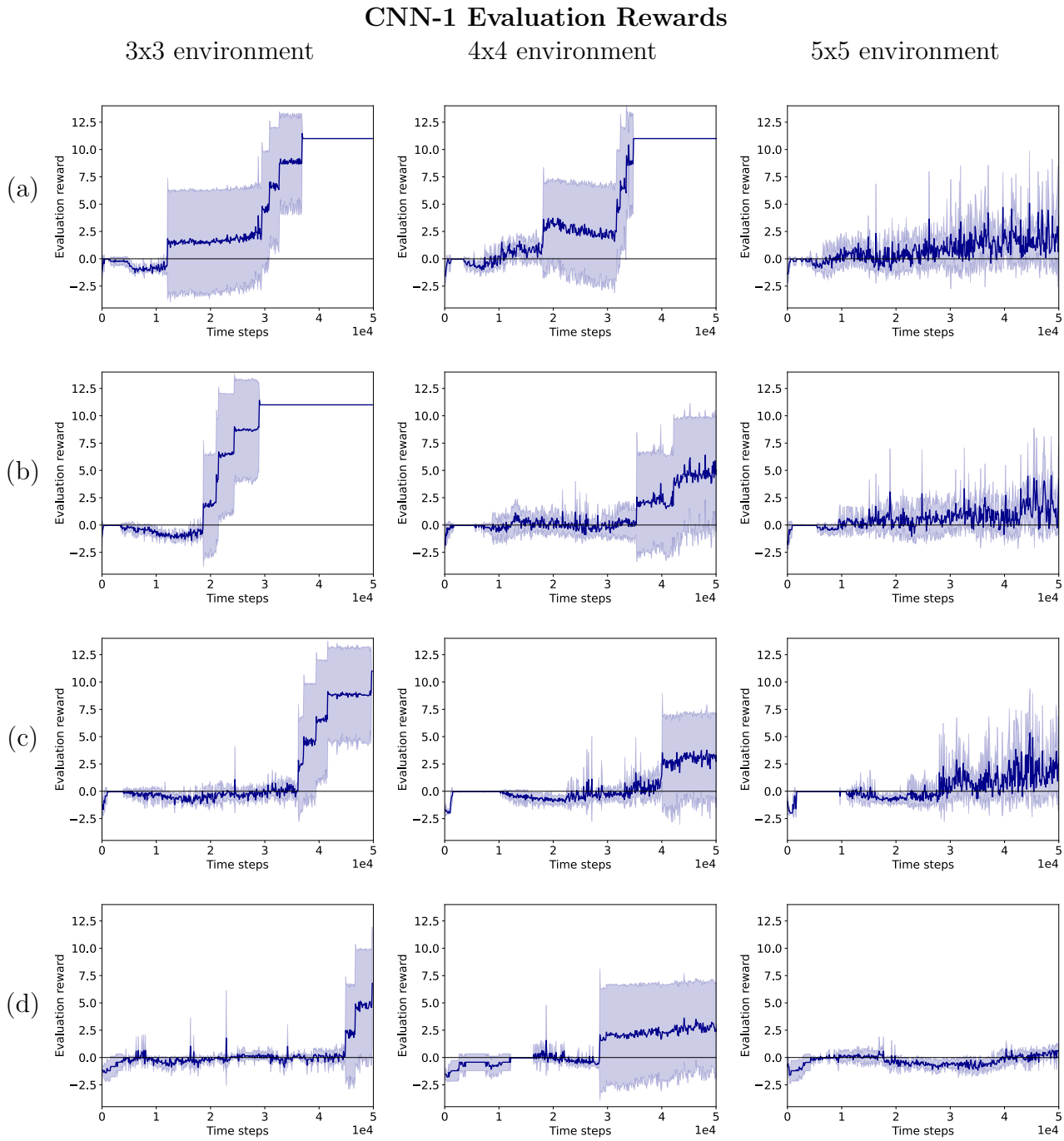


Figure 7: Evaluation reward of the five best experiments as a function of the number of training steps with the standard deviation for model CNN-1 using the 64x64 (a), 32x32 (b), 16x16 (c) and 8x8 (d) network architectures.

to the maximum number of time steps allowed for training. Secondly, the success rate (SR) is defined as the fraction of experiments for each model configuration that successfully converges, i.e., learns to reach the goal. Hence, this measure provides information about the robustness of the network.

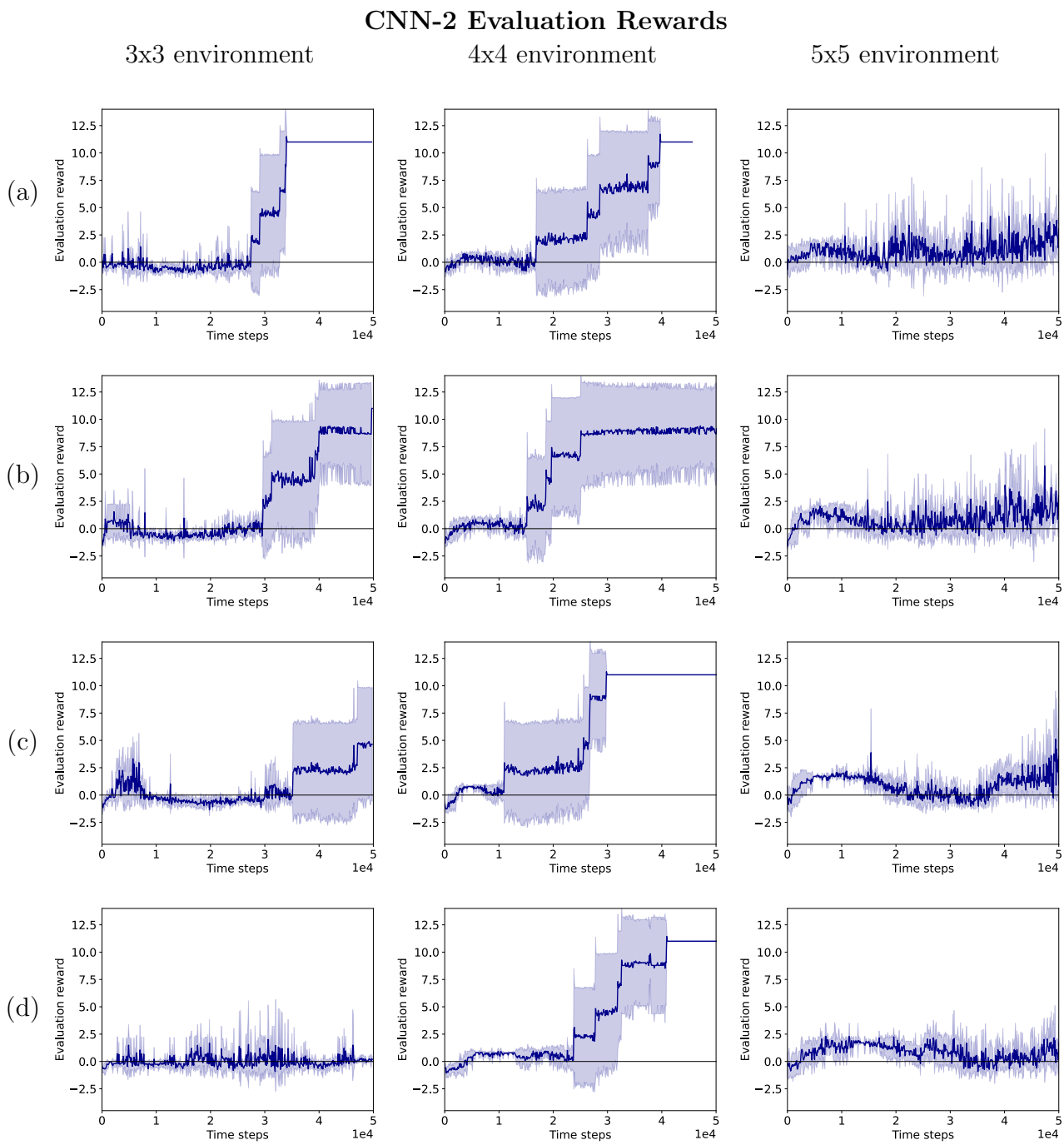


Figure 8: Evaluation reward of the best five experiments as a function of the number of training steps with the standard deviation for model CNN-2 using the 64x64 (a), 32x32 (b), 16x16 (c) and 8x8 (d) network architectures.

Figures 11 and 12 show the ASTC and SR for each model configuration and environment. The performance of model CNN-1, which utilizes the same input data and PQC architecture as [42], is significantly worse compared to results in [42]. An overview comparing the ASTC and SR is found in Appendix III. This indicates that there are some crucial differences between the imple-

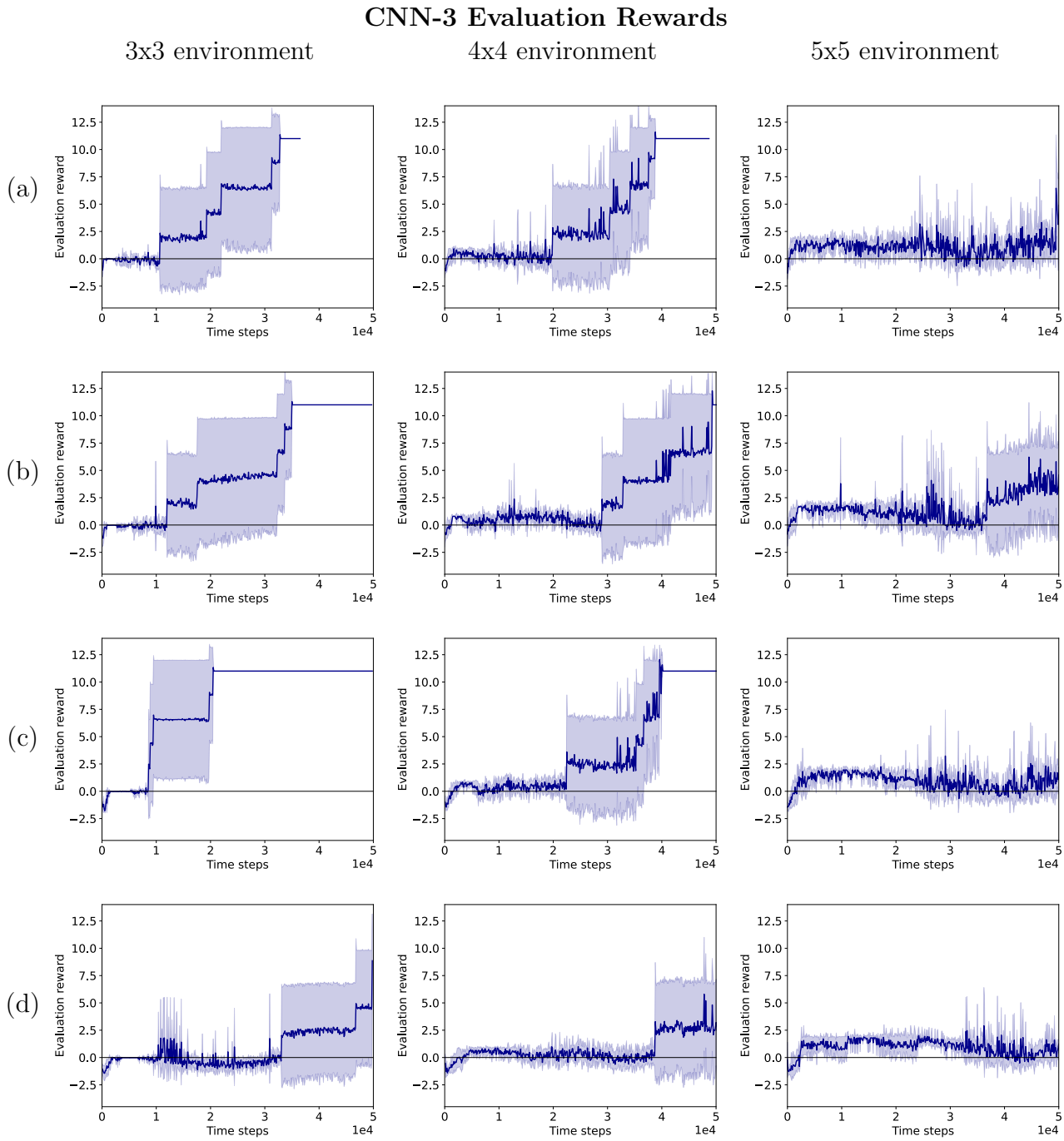
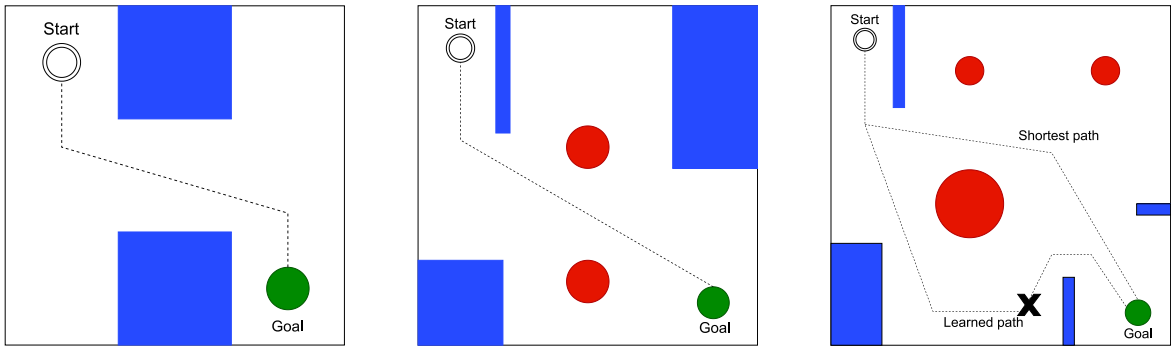


Figure 9: Evaluation reward of the best five experiments as a function of the number of training steps with the standard deviation for model CNN-3 using the 64x64 (a), 32x32 (b), 16x16 (c) and 8x8 (d) network architectures. The early interruptions of certain graphs, for example at *Time steps* = 37600 in graph a) 3x3, stems from the fact that none of the ten experiments performed needed more than the indicated number of time steps to converge.

mentation of the problem in this work and [42]. Various factors, including the DQN algorithm, hyperparameter settings and implementation, could attribute to the observed differences.



(a) 3x3 environment (b) 4x4 environment (c) 5x5 environment

Figure 10: Overview of the shortest learned paths for each environment, as well as the learned path for the 5x5 environment, where the cross indicates the farthest point that could be learned by the agent.

Regarding the addition of LiDAR data, CNN-3 performs best in terms of ASTC and SR for the 3x3 environment, indicating that the additional all-round LiDAR data enables the agent to find the goal in fewer steps, as well as more frequently. CNN-2, which utilizes frontal LiDAR data, only outperforms CNN-1 in terms of SR when the largest network architecture (64x64) is used, and just barely matches the performance of CNN-1 in terms of ASTC. Given the large standard deviations, it is unclear which of these two truly performs better, which suggests the addition of only frontal LiDAR data is less useful than adding all-round LiDAR data for solving the 3x3 environment. In contrast, CNN-2 clearly outperforms CNN-1 in the 4x4 environment, both in terms of SR and ASTC. Moreover, CNN-2 reaches a better ASTC than CNN-3, although the SR of CNN-3 mostly remains higher. This indicates that the three frontal sensors enable the agent to solve the 4x4 environment faster. In addition, any additional LiDAR data points may be useful, but the higher ASTC of CNN-3 shows that it takes more time steps to learn from this additional information. Finally, the 5x5 environment is solved only once after 48 700 time steps by CNN-3, barely before the training episode would have ended. Hence, the 5x5 environment seems too complex to be solved within the maximum number of time steps the agent is allowed to train under the current hyperparameter settings.

The SR and ASTC are plotted against the number of parameters of the classical models in Figures 13 and 14 to analyze whether a correlation can be observed. In the 3x3 environment, the SR is fit to the inverse square of the number of trainable parameters. The R2-scores of 0.90, 0.99 and 0.72 for the models CNN-1, CNN-2 and CNN-3, respectively, indicate that this is a reasonable fit. An exponential decay is fitted to the ASTC in the 3x3 environments, but the R2-scores are significantly lower, indicating that this relation is not significant. In addition, for the 4x4 and 5x5 environments, none of the fits reached an R2-score above 0.5 and have

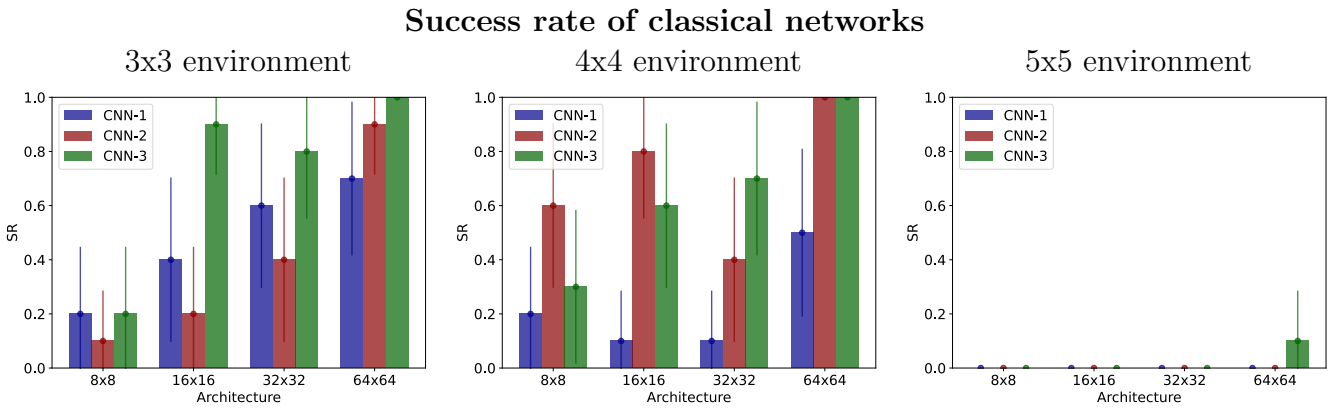


Figure 11: The success rate (SR) of the classical network architectures in each of the environments. The error bars indicate the Wald interval.

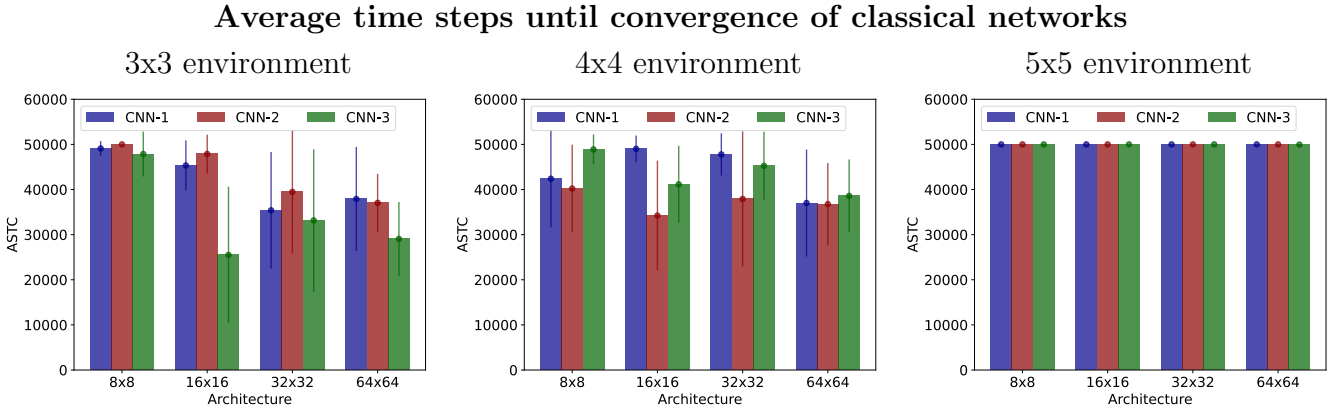


Figure 12: The average number of time steps until convergence (ASTC) per network architecture for the 3x3, 4x4 and 5x5 environments. The error bar indicates the standard deviation.

therefore been omitted. Given these observations, there seems to be no consistent relationship between the number of trainable parameters and the performance of a solution that holds for all the environments. Nevertheless, a general increase in performance is observed as the number of parameters increases, which is expected as larger neural networks tend to be able to capture more complex correlations within the data, to a certain degree [84].

4.1.3 Navigation with Relative Coordinates

The average evaluation rewards per time step of CNN-4, which uses the relative distances to the start and goal positions of the LiDAR, are shown in Figure 15. All training sessions from the 3x3 and 4x4 environments are solved successfully, although it requires relatively many training steps (4500-6700) as well as a larger neural network, consisting of two hidden layers with 128 nodes each, to find a solution. The 5x5 environment is solved only for one out of five trials. The other four experiments suffered from exploding gradients as early as 20 000 steps and latest at

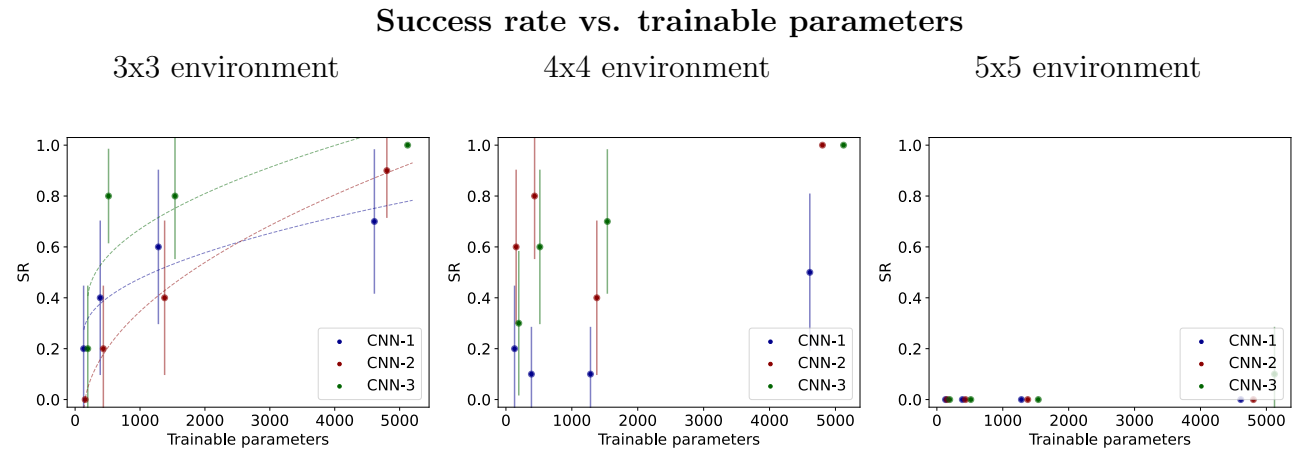


Figure 13: The success rate (SR) of each of the network architectures as a function of the number of parameters, with error bars showing the Wald interval. A square root function is fitted for results from the 3x3 environment with R2-scores of 0.90, 0.99 and 0.72 for the models CNN-1, CNN-2 and CNN-3, respectively. Fits to the data in the larger environments have been omitted due to low R2-scores.

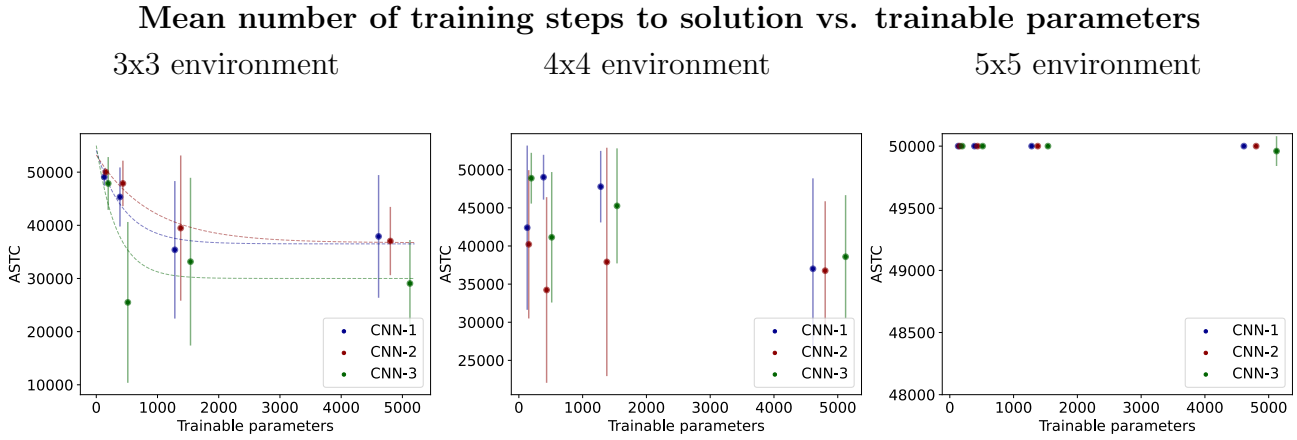


Figure 14: Average number of time steps (ASTC) needed to find a solution as a function of the number of parameters for each classical model. The error bars indicate the standard deviation. An exponential decay is fitted for the 3x3 environment, with R2-scores of 0.51, 0.66 and 0.23 for CNN-1, CNN-2 and CNN-3, respectively.

52 200 steps, which explains the smooth continuation of the graph for the 5x5 environment, as the agent does not learn anymore. This problem is often observed when the estimated Q-values become too large, leading to nearly-infinite gradients. A possible solution could be downscaling the values of the rewards in the reward function, or clipping the gradients [85].

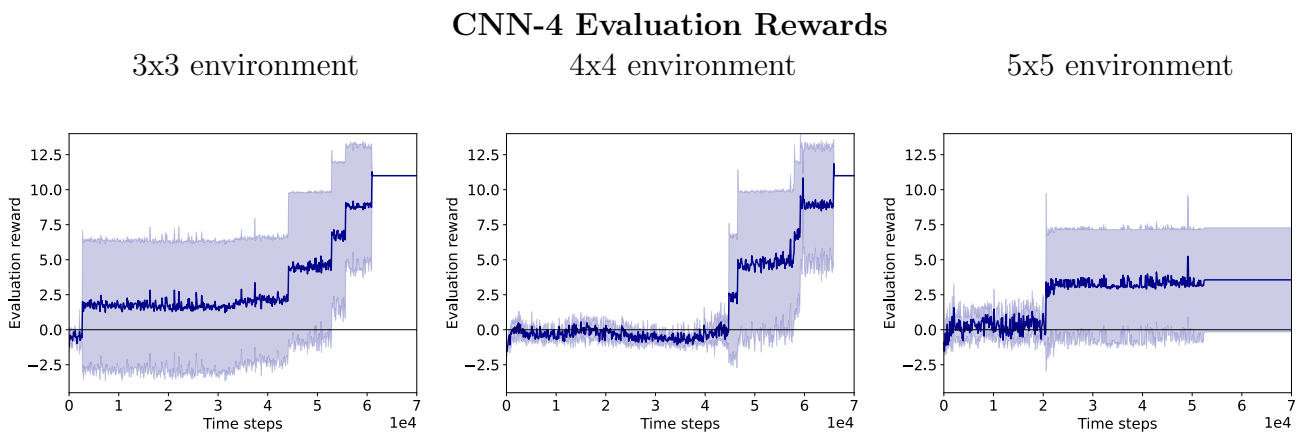


Figure 15: Evaluation reward plotted against the number of time steps for model CNN-4 in different environments using a 128x128 network architecture. The error bands indicate the standard deviation

4.2 Performance of Quantum Models

4.2.1 Overall Training Process

The evaluation rewards of the three quantum experiments plotted against the number of time steps in the 3x3 environment[3] is shown in Figure 16. The standard deviation for all experiments is relatively large, as only three repetitions of each experiment could be executed due to time constraints. This also means that at most three stepwise increases in the evaluation reward can be observed, which would correspond to three successful experiments. Model QNN-2b with ten layers performs best overall, converging to a solution in three out of three experiments and within a number of time steps that is comparable to the best classical network for CNN-2. Specifically, CNN-2 has 4803 trainable parameters, compared to 279 (including the weights of the classical post-processing layer) in the case of QNN-2b. This illustrates a potential advantage of using a PQC instead of a classical network inside the DQN algorithm for this particular task.

4.2.2 Quantitative Comparison

The previously introduced RL performance metrics are the ASTC, which indicates how fast a model can learn, and the SR, which provides information about the stability of the model. However, neither of them address the quality of the solution. This especially hinders the assessment of the learned solution when a model does not find a solution at all, as is the case for several quantum models (Section 4.2.1 and Figure 16).

[3]The 4x4 and 5x5 environments have not been evaluated due to time constraints.

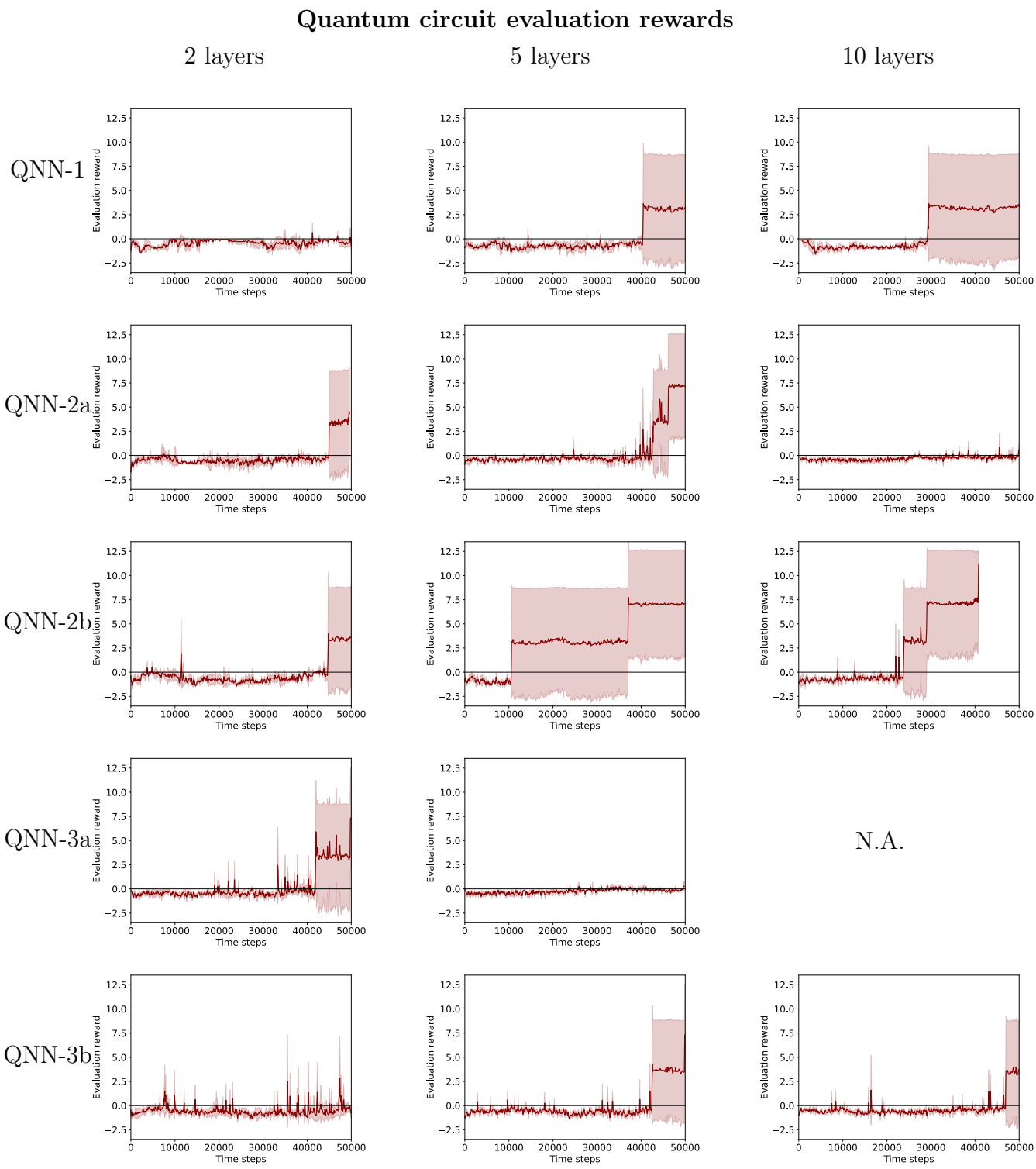


Figure 16: Evaluation reward of three quantum experiments per quantum model plotted against the number of training steps, with the standard deviation indicated by the error bands. Results from QNN-3a with ten layers have been omitted due to incompleteness.

Therefore, solution quality (SQ) metric will be introduced, which is calculated as follows:

$$SQ = \#\text{CP} + \left[1 - \frac{d(\text{agent}, \text{CP}_{t+1})}{d(\text{CP}_t, \text{CP}_{t+1})}\right] \tag{23}$$

The parameter #CP refers to the number of checkpoints passed, with a checkpoint defined as successfully learning to turn in the correct direction in within a 0.3 meter radius of the turning points of the optimal path shown in Figure 10. The goal is also considered a checkpoint, which means that there are three checkpoints in the 3x3 environment. The term $d(\text{agent}, \text{CP}_t + 1)$ refers to the distance between the position of the agent and the position of the next checkpoint. The term $d(\text{CP}_t, \text{CP}_{t+1})$ denotes the total distance between the last checkpoint reached and the next one. Hence, the full term $[1 - d(\text{agent}, \text{CP}_t + 1)/d(\text{CP}_t, \text{CP}_t - 1)]$ shows what fraction of the total path, in terms of checkpoints, has been learned successfully. For example, a solution quality value of SQ = 0.9 would indicate that the agent has almost reached the first checkpoint, SQ = 1.5 indicates that the agent learned to move exactly halfway between the first and the second checkpoint, and a SQ = 3 (highest SQ possible) means that the agent has successfully reached the goal. Hence, the SQ, with of values in the range [0, 3], is a measurement of the quality of the learned solution in terms of successfully navigated turns and paths, even when the goal is not reached.

The ASTC, SR and average SQ (ASQ) have been plotted for each model in Figure 17. The ASQ mosly increases when frontal LiDAR data is employed, indicating that this is valuable input data for the quantum models, in line with the results from the classical models. Most models benefit from an increased number of layers in terms of ASTC, except QNN-2a, which performs well (SR=0.667 and ASTC=46 400) when five layers are applied but badly (SR=0, ASTC=50 000) with ten layers. This may indicate that there is an optimal number of layers for each individual PQC, which has been observed by previous research, e.g., by [86] for quantum classifiers. Overall, the large differences in performance metrics between the various quantum models indicate that the PQC architecture is highly linked to their performance, as previous research has often observed, e.g., [50, 87].

4.3 Quantum Metrics

4.3.1 Expressibility

The expressibility values of the quantum circuits are shown in Figure 18(a). The PQCs with three qubits, namely QNN-1, QNN-2a, and QNN-3a, converge to an expressibility of 0.04 ± 0.001 after the fourth layer. In contrast, the six-qubit PQCs QNN-2b and QNN-3b reach a significantly lower expressibility of 0.0008 ± 0.0003, and only converge to that value after six layers. As the the type and sequence of the gates is similar for the three- and six-qubit

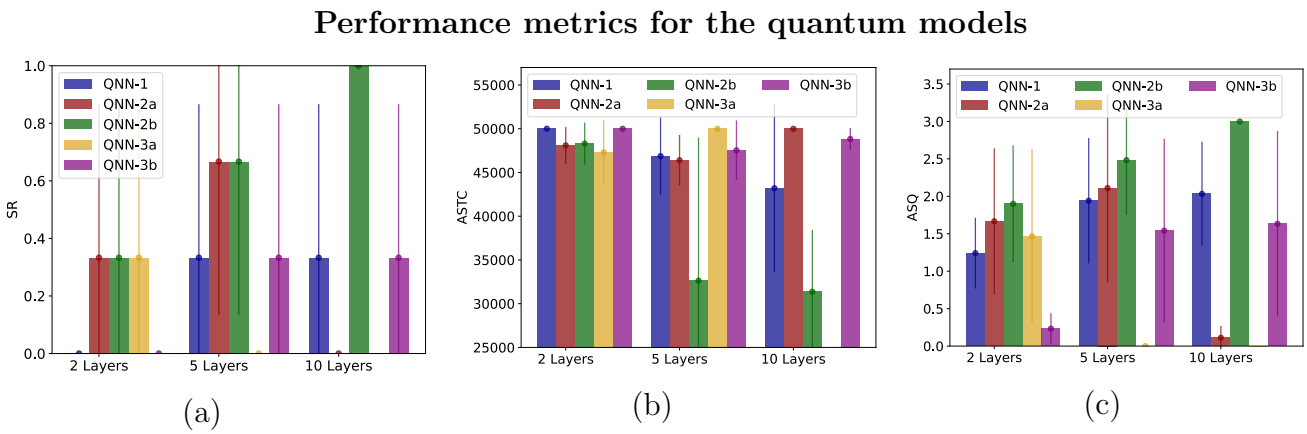


Figure 17: The ASTC, SR and ASQ of the quantum models with 2, 5 and 10 layers are presented in (a), (b) and (c), respectively. The error bars indicate the standard deviation. Model QNN-3b with ten layers has been omitted due to incomplete data.

architectures (Figure 6), this may indicate that increasing the number of qubits improves the expressibility of the circuits. Similar observations have been made by [88].

4.3.2 Entanglement Capability

The entanglement capabilities of the five quantum circuits are shown in Figure 18(b). The models QNN-1, QNN-2a and QNN-3 have a similar maximum entanglement capability of 0.67 ± 0.01, which saturates after four circuit layers. In contrast, the entanglement capability of QNN-2b and QNN-3b (i.e., the six-qubit circuits) only saturates after ten layers, and the absolute entanglement capability is also considerably higher (maximum 0.95 ± 0.01). As entanglement capability depends on the quantum gate operations, rather than the number of qubits, the higher values may be attributed to a fundamental difference in PQC architecture between the three- and six-qubit PQCs. Specifically, the six-qubit PQCs have relatively fewer encoding operations per qubit before the entangling takes place (Figure 6). It seems that this benefits the performance of the PQCs, as QNN-2b and QNN-3b both outperform their three-qubit counterparts QNN-2a and QNN-3a in terms of performance metrics (Figure 17).

4.3.3 Normalized Effective Dimension

All single-qubit gates in the quantum models are trainable, as explained in Section 3.4.2. However, for the calculations of the NED, the gates that encode the data were considered non-trainable. The reasoning behind this choice is that the authors of [56] did not include this factor in their definition of the metric, and it was non-trivial to adjust the calculations to the PQCs used in this work. For the same reason, the classical fully-connected network that processes the outputs from the quantum circuit is also not incorporated in the calculations.

Therefore, NED values presented in this section should be considered as a relative metric to compare the PQCs in this work, as the absolute values do not carry significance. The NED values are plotted against various numbers of layers in Figure 18(c). The NED follows a decreasing trend for all architectures. However, the values of the three-qubit PQCs, QNN-1, QNN-2a, and QNN-3a, decrease more steeply after the second layer than the six-qubit PQCs QNN-2b and QNN-3b. Based on the interpretation of the NED in [56], this would indicate that a relatively larger fraction of the trainable parameters of the six-qubit PQCs still actively contribute to the output after adding more layers, compared to the three-qubit PQCs. As a result, one would expect that adding more layers could be more beneficial for PQCs QNN-2b and QNN-3b than for the others. The results in sections 4.2.1 and 4.2.2 show that this is indeed the case for model QNN-2b, which improves the most between two and ten layers compared to all other models, but not for QNN-3b. Moreover, the best performing quantum model is QNN-2b at ten layers, and results show that this model has the lowest NED overall. Hence, the NED does not seem to be a strong indicator of quantum circuit performance in the context of the problem studied in this work.

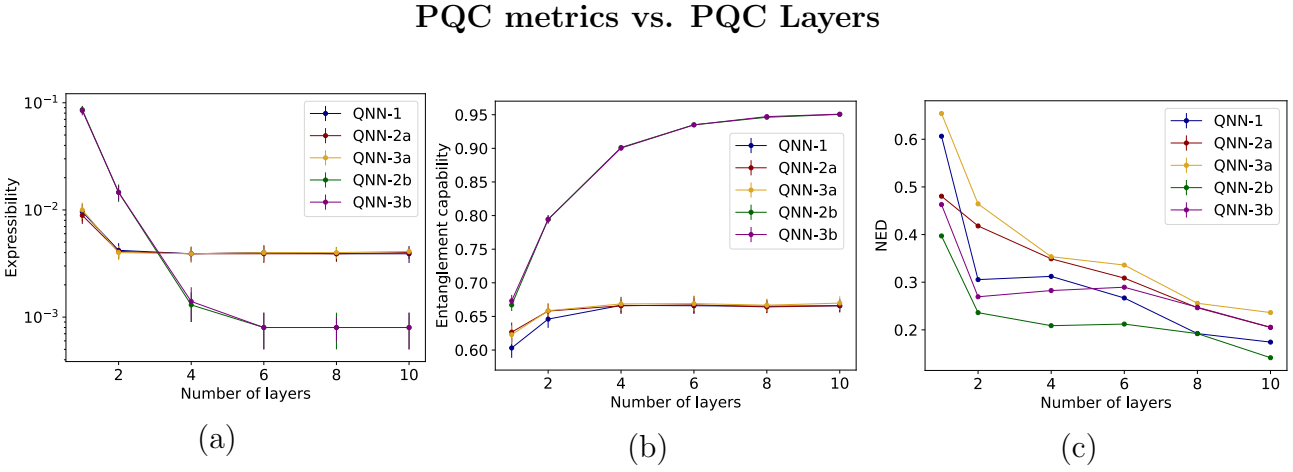


Figure 18: The expressibility (a), entanglement capability (b) and normalized effective dimension (NED)(c) plotted against various numbers of layers in the PQC, with error bars indicating the standard deviation. Note that the vertical axis of the expressibility is a logarithmic scale.

4.4 Correlation between Quantum Metrics and Model Performance

The performance metrics SR, ASTC and ASQ have been plotted against the quantum metrics, namely expressibility, entanglement capability, and normalized effective dimension in Figure 19. Overall, the models not show any strong correlations between the performance metrics and the quantum metrics, based on the r-scores ($r < 0.70$). The only quantum model that shows a very significant individual correlation in all comparisons is QNN-2b ($r > 0.90$). Nevertheless, the sign

Metric	Without noise	Depolarizing noise
SR	0.333 ± 0.23	0.333 ± 0.23
ASTC	46866 ± 4431	46600 ± 4808
SQ	1.94 ± 0.83	1.90 ± 0.79

Table 2: Comparison of performance metrics of QNN-1 with and without depolarizing noise.

of the correlations, i.e., positive or negative, is as expected from the theoretical interpretation of the entanglement capability and expressibility. Specifically, the SR and ASQ, for which higher values indicated better performance, are positively correlated with the entanglement capability and negatively correlated with the expressibility, while the inverse is true for the ASTC, which should be as low as possible. In contrast, the relation between the performance metrics and the effective dimension is opposite to theoretical expectations. In theory, a higher effective dimension should be more favorable as claimed by the authors of [56], who proposed the metric, but the results show that it is actually correlated with worse performance, as the NED shows weak but negative correlations with the SR and ASQ and a positive correlations with the ASTC. Especially QNN-2b, which is the best performing model overall, shows a strong negative correlation with the NED ($r = 0.76$). The weak correlations overall, as well as the unexpected sign of the correlations of the NED with the performance metrics, indicate that the currently investigated quantum metrics are not necessarily good indicators of performance for the problem at hand.

4.5 Influence of a Depolarizing Noise Model

The performance of model QNN-1 with five layers after the implementation of a depolarizing noise model is shown in Figure 20b, next to the results of the same experiment without noise in Figure 20a. The overall training process looks very similar, which is verified by the quantitative comparisons of the performance metrics in table 2. This may indicate that the algorithm is capable of learning the noise, which would make it robust and suitable for execution on real NISQ-hardware. However, the noise rate used for the experiments was based on the single-qubit gate error provided by IBM [82]. Two-qubit gate errors occur with a rate that is an order of magnitude larger, so the current results may not accurately represent the full severity of noise on real hardware.

4.6 Comparison between Classical and Quantum Performance

Figure 21 shows the performances in terms of SR and ASTC plotted against the number of parameters for the classical models with the smallest architecture (8x8), as well as 5-layer versions of the quantum models. These models have a comparable range of trainable parameters

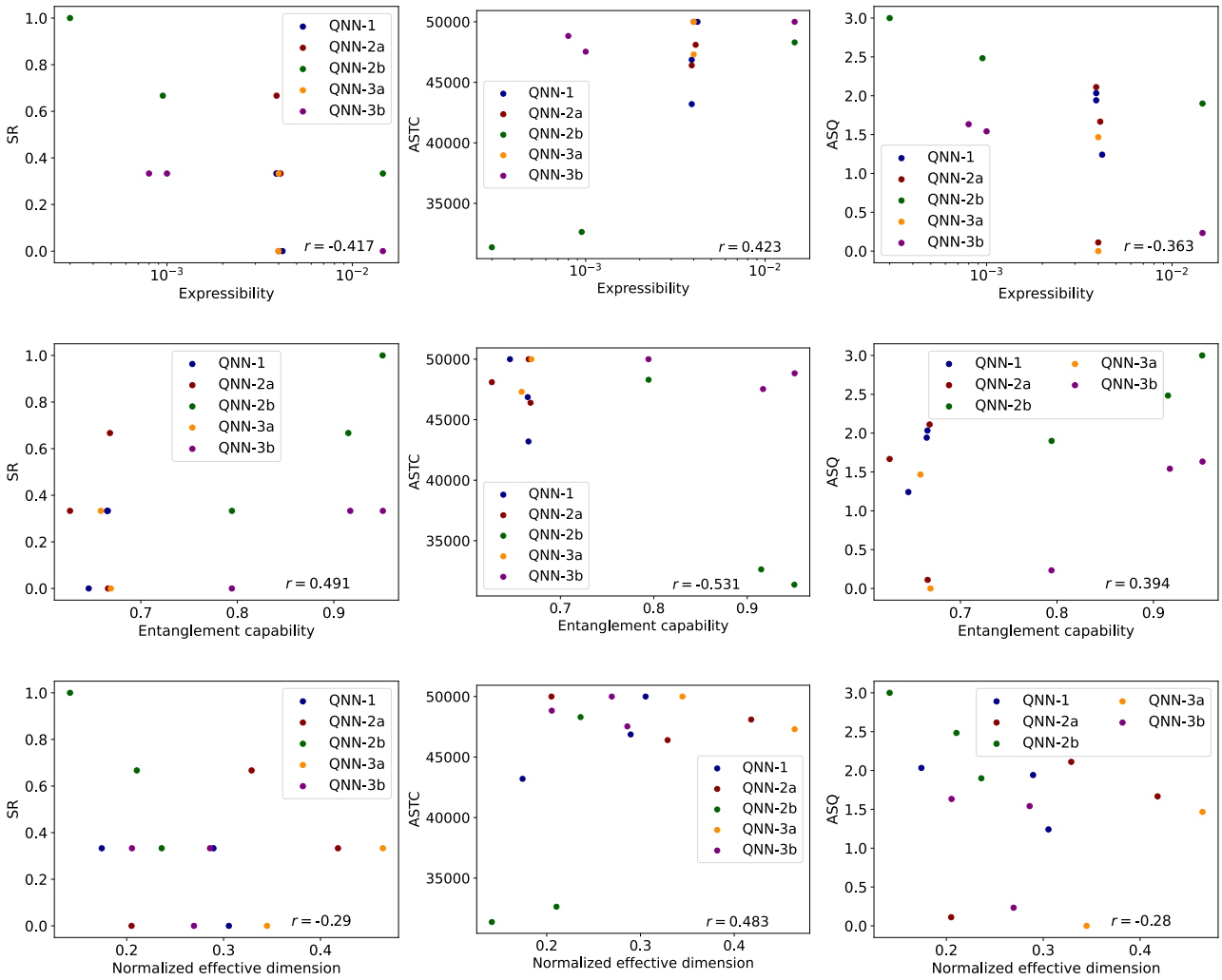
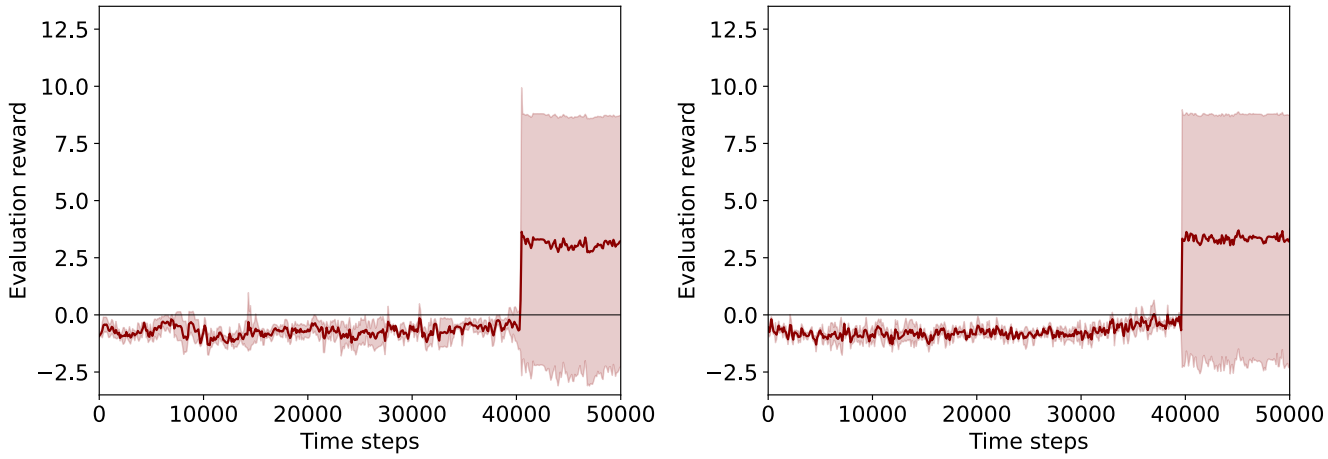


Figure 19: The expressibility (top), entanglement capability (middle) and normalized effective dimension (bottom) plotted against the average success rate (SR), average time steps to convergence (ASTC) and the average solution quality (ASQ). The Pearson correlation for each plot is indicated by r. Error bars have been omitted to improve readability, see Figures 17 and 18.

(99–195). The quantum models perform better, both in terms of SR and ASTC, even when they have fewer trainable parameters. The only exception is QNN-3a, which has the lowest overall performance. Overall, this is a promising result, which may imply a scaling advantage for quantum reinforcement learning with regards to the required number of trainable parameters, at least in the context of navigation tasks.

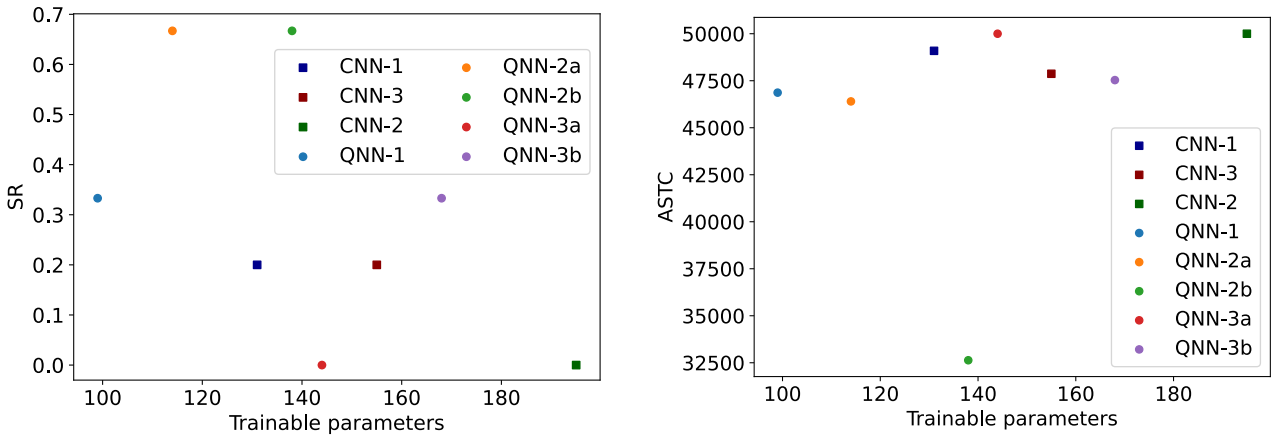
Impact of Depolarizing Noise on QNN-1



(a) (b)

Figure 20: The evaluation rewards of model QNN-1 with five layers without noise (a) and with depolarizing noise (b) plotted against the number time steps. Error bands indicate the standard deviaton.

Classical and Quantum Performance vs. Trainable Parameters



(a) (b)

Figure 21: The SR (a) and ASTC (b) plotted against the number of trainable parameters of the 8x8 classical models and 5-layer quantum models. Classical models are indicated with a square and quantum models with a circle. Error bars have been omitted to improve readability.

5 Discussion

5.1 Classical Models

5.1.1 Classical Model Performance

The results of the classical models show that they need significantly more training steps to find a solution to the problem for all three environments, compared to the results shown by Heimann et. al. [42]. This is not completely unexpected, as decreasing the speed with which the TurtleBot moves effectively increases the number of steps that need to be learned before the goal is reached. However, given the extent of the differences, there are various other factors that should be considered. First, the RL model used in this work is the DQN algorithm, instead of the Double-DQN that is used in [42]. The DQN uses a single network to predict the Q-values and select the actions based on these predictions, whereas the Double-DQN uses a separate network for the action selection. This potentially leads the DQN to predict overly optimistic estimation of Q-values, which lowers performance [89]. The decision to use the DQN algorithm instead was motivated by the unavailability of the Double DQN in the latest version of Stable Baselines-3, and the mention in the paper that the Double-DQN performed 'slightly better on average'. However, the results are significantly different, which may indicate that other factors may contribute to the differences. A likely suspect is the implementation of the algorithm, as various numerical aspects of the DQN algorithm may be handled differently in Stable Baselines-3 and Keras.

5.1.2 LiDAR Data and Environment Complexity

The frontal LiDAR data improves performance in the 4x4 environment compared to the case where no LiDAR data is present, but does not significantly improve performance in the 3x3 environment. In contrast, adding all-round LiDAR data improves performance in both environments. This indicates that the three frontal sensors are mostly beneficial in the 4x4 environment, whereas solving the 3x3 environment requires additional LiDAR data from the sides and/or back of the TurtleBot. A possible explanation for this may be that the path to solve the 4x4 environment is easier to learn, as it only involves one big turn in the top left corner of the environment, after which the TurtleBot could walk straight to the goal, as illustrated in Figure 10c. This means that the agent mostly needs to learn how to avoid objects, for which the frontal LiDAR sensors may be useful. In contrast, the 3x3 and 5x5 environments require the robot to turn multiple times (Figure 10a,c), which may indicate that it becomes more important that the agent learns to orientate itself with respect to the surrounding objects. In this case, it seems sensible that LiDAR data from the sides and back are useful. An important takeaway from these observations is that the addition of LiDAR data impacts the performance of the RL algorithm differently depending on the complexity of the environment. The complexity of the

environment depends not only on its size, but also on the path required to reach the goal and possibly other factors. Establishing a measure for the complexity is therefore important when benchmarking similar sensor-assisted navigation problems.

Model CNN-4, using only sensor data and two relative coordinates to the goal, shows a high convergence rate for the 3x3 and 4x4 environments, which is a promising result. Except for one experiment in the 3x3 environment for which the solution was found after very few time steps, the overall ASTC is very similar for both environments. Therefore, it seems that model CNN-4 is relatively robust to changes in environment size and complexity compared to the other models, which illustrates its generalization capabilities. This is a positive outlook for dealing with environments in which the objects are non-static, which is the case in many real-life situations. However, it should be noted that this model required a significantly larger network architecture and more time steps to converge overall compared to the others. This could make training with quantum models challenging, especially when using real hardware, as the PQCs should remain relatively shallow and able to converge within a limited number of time steps to be compatible with NISQ devices.

5.2 Quantum Models

5.2.1 Encoding LiDAR Data

The results show that adding frontal LiDAR data improves the overall performance of the quantum models compared to the case where no LiDAR data is present, whereas adding all-round LiDAR data has the opposite effect. In contrast, the performance of the classical models improved when all-round LiDAR data was added in the 3x3 environment, compared to the case when only frontal LiDAR data was provided. From these observations, one may conclude that only limited input data can be encoded on a single qubit per layer when using angle encoding. This seems plausible from a theoretical point of view, as applying a sequence of rotation gates without entangling operations in between is equivalent to applying a single rotation gate that is the matrix product of the individual gates. Hence, combining the coordinates and LiDAR data in this way may lead to a loss of crucial information. Therefore, it seems that spreading input data over multiple qubits and adding entanglement between the data uploading parts of the circuit is a beneficial approach for navigation tasks.

5.2.2 Quantum Metrics vs Performance Metrics

No strong correlation was observed between the majority of the quantum metrics (expressibility, entanglement capability, effective dimension) and the performance metrics (ASTC, SR, ASQ). This is in line with the findings of [62], who did not observe clear correlations between the quantum metrics and performance of an agent in the frozen lake environment. The exception

was quantum model QNN-2b, which had a strong positive correlation with the entanglement capability and a negative correlation with the expressibility, which was expected from theoretical interpretations of these metrics [53, 55]. The negative correlation between QNN-2b and the normalized effective dimension, however, was contradictory to the theoretical expectations formulated by [56], suggesting that there are limitations to this metric. Specifically, given that the NED is a normalized metric, it may not capture the benefits of data re-uploading, as additional layers come with more trainable parameters. Although a relatively smaller number of parameters may contribute meaningfully to the predictions of the PQC when layers are added, there are still more relevant parameters in an absolute sense, which contribute to the success of the PQC. One could argue that it is more useful to compare the non-normalized effective dimension, but this prohibits the metric from comparing fundamentally different PQC architectures. Hence, the normalized effective dimension may not be a sufficiently generalizable metric to compare quantum circuits, especially when data re-uploading is used. It should be noted, however, that the DQN algorithm comes with a high variance in performance even in the classical case, as illustrated by the large standard deviations of the performance metrics for the classical models in this work. Therefore, it is too early to conclude that these quantum metrics are not suitable for quantifying the quality of PQC architectures for machine learning tasks. Still, in the case of quantum reinforcement learning, it may be worthwhile to investigate other metrics for further research. For example, the correlation between the performance and the data encoding strategy, where fewer inputs per qubit per layer seemed to increase the performance, may be a direction worth pursuing.

5.3 Comparison between Classical and Quantum Model Performance

Results show that nearly all quantum models perform better in the 3x3 environment than the smallest classical model, which is the only classical model that is comparable in size with respect to the number of trainable parameters. In addition, the best quantum model QNN-2b showed a similar performance compared to the best largest classical model CNN-2, despite having an order of magnitude fewer parameters. This is in line with findings from e.g., [46, 63, 65]. The observation that quantum models perform better with fewer trainable parameters is a positive outlook for quantum machine learning research. However, it is unclear how this advantage scales for problems that require several orders of magnitude more trainable parameters to be solved classically. In addition, state-of-the-art classical models can relatively easily incorporate millions of trainable parameters, whereas quantum models suitable for NISQ-devices are limited to only some hundreds or thousands. Therefore, it is debatable whether it is fair to compare quantum and classical models based on their parameter count. Regardless, further research into the potential scaling advantage of quantum models for larger and more complex problems is necessary before any such conclusions can be drawn.

6 Conclusions and Outlook

This work employed a DQN algorithm to train an agent, specifically a TurtleBot equipped with LiDAR sensors, to navigate through three environments of different sizes and filled with various objects, using a variety of classical neural networks and parametrized quantum circuits to approximate the Q-function. To the best of knowledge, this work is the first to investigate the feasibility of navigating a realistic sensor-assisted robot using quantum reinforcement learning. The agent, environment, and the simplified LiDAR sensor with a variable number of lasers created for this work constitute a realistic simulation environment that can easily be extended to include, for example, noisy LiDAR signals and dynamic obstacles. Therefore, this work paves the way to investigating the use of quantum reinforcement learning in highly realistic settings. In addition, the investigation of various quantum circuits provided indications about which type circuit designs work well for the problem at hand, which is a useful result for future research projects involving sensor-assisted agents.

The feasibility of training the TurtleBot with the (quantum) DQN algorithm was evaluated in various ways. Several performance metrics were introduced to analyze the performance of the agent in terms of robustness, training speed and solution quality. Moreover, the parameterized quantum circuits were analyzed and their performance with respect to various quantum circuit metrics, which have recently been introduced by various members of the quantum machine learning community, was investigated. In addition, a depolarizing noise model was introduced to the sensorless quantum model to investigate the robustness of the algorithm. The performance of the agent without using LiDAR data, similar to the agent used by Heimann et al. [42], did not match the results of their paper. Apart from minor differences in the settings of the robot, the use of the DQN algorithm instead of the Double-DQN algorithm used in the paper [42] may have lowered the performance of the agent, as well as the use of a different implementation of the algorithm. Hence, further work is strongly advised to investigate which of these factors contributed most strongly to the observed differences. In addition, more advanced versions of the DQN algorithm that include e.g., prioritized experience replay may boost performances.

Experiments with the classical models revealed that additional LiDAR data improved the performance of the agent with respect to the average number of steps required to find the goal, as well as the success rate of the experiments. As expected, an overall improved performance was observed as the classical models increased in size with respect to the number of trainable parameters. In addition, the contribution of three or eight extra LiDAR data points to the success of the agent was different in all three environments. Both observations could be linked to the complexity of the environment, which seems to be a non-trivial aspect of the problem and

depends on various factors including the size and path to be learned by the agent. Therefore, future research is recommended to thoroughly analyze and, if possible, quantify, the complexity of the problem at hand, as this is a crucial aspect of benchmarking performances of (quantum) reinforcement learning algorithms. Using the metric of success quality, which was first introduced in this paper and incorporates the distance and complexity of the path with respect to the number of turns that need to be learned at specific locations, may be a good start to develop a more generalizable quantification of problem complexity.

After providing access to LiDAR data, the quantum models showed both improved and decreased performance of the agent in terms of number of steps required to find a solution, success rate and success quality, depending on the quantum circuit architecture. The best performance was reached by a six-qubit quantum circuit with three LiDAR data points, whereas the worst performance was a 3-qubit architecture utilizing nine LiDAR data points. Three quantum metrics were analyzed: expressibility, entanglement capability and normalized effective dimension. Overall, no strong correlations with these quantum metrics were observed. However, the performance of the agent did seem to be correlated with the number of entanglement operations per qubit compared to the number of trainable parameters per qubit. This may be worthwhile to investigate in further research, as it is not yet clear how this potential benefit scales as the number of qubits in the circuit increases.

When comparing the classical and quantum models in terms of number of trainable parameters, all except one of the quantum models outperformed the classical model. In addition, the performance of the best quantum model was similar to the performance of the largest classical model, despite the latter having nearly twenty times fewer trainable parameters. This may indicate an advantage of quantum reinforcement learning for navigation tasks on small scales. However, it is unclear how this advantage scales for problems requiring more trainable parameters. This could be investigated with the fourth model, which was tested only classically in this work, as it required roughly four times as many parameters as the other models.

Furthermore, insights about the generalization capability of both classical and quantum models to dynamic and previously unseen environments should be further investigated. In this work, the robot simply learned a fixed path in a static environment, which could also be solved by simply programming the robot to take a certain sequence of predetermined steps. The true relevance of using (quantum) reinforcement learning appears when the environment includes unpredictable changes such as misplaced or moving objects. Hence, to make the work truly relevant to the real world, the robot should be trained in a new, (semi-)randomly generated environment at the start of each training episode and evaluation. This can readily be implemented in the existing code, which offers flexibility with regards to the simulated environment.

Regarding the implementation of noisy quantum simulations, the performance of the model without LiDAR data under the influence of depolarizing noise was hardly different from the performance of the same model without noise. This might imply that noise can be learned by the model and therefore makes it suitable for execution on real NISQ-hardware. However, future investigations should use more realistic noise models with higher two-qubit gate noise rates and different types of noise to verify these findings.

To summarize, this work has shown for the first time that it is possible to solve a simple sensor-assisted robot navigation task using quantum reinforcement learning and uncovered insights regarding suitability of certain quantum circuits and the complexity of the navigation problem itself. Immediate next steps include investigating what causes the differences in performance between the various quantum circuits, as well as researching the generalization capability of both classical and quantum models to dynamic and previously unseen environments. The latter can readily be tested with the existing code. Although real-life implementation of a robot trained with a quantum reinforcement learning algorithm is currently restricted by the available quantum hardware, this work and future extensions may provide the software and theoretical foundations needed to pursue that goal.

Acknowledgements

First of all, I would like to thank Jeanette Miriam Lorenz for providing me with the unique opportunity to work at Fraunhofer IKS, an institute that truly lies at the intersection of industry and academia and taught me a lot about both sides. In addition, I wish to express my gratitude for the time she invested in my work, despite having an extremely busy schedule. Next, deep gratitude goes out to my co-supervisor Theodora-Augustina Drăgan, who has consistently made time for all my questions, was always open for discussions and gave incredibly valuable feedback. Her insights, patience and continuous support have motivated me throughout the project, and the result would not have been half of what it is without her. Furthermore, I would like to thank my internal supervisor Andreas Walther for his support and valuable feedback, as well as my colleagues from the QAI team for providing valuable insights into their topics of expertise. Additionally, I thank my examiners and the administrative staff from Lund University for their support and flexibility, which was required more than once. I also wish to thank my dearest friends (you know who you are) for their encouragement and listening ear whenever I needed one, celebrating my successes and sharing my frustrations. Finally, heartfelt gratitude goes out to my parents, who have tirelessly supported all my endeavors throughout the years and have always showed genuine interest in my work, despite the distance that separates us. I would not have made it this far without them.

References

[1] Oswaldo Zapata. A short introduction to quantum computing for physicists. *arXiv preprint arXiv:2306.09388*, 2023.

[2] Theodora-Augustina Drăgan. Characteristics of quantum architectures in reinforcement learning applications. Master's thesis, Technical University of Munich, 2022. Retrieved from https://publica.fraunhofer.de/entities/publication/c462c5a8-df4e-4955-a0cb-687d36803dcf/details.

[3] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219, 1996.

[4] Frank Verstraete, Jeroen Dehaene, Bart De Moor, and Henri Verschelde. Four qubits can be entangled in nine different ways. *Physical Review A*, 65(5):052112, 2002.

[5] Alba Cervera-Lierta, José Ignacio Latorre, and Dardo Goyeneche. Quantum circuits for maximally entangled states. *Physical Review A*, 100(2), aug 2019.

[6] Sergey Bravyi, Oliver Dial, Jay M. Gambetta, et al. The future of quantum computing with superconducting qubits. *Journal of Applied Physics*, 132(16), oct 2022.

[7] Frank Arute, Kunal Arya, Ryan Babbush, et al. Quantum supremacy using a programmable superconducting processor. *Nature*, 574:505–510, 2019.

[8] Ville Bergholm, Josh Izaac, Maria Schuld, et al. Pennylane: Automatic differentiation of hybrid quantum-classical computations. *arXiv preprint arXiv:1811.04968*, 2022.

[9] Yvonne Y. Gao, M. Adriaan Rol, Steven Touzard, et al. Practical guide for building superconducting quantum devices. *PRX Quantum*, 2(4), 2021.

[10] Yuhui Wang. Analysis on the mechanism of superconducting quantum computer. *Journal of Physics: Conference Series*, 1634(1):012040, sep 2020.

[11] Steven A. Moses, Charles H. Baldwin, Michael S. Allman, et al. A race track trapped-ion quantum processor. *arXiv preprint arXiv:2305.03828*, 2023.

[12] Dominic Widdows, Jyoti Rani, and Emmanuel M. Pothos. Quantum circuit components for cognitive decision-making. *Entropy*, 25(4), 2023.

[13] Colin D. Bruzewicz, John Chiaverini, Robert McConnell, et al. Trapped-ion quantum computing: Progress and challenges. *Applied Physics Reviews*, 6(2), may 2019.

[14] Loïc Henriet, Lucas Beguin, Adrien Signoles, et al. Quantum computing with neutral atoms. *Quantum*, 4:327, sep 2020.

[15] Lars Madsen, Fabian Laudenbach, Mohsen Askarani, et al. Quantum computational advantage with a programmable photonic processor. *Nature*, 606:75–81, 06 2022.

[16] Sara Bartolucci, Patrick Birchall, Hector Bombin, et al. Fusion-based quantum computation. *Nature Communications*, 14(1):912, 2023.

[17] Sergei Slussarenko and Geoff J. Pryde. Photonic quantum information processing: A concise review. *Applied Physics Reviews*, 6(4), oct 2019.

[18] Conor E. Bradley, Joe Randall, Mohamed H. Abobeih, et al. A ten-qubit solid-state spin register with quantum memory up to one minute. *Physical Review X*, 9(3):031045, 2019.

[19] Adam Kinos, David Hunger, Roman Kolesov, et al. Roadmap for rare-earth quantum computing. *arXiv preprint arXiv:2103.15743*, 2021.

[20] Miroslav Urbanek, Benjamin Nachman, Vincent R. Pascuzzi, et al. Mitigating depolarizing noise on quantum computers with noise-estimation circuits. *Physical Review Letters*, 127(27), dec 2021.

[21] Jin-Sung Kim, Lev S. Bishop, Antonio D. Có rcoles, et al. Hardware-efficient random circuits to classify noise in a multiqubit system. *Physical Review A*, 104(2), aug 2021.

[22] Miroslav Urbánek, Benjamin Philip Nachman, Vincent R. Pascuzzi, et al. Mitigating depolarizing noise on quantum computers with noise-estimation circuits. *Physical review letters*, 127 27:270502, 2021.

[23] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, oct 1997.

[24] Alberto Peruzzo, Jarrod McClean, Peter Shadbolt, et al. A variational eigenvalue solver on a photonic quantum processor. *Nature Communications*, 5(1), jul 2014.

[25] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. A quantum approximate optimization algorithm. *arXiv preprint arXiv:1411.4028*, 2014.

[26] M. Cerezo, Andrew Arrasmith, Ryan Babbush, et al. Variational quantum algorithms. *Nature Reviews Physics*, 3(9):625–644, Aug 2021.

[27] Zhenyu Cai, Ryan Babbush, Simon C. Benjamin, et al. Quantum error mitigation. *arXiv preprint arXiv:2210.00921*, 2022.

[28] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

[29] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3):279–292, May 1992.

[30] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015.

[31] Guoming Huang, Xiaofang Yuan, Zhixian Liu, et al. Deep reinforcement learning-based multi-objective path planning on the off-road terrain environment for ground vehicles. *arXiv preprint arXiv:2305.13783*, 2023.

[32] XiaoDan Wu, RuiChang Li, Zhen He, et al. A value-based deep reinforcement learning model with human expertise in optimal treatment of sepsis. *NPJ Digital Medicine*, 6(1):15, 2023.

[33] Haoran Guan. Self-inspection method of unmanned aerial vehicles in power plants using deep q-network reinforcement learning. *arXiv preprint arXiv:2303.09013*, 2023.

[34] Lokesh Chandra Das and Myounggyu Won. Lcs-tf: Multi-agent deep reinforcement learning-based intelligent lane-change system for improving traffic flow. *arXiv preprint arXiv:2303.09070*, 2023.

[35] Inhwan Kim, Sarvar Hussain Nengroo, and Dongsoo Har. Reinforcement learning for navigation of mobile robot with lidar. In *2021 5th International Conference on Electronics, Communication and Aerospace Technology (ICECA)*, pages 148–154. IEEE, 2021.

[36] Bumgeun Park, Jihui Lee, Taeyoung Kim, et al. Kick-motion training with dqn in ai soccer environment. In *2023 International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*, pages 689–692. IEEE, 2023.

[37] Yuhong Deng, Xiaofeng Guo, Yixuan Wei, et al. Deep reinforcement learning for robotic pushing and picking in cluttered environment. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019.

[38] Theresa Eimer, Carolin Benjamins, and Marius Lindauer. Hyperparameters in contextual rl are highly situational. *arXiv preprint arXiv:2212.10876*, 2022.

[39] Jonathan Wei Zhong Lau, Kian Hwee Lim, Harshank Shrotriya, et al. NISQ computing: where are we and where do we go? *Association of Asia Pacific Physical Societies Bulletin*, 32(1):27, December 2022.

[40] Kishor Bharti, Alba Cervera-Lierta, Thi Ha Kyaw, et al. Noisy intermediate-scale quantum algorithms. *Rev. Mod. Phys.*, 94:015004, Feb 2022.

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

[42] Dirk Heimann, Hans Hohenfeld, Felix Wiebe, and Frank Kirchner. Quantum deep reinforcement learning for robot navigation tasks. *arXiv preprint arXiv:2202.12180*, 2022.

[43] Maria Schuld, Ryan Sweke, and Johannes J. Meyer. Effect of data encoding on the expressive power of variational quantum-machine-learning models. *Phys. Rev. A*, 103:032430, Mar 2021.

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

[45] Maniraman Periyasamy, Marc Hölle, Marco Wiedmann, et al. Batch quantum reinforcement learning. *arXiv preprint arXiv:2305.00905*, 2023.

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

[47] Kosuke Mitarai, Makoto Negoro, Masahiro Kitagawa, and Keisuke Fujii. Quantum circuit learning. *Physical Review A*, 98(3):032309, 2018.

[48] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[49] Sofiene Jerbi, Casper Gyurik, Simon Marshall, et al. Parametrized quantum policies for reinforcement learning. *Advances in Neural Information Processing Systems*, 34:28362–28375, 2021.

[50] Mohannad M. Ibrahim, Hamed Mohammadbagherpoor, Cynthia Rios, et al. Evaluation of parameterized quantum circuits with cross-resonance pulse-driven entanglers. *IEEE Transactions on Quantum Engineering*, 3:1–13, 2022.

[51] Yalin Liao and Junpeng Zhan. Expressibility-enhancing strategies for quantum neural networks. *arXiv preprint arXiv:2211.12670*, 2023.

[52] Thomas Hubregtsen, Josef Pichlmeier, Patrick Stecher, et al. Evaluation of parameterized quantum circuits: on the relation between classification accuracy, expressibility, and entangling capability. *Quantum Machine Intelligence*, 3(9), 2021.

[53] Sukin Sim, Peter D. Johnson, and Alá n Aspuru-Guzik. Expressibility and entangling capability of parameterized quantum circuits for hybrid quantum-classical algorithms. *Advanced Quantum Technologies*, 2(12):1900070, oct 2019.

[54] Maria Schuld, Alex Bocharov, Krysta M. Svore, et al. Circuit-centric quantum classifiers. *Physical Review A*, 101(3), mar 2020.

[55] David A. Meyer and Nolan R. Wallach. Global entanglement in multiparticle systems. *Journal of Mathematical Physics*, 43(9):4273–4278, aug 2002.

[56] Amira Abbas, David Sutter, Christa Zoufal, Aurelien Lucchi, et al. The power of quantum neural networks. *Nature Computational Science*, 1(6):403–409, jun 2021.

[57] Frederik Kunstner, Philipp Hennig, and Lukas Balles. Limitations of the empirical fisher approximation for natural gradient descent. *Advances in neural information processing systems*, 32, 2019.

[58] Ryo Karakida, Shotaro Akaho, and Shun-ichi Amari. Universal statistics of fisher information in deep neural networks: Mean field approach. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 1032–1041. PMLR, 2019.

[59] Oksana Berezniuk, Alessio Figalli, Raffaele Ghigliazza, and Kharen Musaelian. A scale-dependent notion of effective dimension. *arXiv preprint arXiv:2001.10872*, 2020.

[60] Jason W. Rocks and Pankaj Mehta. Memorizing without overfitting: Bias, variance, and interpolation in overparameterized models. *Phys. Rev. Res.*, 4:013201, Mar 2022.

[61] Adrià Puigdomènech Badia, Bilal Piot, Steven Kapturowski, et al. Agent57: Outperforming the atari human benchmark. In *International conference on machine learning*, pages 507–517. PMLR, 2020.

[62] Theodora-Augustina Drăgan, Maureen Monnet, Christian B. Mendl, and Jeanette Miriam Lorenz. Quantum reinforcement learning for solving a stochastic frozen lake environment and the impact of quantum architecture choices. *arXiv preprint arXiv:2212.07932*, 2022.

[63] Owen Lockwood and Mei Si. Reinforcement learning with quantum variational circuit. In *Proceedings of the AAAI conference on artificial intelligence and interactive digital entertainment*, volume 16, pages 245–251, 2020.

[64] Samuel Yen-Chi Chen. Quantum deep q learning with distributed prioritized experience replay. *arXiv preprint arXiv:2304.09648*, 2023.

[65] Samuel Yen-Chi Chen. Asynchronous training of quantum reinforcement learning. *arXiv preprint arXiv:2301.05096*, 2023.

[66] Samuel Yen-Chi Chen, Chao-Han Huck Yang, Jun Qi, Pin-Yu Chen, et al. Variational quantum circuits for deep reinforcement learning. *arXiv preprint arXiv:1907.00397*, 2020.

[67] Nico Meyer, Daniel D. Scherer, Axel Plinge, et al. Quantum natural policy gradients: Towards sample-efficient reinforcement learning. *arXiv preprint arXiv:2304.1357*, 2023.

[68] Niels M. P. Neumann, Paolo B. U. L. de Heer, and Frank Phillipson. Quantum reinforcement learning: Comparing quantum annealing and gate-based quantum computing with classical deep reinforcement learning. *Quant. Inf. Proc.*, 22(2):125, 2023.

[69] BAQIS Quafu Group. Quafu-rl: The cloud quantum computers based quantum reinforcement learning. *arXiv preprint arXiv:2305.17966*, 2023.

[70] Eva Andrés, Manuel Pegalajar Cuéllar, and Gabriel Navarro. On the use of quantum reinforcement learning in energy-efficiency scenarios. *Energies*, 15(16), 2022.

[71] El Amine Cherrat, Snehal Raj, Iordanis Kerenidis, et al. Quantum deep hedging. *arXiv:2303.16585*, 2023.

[72] Dipesh Niraula, Jamalina Jamaluddin, Martha Matuszak, et al. Quantum deep reinforcement learning for clinical decision support in oncology: application to adaptive radiotherapy. *Scientific Reports*, 11, 12 2021.

[73] Faiza Gul, Wan Rahiman, and Syed Sahal Nazli Alhady. A comprehensive study for robot navigation techniques. *Cogent Engineering*, 6(1):1632046, 2019.

[74] Christoforos Mavrogiannis, Francesca Baldini, Allan Wang, Dapeng Zhao, Pete Trautman, Aaron Steinfeld, and Jean Oh. Core challenges of social robot navigation: A survey. *ACM Transactions on Human-Robot Interaction*, 12(3):1–39, 2023.

[75] René BM de Koster. Automated and robotic warehouses: developments and research opportunities. *Logistics and Transport*, 38(2):33–40, 2018.

[76] Ismot Sadik Peyas, Zahid Hasan, Md Rafat Rahman Tushar, Al Musabbir, Raisa Mehjabin Azni, and Shahnewaz Siddique. Autonomous warehouse robot using deep q-learning. In *TENCON 2021-2021 IEEE Region 10 Conference (TENCON)*, pages 857–862. IEEE, 2021.

[77] Adithya Balachandran, Anil Lal, and Pramod Sreedharan. Autonomous navigation of an amr using deep reinforcement learning in a warehouse environment. In *2022 IEEE 2nd Mysore Sub Section International Conference (MysuruCon)*, pages 1–5. IEEE, 2022.

[78] Gokul Subramanian Ravi, Kaitlin N Smith, Pranav Gokhale, and Frederic T Chong. Quantum computing in the cloud: Analyzing job and machine characteristics. In *2021 IEEE International Symposium on Workload Characterization (IISWC)*, pages 39–50. IEEE, 2021.

[79] Erwin Coumans. Real-time collision detection and multi-physics simulation for vr, games, visual effects, robotics, machine learning etc., 2023.

[80] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning, 2016–2023.

[81] Tyson Jones and Julien Gacon. Efficient calculation of gradients in classical simulations of variational quantum algorithms. *arXiv preprint arXiv:2009.02823*, 2020.

[82] IBM Quantum. Computing resources, 2023. Data retrieved from IBM Quantum Resources, `https://quantum-computing.ibm.com/services/resources`. Date of access: 02/10/2023.

[83] Antonin Raffin, Ashley Hill, Adam Gleave, et al. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.

[84] Ben Poole, Subhaneil Lahiri, Maithra Raghu, et al. Exponential expressivity in deep neural networks through transient chaos. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

[85] Yasuhiro Fujita and Shin-ichi Maeda. Clipped action policy gradient. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1597–1606. PMLR, 10–15 Jul 2018.

[86] Philip Easom-Mccaldin, Ahmed Bouridane, Ammar Belatreche, et al. On depth, robustness and performance using the data re-uploading single-qubit classifier. *IEEE Access*, 9:65127–65139, 2021.

[87] Li Ding and Lee Spector. Multi-objective evolutionary architecture search for parameterized quantum circuits. *Entropy*, 25(1):93, jan 2023.

[88] Chih-Chieh Chen, Masaya Watabe, Kodai Shiba, et al. On the expressibility and overfitting of quantum circuit learning. *ACM Transactions on Quantum Computing*, 2(2), jul 2021.

[89] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 30(1), Mar. 2016.

Appendix

I Table of Common Quantum Gates

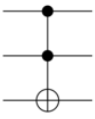
Operator	Gate(s)	Matrix
Pauli-X (X)		$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$
Pauli-Y (Y)		$\begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$
Pauli-Z (Z)		$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$
Hadamard (H)		$\frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$
Phase (S, P)		$\begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}$
$\pi/8$ (T)		$\begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix}$
Controlled Not (CNOT, CX)		$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$
Controlled Z (CZ)		$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$
SWAP		$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
Toffoli (CCNOT, CCX, TOFF)		$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$

Figure 22: Overview of common quantum logic gates.

II Trainable Parameter Count of All Models

	8x8	16x16	32x32	64x64
CNN-1	131	387	1283	4611
CNN-2	155	435	1379	4803
CNN-3	195	515	1539	5123
CNN-4	187	499	1507	5059

Table 3: Number of parameters of the classical neural networks (CNN) for each input configuration (rows) (see Table 1) and architecture of the hidden layers (columns), denoted as m x n, with m and n the number of neurons for the first and second hidden layer, respectively.

	1	2	5	10
QNN-1	27	45	99	189
QNN-2a	30	51	114	219
QNN-2b	42	66	138	258
QNN-3a	36	63	144	279
QNN-3b	48	78	168	318

Table 4: Number of parameters of the quantum neural networks (QNN) for each input configuration (rows) (see Table 1) per number of layers. The fully connected classical layer at the end is not included.

III Comparison of Results

3x3 Environment

Size	SR		ASTC	
	Thesis	Paper	Thesis	Paper
8x8	0.2	0.85	49090	22630
16x16	0.4	1	45330	13960
32x32	0.6	1	35390	10860
64x64	0.7	0.85	37910	8670

4x4 Environment

Size	SR		ASTC	
	Thesis	Paper	Thesis	Paper
8x8	0.2	0.8	42400	29750
16x16	0.1	0.85	49020	14730
32x32	0.1	0.8	47780	14920
64x64	0.5	1	37010	11430

5x5 Environment

Size	SR		ASTC	
	Thesis	Paper	Thesis	Paper
8x8	0.0	-	50000	-
16x16	0.0	0.4	50000	43140
32x32	0.0	0.85	50000	32780
64x64	0.0	1	50000	26390

Table 5: An overview of the success rate (SR) and average number of steps to convergence (ASTC) from this thesis and the paper by Heimann et al. [42].