DEPARTMENT OF ELECTRICAL ENGINEERING

TECHNICAL UNIVERSITY MUNICH

Master's Thesis in Electrical Engineering

Quantum-Enhanced Deterministic Control for Continuous Robot Navigation Tasks

Alexander Künzner

DEPARTMENT OF ELECTRICAL ENGINEERING

TECHNICAL UNIVERSITY MUNICH

Master's Thesis in Electrical Engineering

Quantum-Enhanced Deterministic Control for Continuous Robot Navigation Tasks

Quantenverstärkte Deterministische Steuerung für Kontinuierliche Aufgaben der Roboternavigation

Author:	Alexander Künzner
External Supervisor:	MSc. Theodora-Augustina Drăgan
Internal Supervisor:	Prof. Dr. Robert Wille
Submission Date:	15 October 2024

Abstract

This thesis studies whether integrating variational quantum circuits (VQC) into reinforcement learning (RL) frameworks can address inherent challenges classical RL face - scaling to continuous action and state spaces in more industrial-like environments that rely only on local feature generation. It aims to determine if quantum and hybrid models offer performance advantages and parameter efficiency over classical neural networks (NN) in simulated robot navigation tasks. An advanced RL environment is implemented, simulating continuous robot navigation with local sensory inputs like LiDAR, velocity, and orientation to closely mimic real-world conditions. A dynamic reward system encourages consistent progress and penalizes inefficiencies. Within the deep deterministic policy gradient (DDPG) framework, three function approximators are investigated: classical NNs and two VQC-based ansatzes, hybrid and quantum. Each is evaluated across multiple VQC setups by varying the qubit count and the number of layers to determine effects on performance and scalability. The ansatzes are then compared against classical NNs with a similar number of parameters to assess relative performance and parameter efficiency. Training and testing of all 117 models are conducted in a simulated robot navigation environment to systematically compare performance, aiming to uncover the strengths and limitations of integrating quantum computing into RL for robotic applications. The results demonstrate that quantum-enhanced models using VQCs successfully learn continuous robot navigation tasks using only local features in three environments of varying difficulty. Increasing the depth of the VQCs, particularly by adding more layers, substantially improves performance, with deeper quantum circuits (QC) outperforming simpler ones. Hybrid models provide balanced performance, especially in more complex environments, by showing characteristics of quantum and classical approaches. Some quantum models exhibit superior parameter efficiency, achieving comparable or superior results to classical models, though with significantly fewer trainable parameters. Findings confirm the potential of quantum reinforcement learning (QRL) to address challenges in continuous robot navigation tasks, indicating possible quantum advantages in parameter efficiency. However, practical implementation on real quantum hardware remains challenging due to current hardware limitations and noise constraints. Further research is needed to explore advanced algorithms and address issues like hyper-parameter sensitivity and barren plateaus to fully realize the potential of quantum-enhanced RL in robotics.

Contents

1. Introduction

Reinforcement learning (RL) has emerged as a powerful framework for solving complex decision-making problems in recent years, with applications ranging from game-playing artificial intelligence (AI) [1] to autonomous robotics [2, 3]. Despite its success, RL faces significant challenges when scaling to high-dimensional state and action spaces [4]. Traditional algorithms often employ semi-efficient strategies to optimize computational resources or limit research scenarios to discrete settings [5], restricting state and action spaces to non-realistic environments. Furthermore, the current RL literature frequently utilizes Atari 2600 games as standard benchmarks [6], which incorporate globally generated features instead of local ones, highlighting the limitation of RL to idealized environments. As neural networks (NN) increase in size to address growing problem complexities [7], the number of weights - and consequently the computational resources required - also increases. In contrast to these limitations, quantum computing has attracted increasing attention due to its potential to solve specific classes of problems polynomially or exponentially faster than classical computers [8, 9], as demonstrated by Shor's [10] and Grover's algorithms [11]. QC leverages principles of quantum mechanics, such as superposition and entanglement, to process information in fundamentally new ways [8]. This has led to the exploration of quantum-enhanced machine learning (QML) methods, including quantum reinforcement learning (QRL), which aims to exploit quantum computational advantages to improve RL performance [9].

This thesis explores the integration of QC into RL, focusing specifically on the application of variational quantum circuits (VQC) within the RL framework as function approximators. By leveraging quantum mechanics principles, this work addresses inherent challenges in classical RL, including the lack of more industrial-like environments, the application of function approximation in continuous action and state spaces, and the reliance on local feature generation. Additionally, it seeks to tackle the growing computational requirements of NNs for more complex problems.

The contributions of this thesis provide insights into the scalability and hybridization performances of different quantum-enhanced ansatzes that build upon the deep deterministic policy gradient (DDPG) algorithm to learn policies in robot navigation scenarios across three distinct environments. Thirteen different architectures based on quantum, hybrid, and classical ansatzes are compared across 117 models, including 1,170 evaluation runs. The work demonstrates the feasibility of quantum-enhanced function approximators for complex robotics tasks. It confirms that VQC-based models can successfully learn continuous robot navigation in a simulated environment using only local features. The results also imply potential quantum advantages in parameter efficiency compared to a classical baseline, with a hybrid ansatz balancing characteristics of both quantum and classical methods.

The thesis is structured as follows: Chapter 2 introduces the theory of RL, providing an overview of theory and algorithms focusing on actor-critic models. The fundamentals of QC are discussed in Chapter 3, including the foundational workings of quantum physics within the QML frame. Chapter 4 merges insights from RL and QC to provide context on the spectrum of QRL methods and related work applying VQC-based techniques for RL. Moving on to the limitations of current RL techniques, Chapter 5 introduces the RL environment and circuits used in this work. Chapter 6 analyzes and summarizes the training and test results, including discussions of trends and interpretations of the three approaches. Finally, Chapter 7 concludes the thesis by placing its insights into a broader scope and providing recommendations for future research directions.

2. Reinforcement Learning

2.1. Fundamentals

Supervised learning, unsupervised learning, and RL are three fundamental paradigms within ML, each differing in how they utilize data. In supervised learning, models are trained on labeled datasets, where the input data is paired with the correct output, allowing the model to learn a mapping from inputs to outputs. On the other hand, unsupervised learning deals with unlabeled data, focusing on identifying patterns or structures within the data, such as clustering or dimensionality reduction [12]. RL diverges from both by placing an agent to interact with an environment to learn optimal behaviors (policies) through trial-and-error [13], where each interaction serves as sampled data that informs the learning process of the agent.

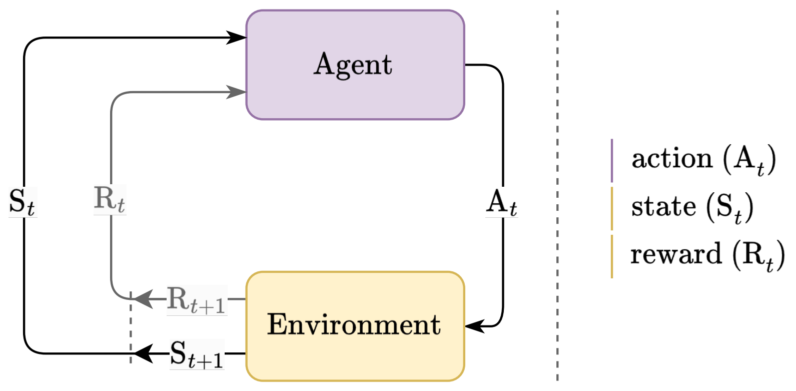


Figure 2.1.: Reinforcement learning loop: agent-environment interaction [12].

In RL, the learning process pivots around an agent, the decision-maker, that interacts with an environment to learn a policy $\pi : S \rightarrow A$, which maps states $s_t \in S$ to actions $a_t \in A$ [12, 14]. The primary objective of the policy is to maximize the cumulative reward, also known as the return, which is the total reward an agent accumulates over time [13]. This dynamic interaction between the agent and the environment is vital to the RL process, as it mirrors the adaptive nature of learning from experience [12]. The learning process can be broken down into several steps, as shown in Figure 2.1. Note that agent and policy are often used interchangeably as "the policy is . . . the agent's brain" [15, p. 33].

- **State (S_t):** At each time step t, the agent observes the current state S_t of the environment. The state represents the current situation or context in which the agent finds itself, including features or conditions relevant to the task at hand. Feature elements of an observed state could be the coordinates of a robot, its orientation, or sensor measurements.
- **Action (A_t):** Based on the observed state S_t, the agent chooses an action A_t. The action is selected according to the agent's current policy π, which determines actions to take in response to different states.
- **Environment Response:** After the agent takes action A_t, the environment transitions to a new state S_{t+1}. This transition is influenced by the agent's action and the underlying dynamics of the environment, including noise.
- **Reward (R_t):** As a result of the action taken, the agent receives feedback in the form of a reward R_t. The reward R_t, which can be positive (a reward) or negative (a penalty), is a scalar value that establishes a measurement of how favorable the outcome of the action was in terms of achieving the agent's general objective.
- **Loop Continuation:** The process then repeats, with the agent using the new observed state S_{t+1} to select the subsequent action A_{t+1}, receiving a new reward R_{t+1}, and so on. This loop continues until a terminating condition is met, such as reaching a goal state, a maximum number of steps, or convergence to an optimal policy. Over time, the agent's policy is updated to maximize its cumulative reward, improving its actions to become more effective.

The agent-environment interaction described above forms the basis of RL. However, RL is typically framed within the mathematical structure of a Markov decision process (MDP) [16, 1, 13, 12]. The details of this framework are discussed in the following section.

2.1.1. Markov Decision Processes

MDPs are a fundamental mathematical framework for modeling sequential decision-making problems [13]. Although there exist various types and classifications of MDPs, such as partially observable MDPs (POMDP) [16, 13], constrained, and risk-aware MDPs [17], this thesis focuses on standard MDPs and their mathematical framework. In real-world scenarios, standard MDPs often fall short due to noisy sensor readings or other factors unknown to the agent, making extensions like POMDPs necessary. However, this thesis will be limited to the standard MDP framework, which forms the core of RL by defining the interaction between an agent and its environment over discrete time steps [1]. An MDP is characterized by a set of states, actions, transition probabilities, and rewards, which collectively describe the environment in which the agent operates [3]. The content in Section 2 is primarily based on the works of [12] and [13].

While many MDP extensions address various complexities, they differ in the nature of their state and action spaces. According to [13], MDPs can be classified into finite and infinite categories. Finite MDPs have a limited number of states and actions, meaning the agent encounters only a finite set of distinct states and can choose from a finite set of actions [1]. In contrast, infinite MDPs involve state or action spaces that are unbounded or continuous, typical in scenarios such as a robot's physical positioning or the continuous adjustment of a steering wheel. The following key components formally define an MDP:

1. **State Space ($\mathcal{S}$)**: The state space $\mathcal{S}$ represents all possible situations or configurations in which the agent can find itself within the environment. A state $S_t \in \mathcal{S}$ at time step t contains all the information necessary for decision-making. In a finite MDP, $\mathcal{S}$ consists of a limited number of discrete states. In contrast, in an infinite MDP, $\mathcal{S}$ may represent a continuous set corresponding to infinite possible states, such as a robot's position and orientation.

2. **Action Space ($\mathcal{A}$)**: The action space $\mathcal{A}$ consists of all possible actions the agent can perform. For a given state $S_t \in \mathcal{S}$, the agent selects an action $A_t \in \mathcal{A}(S_t)$ based on its policy, which is a decision-making strategy. In finite MDPs, the set of actions $\mathcal{A}$ is finite and discrete. In infinite MDPs, the action space can be continuous, such as choosing a real-valued speed or steering angle.

3. **Joint State-Reward Probabilities ($p(s', r \mid s, a)$)**: The joint state-reward probability function defines the probability of transitioning to a state s' and receiving a reward r, given that the agent was in state s and took action a. This function is represented as [12]:

$$p(s', r \mid s, a) = \Pr\{S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a\}, \tag{2.1}$$

   where $s' \in \mathcal{S}$, $r \in \mathcal{R}$, $s \in \mathcal{S}$, and $a \in \mathcal{A}$. The joint probability describes the combined interactions of the environment and the reward structure in RL. It captures the likelihood of arriving at a specific state while receiving a certain reward.

This probability integrates the transitions between states and the corresponding rewards, reflecting the complete outcome of an action chosen by a policy π.

4. **State-Transition Probabilities ($p(s' \mid s, a)$)**: The state-transition probabilities describe the likelihood of moving from one state to another given a specific action taken by the agent. These probabilities can be derived from the joint state-reward probabilities (cf. Equation 2.1) by summing over all possible rewards $r \in \mathcal{R}$:

$$p(s' \mid s, a) = \sum_{r \in \mathcal{R}} p(s', r \mid s, a). \tag{2.2}$$

   In this context, $p(s' \mid s, a)$ represents the probability of transitioning to state s' given that the agent was in state s and took action a. These state-transition probabilities allow the agent to predict the consequences of its actions and adjust its policy accordingly, which is essential for learning optimal behaviors in RL.

5. **Expected Rewards ($r(s, a)$ and $r(s, a, s')$)**: The expected rewards can be represented in two different ways, depending on the level of detail required for the decision-making process:

   - **Expected Reward for State-Action Pairs ($r(s, a)$)**: This function represents the expected reward received after taking action a in state s. It is defined as [13]:

$$r(s, a) = \mathbb{E}[R_t \mid S_{t-1} = s, A_{t-1} = a] = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p(s', r \mid s, a). \tag{2.3}$$

     Here, $r(s, a)$ captures the expected reward over all possible state transitions and associated rewards, providing a scalar measurement of the desirability of an action in a given state. This is useful in RL for value-based methods, which involve estimating the value of taking an action in a particular state to make informed decisions about which actions to take to maximize future rewards.

   - **Expected Reward for State-Action-Next-State Triples ($r(s, a, s')$)**: This function provides a more detailed view by considering a specific transition to the next state s'. It is given by [12]:

$$r(s, a, s') = \mathbb{E}[R_t \mid S_{t-1} = s, A_{t-1} = a, S_t = s'] = \sum_{r \in \mathcal{R}} r \frac{p(s', r \mid s, a)}{p(s' \mid s, a)}. \tag{2.4}$$

     This representation of the expected reward helps understand the immediate outcomes of specific transitions, allowing more granular control in policy design and optimization. In contrast, the expected reward for a state-action pair, denoted as $r(s,a)$, considers the average reward over all possible transitions resulting from taking action a in state s. It is calculated by summing over all possible next states and their associated rewards, weighted by the state-transition probabilities. This difference is significant in RL because $r(s,a,s')$ focuses on the reward given a particular next state, allowing the agent to differentiate between the outcomes of specific transitions. Model-based Algorithms, which aim to predict and optimize over exact state transitions rather than general expectations, benefit from this more detailed expected reward representation.

In an MDP, the agent aims to maximize its long-term cumulative reward, defined as the total sum of rewards received over time [3]. This cumulative reward can be specified in two ways:

- **Cumulative Reward**: The sum of rewards until a terminal time step T, suitable for tasks with natural endpoints:

$$G_t = \sum_{k=0}^{T-t} R_{t+k+1}, \tag{2.5}$$

- **Discounted Cumulative Reward**: Future rewards are discounted by a factor $\gamma \in [0,1)$ to reflect their present value [13], suitable for continuing tasks:

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}. \tag{2.6}$$

In summary, the MDP framework provides a complete mathematical model for decision-making in environments where outcomes are partly random and partly under the control of an agent. By specifying state and action spaces, transition and reward probabilities, and expected rewards, MDPs formally model the interactions between an agent and its environment. This structure is crucial in RL, laying the foundation for understanding how agents learn from their actions to maximize cumulative rewards over time. Modeling and analyzing these components allows for developing algorithms that can effectively handle various environments, whether finite or infinite, discrete or continuous.

With a solid understanding of the MDP framework, the focus now shifts to the agents themselves, particularly the policies that guide their decision-making processes. The next section will explore policies, how agents use them to select actions based on the current state, and how these policies can be optimized to achieve the best possible outcomes.

2.1.2. Policies and Value Functions

The agent is the decision-maker that interacts with the environment, learns from it, and makes decisions to maximize cumulative rewards [13]. The policy is a central component of the agent's decision-making process, defining the agent's behavior by mapping states to actions. Formally, a policy π is a function $\pi : \mathcal{S} \to \mathcal{A}$, where $\pi(a|s)$ denotes the probability of taking action a when in state s [13]. The notation used for a policy depends on its type. A stochastic policy, for example, is typically denoted by π, where actions are selected according to a probability distribution, i.e., $A_t \sim \pi(\cdot|S_t)$. In contrast, a deterministic policy is usually denoted by μ, where actions are a specific state function, represented as $A_t = \mu(S_t)$ [15]. This chapter will use the notation π to represent a policy in general contexts. The appropriate notation will be used when necessary (e.g., in Section 2.3.2).

Another key concept in RL is the value function, which estimates an agent's expected cumulative reward from a given state following a particular policy. The state-value function $v_\pi(s)$ for a policy π is defined as [12]:

$$v_\pi(s) = \mathbb{E}_\pi \left[G_t \mid S_t = s \right] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s \right], \tag{2.7}$$

where G_t represents the cumulative reward from time step t, and γ is the discount factor that balances the importance of immediate and future rewards. Similarly, the action-value function, or Q-function, $q_\pi(s,a)$, represents the expected cumulative reward starting from state s, taking action a, and thereafter following policy π [13]:

$$q_\pi(s,a) = \mathbb{E}_\pi \left[G_t \mid S_t = s, A_t = a \right] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a \right]. \tag{2.8}$$

RL algorithms can be broadly classified based on how they utilize the policy and value functions. Policy-based methods optimize the policy by adjusting its parameters to maximize expected rewards, while value-based methods focus on estimating the value functions to guide the agent's actions indirectly. Some algorithms combine both approaches, known as actor-critic methods, where the policy (actor) and value function (critic) are learned simultaneously [18].

According to [19], the Bellman equation is a fundamental component of dynamic programming and plays a crucial role in RL algorithms. This equation breaks down the value function into recursive subproblems, which helps optimize sequential decision-making processes. The Bellman equation is based on the principle of optimality, which states that an optimal policy must ensure that, regardless of the starting state and initial action, the remaining decisions form an optimal policy relative to the state resulting from the first decision. In RL, this equation provides a method to calculate the value of a state or a state-action pair by considering the expected cumulative rewards, balancing immediate rewards and future gains. This creates a recursive relationship that facilitates iterative improvements of value estimates over time, allowing the agent to learn optimal strategies for interacting with the environment. The Bellman equation for the state-value function $v_\pi(s)$ under a policy π is given by [12]:

$$v_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s] = \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma v_\pi(s')], \tag{2.9}$$

for all $s \in \mathcal{S}$. This equation expresses the value of a state s as the expected sum of immediate rewards and the discounted value of the subsequent states, assuming the agent follows policy π. It captures the trade-off between taking an immediate action and the future benefits of states that result from that action. Similarly, the Bellman equation for the action-value function $q_\pi(s,a)$ under a policy π is defined as [12]:

$$q_\pi(s,a) = \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] = \sum_{s',r} p(s',r|s,a)[r + \gamma q_\pi(s',a')], \tag{2.10}$$

for all $(s,a) \in \mathcal{S} \times \mathcal{A}$. This equation calculates the expected return of taking action a in state s and then following policy π. It helps evaluate how good a specific action is at a given state under the policy.

Value functions are used to evaluate and compare policies. A policy π is considered better than or equal to another policy π' if it yields a greater or equal expected (discounted) return for all states. Mathematically, this means $v_\pi(s) \geq v_{\pi'}(s)$ for all states $s \in \mathcal{S}$. An optimal policy, represented as π^*, achieves the highest value for every state, as defined by the optimal value function:

$$v^*(s) = \max_\pi v_\pi(s), \tag{2.11}$$

for all $s \in \mathcal{S}$. Similarly, the optimal action-value function $q^*(s,a)$ is defined as:

$$q^*(s,a) = \max_\pi q_\pi(s,a), \tag{2.12}$$

for all $(s,a) \in \mathcal{S} \times \mathcal{A}$. The Bellman optimality equations for these optimal value functions define the value of a state or state-action pair under an optimal policy by relating it to the values of subsequent states, assuming the best actions are taken [13]:

$$v^*(s) = \max_a \sum_{s',r} p(s',r \mid s,a)[r + \gamma v^*(s')], \tag{2.13}$$

$$q^*(s,a) = \sum_{s',r} p(s',r \mid s,a)[r + \gamma \max_{a'} q^*(s',a')], \tag{2.14}$$

for all $s \in \mathcal{S}$ and $(s,a) \in \mathcal{S} \times \mathcal{A}$, respectively. Equations 2.13 and 2.14 are foundational in RL because they enable dynamic programming methods to iteratively update value functions. By utilizing these recursive relationships, algorithms such as policy iteration and value iteration systematically improve value estimates, eventually converging to the optimal value function. This approach enables the agent to make optimal decisions by assessing the long-term impact of its actions and maximizing cumulative (discounted) rewards.

In summary, this section has outlined the key concepts of policies and value functions, which are essential for decision-making in RL. Understanding these components is crucial for developing algorithms that can effectively learn and adapt to various environments. The subsequent section will explore how these concepts are applied in different RL algorithms, emphasizing the methods used to update value functions based on the agent's experiences.

2.2. Learning Methods

RL involves various methods agents use to learn and improve their decision-making strategies. These methods can be broadly categorized based on how they update policies and value functions, which are critical for balancing exploration (discovering the environment) and exploitation (making use of gained knowledge) [12]. Understanding the distinctions between these methods is necessary for selecting the appropriate RL algorithm for a given task.

2.2.1. Model-Free and Model-Based Methods

RL methods can be divided into two main categories: *model-free* and *model-based* methods. Model-free methods do not rely on a model of the environment to make decisions. Instead, they learn directly from environmental interactions by updating the policy or value function based on observed experiences, typically represented as state-action-reward-next state tuples [13]. These tuples, commonly called "experiences", capture the state of the environment, the action taken by the agent, the reward received, and the subsequent state reached. Model-free methods are generally simpler to implement and are suitable for environments where building an accurate model is difficult or impossible. Examples of model-free algorithms include Q-learning and policy gradient methods [15].

In contrast, model-based methods employ a model of the environment to predict future states and rewards, which allows the agent to plan by simulating different action sequences and evaluating their outcomes. This can lead to more sample-efficient learning, as the agent can explore potential scenarios without directly interacting with the environment [15]. Model-based algorithms are often more complex but can be advantageous in environments where accurate models are available or can be learned effectively [15]. Examples of model-based methods include AlphaZero [20].

2.2.2. On-Policy and Off-Policy Learning

The terms *on-policy* and *off-policy* describe how agents learn from their experiences. On-policy learning updates the policy used to make decisions based on the actions taken by the agent under the current policy [13]. An example of this method is SARSA (State-Action-Reward-State-Action) [12], which learns the value of the policy currently being executed by the agent. On-policy learning is beneficial in dynamic environments where the policy must adapt continuously to changing conditions, as it encourages exploration by incorporating the agent's actions into policy updates [12].

Off-policy learning methods, such as Q-learning, learn the value of an optimal policy independently from the behavior policy's actions to explore the environment. This flexibility allows off-policy methods to learn from data generated by different policies [13]. It makes them more sample-efficient and suitable for environments where collecting new data is costly or impractical [12, 18]. Off-policy learning is advantageous when the goal is to optimize a policy without being constrained by the policy used for exploration [13].

2.2.3. Value-Based Methods

Value-based methods in RL focus on estimating value functions to guide decision-making. These methods work by approximating the value of states or state-action pairs and using these estimates to derive an optimal policy [12]. Two primary value-based methods are Monte Carlo (MC) and temporal-difference (TD) methods.

MC methods estimate value functions by averaging the returns obtained from complete episodes [12]. This model-free approach can be applied in both on- and off-policy settings. In the former, the value function is updated based on returns from episodes generated under the policy being followed by the agent, ensuring that learning remains consistent with the agent's experiences. On the other hand, off-policy MC methods estimate the value function for a different target policy by using returns generated by a behavior policy, which allows learning from diverse experiences [12, 13].

Unlike MC methods, TD methods update value estimates based on both observed rewards and the estimated value of the next state, allowing them to bootstrap updates. This makes TD methods more sample-efficient as they do not need to wait for an episode to complete before making updates. Examples of TD methods include SARSA and Q-learning. SARSA is an on-policy TD algorithm that updates the action-value function based on the actions taken by the current policy [12]. In contrast, Q-learning is an off-policy TD algorithm that updates the action-value function using the maximum estimated value of the next state, regardless of the action taken by the current policy [12, 18].

2.2.4. Policy Gradient Methods

Policy gradient methods directly optimize the policy by adjusting the parameters of a policy function that maps states to actions. Unlike value-based methods, which optimize the policy indirectly through value estimation, policy gradient methods aim to maximize the expected return directly by following the gradient of the performance measure [12]. This gradient, denoted as $\nabla_\theta J(\theta)$, represents the direction in which the policy parameters θ should be adjusted to increase the expected cumulative reward. By sampling trajectories from the policy and using them to estimate this gradient, policy gradient methods update the parameters in the direction that maximizes performance. The parameter update rule for policy gradient methods is given by [18]:

$$\theta_{t+1} = \theta_t + \alpha \nabla_\theta J(\theta)\big|_{\theta=\theta_t} \tag{2.15}$$

where θ_t represents the parameters at iteration t, α is the learning rate, and $\nabla_\theta J(\theta)$ is the gradient of the expected return with respect to the parameters θ. These methods are particularly useful for environments with continuous action spaces [3] or when the policy cannot be easily derived from value functions, as they can effectively handle high-dimensional or stochastic policies [18].

2.2.5. Actor-Critic Methods

Actor-critic methods combine elements of both value-based and policy gradient methods [13]. The actor component represents the policy, selecting actions, while the critic component evaluates these actions by estimating the value function. This interaction between the actor and critic is illustrated in Figure 2.2, where the actor is responsible for policy improvement and the critic for policy evaluation through value function approximation. This combination allows actor-critic methods to benefit from the stability and efficiency of value-based methods while integrating the direct optimization of policy gradient methods. Examples include DDPG [5], advantage actor-critic (A2C) [21], and proximal policy optimization (PPO) [22], which are commonly used in scenarios requiring robust policy improvement and sample efficiency [18].

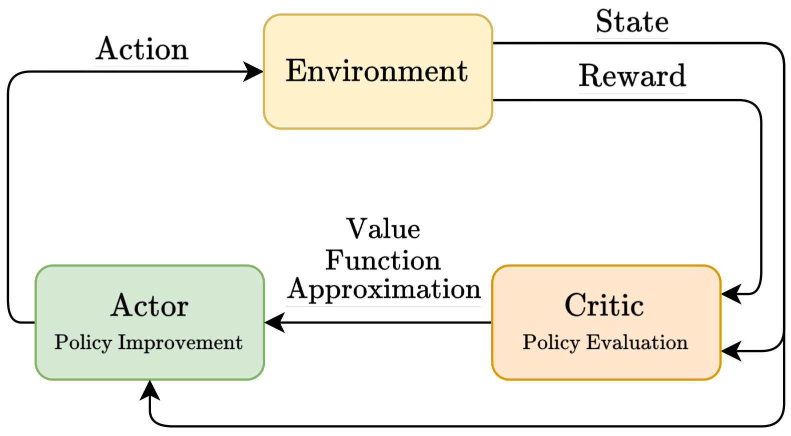


Figure 2.2.: Simplified actor-critic interaction [18].

2.2.6. Monte Carlo vs. Temporal-Difference Methods

Monte Carlo and TD methods represent two fundamental approaches in value-based RL. Monte Carlo methods rely on complete episodes to update value estimates, which can lead to high variance due to their dependency on full trajectories but remain unbiased [12]. In contrast, by incorporating bootstrapping, TD methods reduce variance by using the estimated value of the next state for updates but introduce bias as they rely on other estimated values for learning [12, 18]. This trade-off between bias and variance makes TD methods more sample-efficient, especially in environments where full episodes are long or impractical to simulate.

2.2.7. Overview of Concepts

Table 2.1 provides an overview of various RL methods. The table uses the abbreviations MF for model-free, MB for model-based, PO for policy optimization, and Q-L for Q-learning. Abbreviations for the algorithms can be found in Table A.1. AlphaZero learns from simulated experiences generated by its own policy through Monte Carlo tree search (MCTS) [20]. Thus, it does not fit into conventional policy-type classifications and is labeled as N/A. This categorization helps give an overview of current RL methods, aiding in understanding the landscape of RL and the variety of approaches available for different applications.

As outlined in Section 2.2, RL algorithms can be distinguished by various characteristics, including the type of model, policy, and method used. The choice of approach depends on the specific task and the environment in which the agent operates. While this section provided a general overview of these characteristics, it is not exhaustive. The following section will explore solution methods for RL problems, starting with the simpler tabular solutions and then moving on to approximate solutions.

Algorithm	**Category**	**Learning Approach**	**Policy Type**
A2C / A3C	MF	PO	On-Policy
PPO	MF	PO	On-Policy
TRPO	MF	PO	On-Policy
DDPG	MF	PO, Q-L	Off-Policy
TD3	MF	PO, Q-L	Off-Policy
SAC	MF	PO, Q-L	Off-Policy
DQN	MF	Q-L	Off-Policy
C51	MF	Q-L	Off-Policy
QR-DQN	MF	Q-L	Off-Policy
HER	MF	Q-L	Off-Policy
World Models	MB	Learn the model	Off-Policy
I2A	MB	Learn the model	Off-Policy
MBMF	MB	Learn the model	Off-Policy
MBVE	MB	Learn the model	Off-Policy
AlphaZero	MB	Given the Model	N/A

Table 2.1.: Overview of RL algorithms [15].

2.3. Solution Methods

This section explores different solution methods for RL problems. The first approach discussed is tabular methods, suitable for environments with small, discrete state and action spaces [18]. These methods allow for a straightforward representation of value functions, making them effective in simple scenarios. However, as the complexity of the environment increases, tabular methods become impractical [23].

To address more complex problems, approximate solutions are introduced. These methods use function approximators to handle larger or continuous state and action spaces where tabular methods fail. Among the various approximation techniques, special attention is given to DDPG, as employed in the robotic navigation case studied in this thesis. DDPG combines deep learning and RL elements to effectively operate in high-dimensional and continuous action spaces, making it a suitable choice for robotic control tasks.

2.3.1. Tabular Solutions

Tabular solutions are a fundamental approach in RL where a table, often called a Q-table or value table, is used to store and update the value of each state-action pair. This method is feasible when dealing with environments that have small, discrete state and action spaces, where all possible state-action combinations can be explicitly represented and updated [18]. An example where tabular solutions are practical is grid-based environments, where an agent navigates a grid to reach a goal, and both the states (grid positions) and actions (moving up, down, left, or right) are finite and discrete.[1]

Methods such as Q-learning and SARSA are effective in these simpler environments as they provide an exhaustive exploration of the state space, ensuring the agent learns the optimal policy through iterative updates. However, this approach becomes impractical in environments with larger or continuous state and action spaces due to the exponential growth in the table size required to store all state-action values. In such cases, approximations are necessary to efficiently handle the complexity of the environment.

2.3.2. Approximate Solutions

In RL, exact solutions are often infeasible due to the complexity of the environment or the high-dimensional state and action spaces. Thus, approximate solutions are unavoidable in achieving practical results. This thesis applies the DDPG algorithm, an approximate solution method for continuous action spaces, to address the robot navigation problem.

[1]See `https://github.com/Farama-Foundation/Minigrid` for examples.

DDPG

DDPG is an actor-critic method, a class of algorithms that combines value-based methods and policy gradient methods to improve learning stability and efficiency [24, 5]. As described in Section 2.2.5 and illustrated in Figure 2.2, actor-critic methods utilize two components: the actor, which represents the policy and selects actions, and the critic, which evaluates these actions by estimating the value function. DDPG extends these principles to continuous action spaces, making it suitable for robotic control tasks. The pseudo-code of DDPG is shown in Algorithm 1.

DDPG employs an experience replay buffer (RB), denoted as $\mathcal{D}$, to handle the complexity of continuous action spaces and improve training efficiency. The RB stores transitions, or experiences, (s_t, a_t, r_t, s_{t+1}) collected during the agent's interaction with the environment. This approach allows DDPG to break the temporal correlations between consecutive experiences by sampling random minibatches of transitions from the RB when updating the networks [25, 1]. By learning from various past experiences rather than relying solely on the most recent ones, the RB facilitates off-policy learning. This stabilizes the learning process and avoids divergence, especially in complex environments with continuous actions [1].

Actor-critic methods like DDPG extend the concepts of the Bellman optimality equations (cf. Section 2.1.2) to continuous action spaces and adapt them to meet the stability needs of gradient-based methods. In DDPG, the policy update is conducted by moving in the direction of the gradient of the action-value function q, rather than performing a global maximization of q, which can be computationally intensive in high-dimensional spaces [24, 5]. The deterministic policy $\mu_\theta(s)$ is updated using the rule:

$$\theta_{t+1} = \theta_t + \alpha \nabla_\theta q(s,a)\big|_{a=\mu_\theta(s)}, \tag{2.16}$$

which ensures improvement by adjusting the parameters to increase the expected return.

Given that the action space in DDPG is continuous, it is assumed that the function $q^*(s,a)$ (cf. Equation 2.14) is differentiable with respect to the action a. This allows a gradient-based approach to efficiently update the policy $\mu(s)$. Instead of solving an optimization problem to compute $\max_a q(s,a)$, this is approximated by $q(s, \mu(s))$, with the gradient moving toward the optimal Q-value [15]. To minimize the Mean Squared Bellman Error (MSBE), a stochastic gradient descent method is employed. The MSBE loss function for the Q-learning component of DDPG is defined as:

$$L(\phi, \mathcal{D}) = \mathbb{E}_{(s,a,r,s',d)\sim\mathcal{D}}\left[\left(q_\phi(s,a) - \left(r + \gamma(1-d)q_{\phi_{\text{targ}}}(s', \mu_{\theta_{\text{targ}}}(s'))\right)\right)^2\right], \tag{2.17}$$

where $\mathcal{D}$ represents the RB, d is a binary indicator that is 1 if the episode ends at state s' and 0 otherwise, and $\mu_{\theta_{\text{targ}}}$ denotes the target policy [5].

To ensure stability during training, DDPG employs two sets of function approximators: online networks (critic: ϕ, actor: θ) and target networks (critic: ϕ_{targ}, actor: θ_{targ}). The target networks serve as slowly updated copies of the online networks, providing more stable target Q-values for the learning process. Gradually updating the target network parameters reduces the risk of oscillations or divergence in the estimated Q-values, as rapid changes can destabilize the learning process [5]. The target networks are updated using a technique known as Polyak averaging, which adjusts the network parameters incrementally toward the online network parameters [15]:

$$\phi_{\text{targ}} \leftarrow (1-\tau)\phi_{\text{targ}} + \tau\phi, \tag{2.18}$$
$$\theta_{\text{targ}} \leftarrow (1-\tau)\theta_{\text{targ}} + \tau\theta, \tag{2.19}$$

where τ is a small hyper-parameter, typically set to around 0.005. A small τ ensures that the target network parameters are updated predominantly by their previous values, with only a minor influence from the current online network parameters. Gradual updates help maintain stable learning by providing consistent target values and reducing the risk of oscillations or divergence in the estimated Q-values [5].

Because DDPG uses a deterministic policy, it relies on off-policy learning to encourage adequate exploration. This is accomplished by adding noise to the actions during training, essential for preventing insufficient exploration that can lead to suboptimal learning, especially in the early stages. Although Ornstein-Uhlenbeck (OU) noise was initially preferred, recent studies have shown that uncorrelated Gaussian noise with a zero mean is equally effective [15]. Therefore, this work adopts Gaussian noise with zero mean to enhance exploration and support the development of a robust policy [5].

In summary, DDPG combines deterministic policy gradients with the actor-critic framework to learn policies in continuous action spaces, making it well-suited for the robot navigation problem addressed in this work. DDPG enables efficient learning of navigation strategies in complex environments by directly optimizing the policy to maximize expected returns. The hyper-parameters used for the robot navigation task are listed in Table A.1.

Training stability is maintained through target networks, updated slowly relative to the online networks to prevent disruptive changes, thus stabilizing policy development. Minimizing the MSBE via stochastic gradient descent ensures accurate value estimates

Algorithm 1 DDPG [5]

```
Initialize (randomly) critic network $q_\phi(s,a)$ and actor $\mu_\theta(s)$ with weights $\phi$ and $\theta$
Initialize target networks $q_{\phi_{\text{targ}}}$ and $\mu_{\theta_{\text{targ}}}$ with weights $\phi_{\text{targ}} \leftarrow \phi$, $\theta_{\text{targ}} \leftarrow \theta$
Initialize replay buffer $\mathcal{D}$
for episode = 1, M do
    Initialize a random process $\mathcal{N}$ for action exploration
    Receive initial observation state $s_1$
    for t = 1, T do
        Select action $a_t = \mu_\theta(s_t) + \mathcal{N}_t$ according to the current policy
            and exploration noise
        Execute action $a_t$ and observe reward $r_t$, new state $s_{t+1}$, and termination $d_t$
        Store transition $(s_t, a_t, r_t, s_{t+1}, d_t)$ in $\mathcal{D}$
        Sample a random minibatch of $N$ transitions $(s_i, a_i, r_i, s_{i+1}, d_i)$ from $\mathcal{D}$
        Set $y_i = r_i + \gamma(1 - d_i) q_{\phi_{\text{targ}}}(s_{i+1}, \mu_{\theta_{\text{targ}}}(s_{i+1}))$
        Update critic by minimizing the loss:
            $L = \frac{1}{N}\sum_i (y_i - q_\phi(s_i, a_i))^2$
        Update the actor policy using the sampled policy gradient:
            $\nabla_\theta J \approx \frac{1}{N}\sum_i \nabla_a q_\phi(s,a)\big|_{s=s_i, a=\mu_\theta(s_i)} \nabla_\theta \mu_\theta(s)\big|_{s=s_i}$
        Update the target networks:
            $\phi_{\text{targ}} \leftarrow (1-\tau)\phi_{\text{targ}} + \tau\phi$
            $\theta_{\text{targ}} \leftarrow (1-\tau)\theta_{\text{targ}} + \tau\theta$
    end for
end for
```

for effective navigation decisions. The experience RB also allows the agent to learn from diverse past navigation experiences. This off-policy approach, combined with exploration noise techniques, enhances the agent's ability to thoroughly explore the state-action space and learn robust navigation policies.

This concludes the chapter on classical RL, which has covered various fundamental algorithms and methods. The next chapter introduces the principles of quantum computing, providing an overview of its core concepts and potential capabilities. Subsequently, the focus shifts to the intersection of quantum computing and RL, exploring how quantum algorithms might transform decision-making processes in complex environments.

3. Quantum Computing

In this chapter, the concept of quantum computing is introduced and explored in the context of its application to ML. The intersection of these two fields, known as QML, is a rapidly growing area of research [26]. Traditionally, NNs are used as function approximators when exact solutions are computationally impractical or unknown (cf. Section 2.3.2). This chapter, however, introduces an alternative approach by presenting the fundamentals of quantum computing, followed by a discussion on how quantum computing can be harnessed to enhance ML techniques. In particular, the chapter explores the role of parameterized quantum circuits (PQC) as function approximators.

3.1. Fundamentals

Quantum computing represents a novel computational paradigm that exploits the principles of quantum mechanics as its underlying foundation [27]. Unlike classical computers, where each bit is limited to representing one of two values (0 or 1), quantum bits (qubits) - the fundamental unit of information in a quantum system [26] - can simultaneously represent 2^N values. This ability stems from the phenomenon of superposition, a defining property of qubits [26, 28]. A qubit can exist in a superposition, meaning it holds a probability of being found in the state $|0\rangle$, $|1\rangle$, or a combination of both, described mathematically using Dirac notation [28]. The state of a qubit in superposition can be represented as a linear combination of basis states, typically expressed as:

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle\,, \tag{3.1}$$

where the coefficients $\alpha, \beta \in \mathbb{C}$ are complex numbers representing the probability amplitudes for the respective states $|0\rangle$ and $|1\rangle$ [28, 29]. The probability of finding the qubit in state $|0\rangle$ or $|1\rangle$ is given by squaring the absolute value of these coefficients. Therefore, they must satisfy the normalization condition:

$$|\alpha|^2 + |\beta|^2 = 1, \tag{3.2}$$

which ensures that the total probability of finding the qubit in one of its possible states is equal to one [10, 28]. In Dirac notation, the basis states $|0\rangle$ and $|1\rangle$ can also be expressed as vectors in the computational basis. Specifically:

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}. \tag{3.3}$$

The number of possible states grows exponentially for multiple qubit systems [28]. For instance, in a 3-qubit system ($N = 3$), there are 2^3 possible states. The state of the system is:

$$|\psi\rangle = \alpha_1 |000\rangle + \alpha_2 |001\rangle + \alpha_3 |010\rangle + \cdots + \alpha_8 |111\rangle, \tag{3.4}$$

and the normalization condition from Equation 3.2 expands accordingly:

$$\sum_{i=1}^{2^N} |\alpha_i|^2 = 1. \tag{3.5}$$

More generally, for a system of N qubits, the total state can be expressed as a superposition of all possible computational basis states. This is written as:

$$|\phi\rangle = \sum_{i=0}^{2^N-1} |i\rangle = \sum_{b_0,b_1,\ldots,b_{N-1}\in\{0,1\}} |b_0 b_1 \ldots b_{N-1}\rangle, \tag{3.6}$$

where $|i\rangle$ represents the computational basis states, and the summation runs over all possible bit strings $b_0, b_1, \ldots, b_{N-1}$, with $b_i \in \{0, 1\}$. This generalizes the previous example to an arbitrary number of qubits, where the superposition contains all the possible states of the system.

Another fundamental property of quantum mechanics is entanglement. Entanglement describes a phenomenon of two qubits becoming intertwined such that their descriptions cannot be separated [29, 28]. In this intertwined state, the properties of each qubit depend on one another, making it impossible to factor the system into distinct components [29]. Consequently, the combined state ($|\psi\rangle_{AB}$) of the system cannot be expressed as a simple product of the states of each qubit ($|\psi\rangle_A$, $|\psi\rangle_B$) [30]:

$$|\psi\rangle_{AB} \neq |\psi\rangle_A \otimes |\psi\rangle_B. \tag{3.7}$$

Moreover, measuring one qubit's state immediately affects the other's state, regardless of the distance between them [29]. Entanglement thus creates a strong correlation between the qubits, enabling new mechanisms that classical bits cannot replicate. Maximally entangled quantum states, known as Bell states, exemplify this phenomenon. For instance, in one type of Bell state, a pair of entangled qubits can exist simultaneously in the state $|00\rangle$ or $|11\rangle$, expressed mathematically as [30]:

$$|\psi_{\text{entangled}}\rangle = \frac{1}{\sqrt{2}} \left(|00\rangle + |11\rangle\right). \tag{3.8}$$

Bell states are essential in applications such as quantum teleportation and quantum cryptography [28], although these concepts are beyond the scope of this thesis.

As previously described in Equation 3.1, the general state of a single qubit can also be written in terms of trigonometric functions:

$$|\psi\rangle = \cos\left(\frac{\theta}{2}\right)|0\rangle + e^{i\phi}\sin\left(\frac{\theta}{2}\right)|1\rangle\,, \tag{3.9}$$

where θ and ϕ are real numbers, annotated as spherical coordinates that map the state onto the surface of the Bloch sphere [31, 28], as illustrated in Figure 3.1. The term $e^{i\phi}$ represents a phase factor and introduces a phase shift in the superposition of the components $|0\rangle$ and $|1\rangle$.

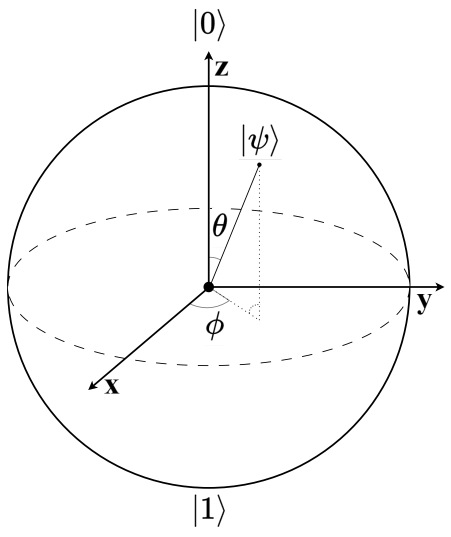


Figure 3.1.: Bloch sphere representation of a state vector.

Comparing Equation 3.1 and 3.9, one can deduce:

$$\alpha = \cos\left(\frac{\theta}{2}\right), \quad \beta = e^{i\phi}\sin\left(\frac{\theta}{2}\right). \tag{3.10}$$

The geometrical representation of the state space of a qubit is especially useful for tracking the evolution of quantum states over time [32]. This evolution can be visualized as a rotation of the Bloch vector, which points from the sphere's center to its surface, illustrating the qubit's state [32, 31]. Such rotation can be expressed mathematically as a unitary transformation, realized through quantum gates, the building block of QCs.

Contrary to manipulating bits via classical logic, quantum gates manipulate qubits by changing their phases and probabilities. Examples of such gates are listed in Table 3.1. A quantum computer, analogous to a classical computer, operates based on multiple quantum gates, which together form QCs [28]. A simple example of such a circuit is illustrated in Figure 3.2.

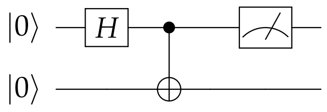


Figure 3.2.: A simple quantum circuit generating the Bell state from Equation 3.8.

Initially, the two-qubit system is in a separable state since the qubits are not entangled, as opposed to the case in Equation 3.7. The joint state of the qubits is expressed as $|\psi_0\rangle = |0\rangle \otimes |0\rangle = |00\rangle$. A Hadamard gate is then applied to the first qubit, putting it into an equal superposition, transforming the state as follows: $H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$. This results in the system state:

$$|\psi_1\rangle = H|0\rangle \otimes |0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes |0\rangle$$

which can be expanded as:

$$|\psi_1\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |10\rangle).$$

A controlled-NOT (CNOT) gate operates on two qubits, where the first qubit acts as the control and the second qubit serves as the target, flipping the target qubit's state only when the control qubit is in the $|1\rangle$ state [28] (cf. Table 3.1). Therefore, applying the CNOT results in the following state change:

$$|\psi_2\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle).$$

Finally, after the unitary operations transform the quantum state, a measurement is performed to extract classical information. This is indicated as the meter symbol on the wire of the first qubit. In quantum mechanics, the measurement of an observable O on a quantum state $|\psi\rangle$ yields an expectation value, denoted as [33]:

$$\langle O\rangle_\psi = \langle\psi|O|\psi\rangle.$$

The measurement collapses the superposition of the qubits into one of the basis states, with probabilities determined by the quantum state's probability amplitudes. For instance, in the case of the Bell state (Equation 3.8), measuring the first qubit will collapse the system into either $|00\rangle$ or $|11\rangle$, each with a probability of $\frac{1}{2}$. The role of measurement in this thesis is limited to extracting classical data from quantum systems as needed. For a more detailed discussion on quantum measurement, refer to [28, 34, 35].

While the previously discussed circuit is a relatively small example, accumulating a larger number of quantum gates enables the implementation of more complex algorithms. Leveraging the unique properties of superposition and entanglement, quantum computers can process information in parallel [36, 37] and establish strong correlations between qubits, resulting in computational speedups that surpass those of classical computers [38, 28, 29]. Notable algorithms demonstrating such speedups include Shor's algorithm for factoring large integers - a computationally challenging task for classical computers [10, 30] - and Grover's algorithm, which achieves a quadratic performance improvement in search-based problems [11, 28]. While several quantum algorithms offer speedups over classical methods in specific areas, the question of whether quantum computing provides a significant advantage for ML tasks remains open [26, 28]. The following sections will explore the applicability of quantum computation in the domain of ML in greater depth.

3.2. Quantum Machine Learning

Building upon the fundamentals of quantum computing, this section explores how quantum computers can be utilized in the domain of QML. Given the computational speedups offered by certain quantum algorithms, it is of considerable interest whether applying quantum computation in ML could lead to similar improvements. Previous works have demonstrated such speedups in specific problems within QML [39, 40, 41], suggesting potential advantages over classical approaches. However, the question of whether QML can provide significant and practical advantages across a broad range of ML tasks remains open.

This is especially true considering the limited availability and practicality of current quantum computers, which remain significant challenges [28]. Even IBM's quantum hardware produces inconsistent results across repeated experiments, as shown in [37]. These inconsistencies stem from several key issues, including fault tolerance [42], noise and decoherence effects, and hardware limitations, which characterize the current era

Gate Name	Symbol	Description	Matrix Notation
Identity	I	Leaves qubit unchanged	$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$
Pauli-X	X	Bit-flip gate (like classical NOT)	$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$
Pauli-Y	Y	Bit-flip + phase flip gate	$\begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$
Pauli-Z	Z	Phase-flip gate	$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$
Hadamard	H	Creates superposition by applying a uniform rotation	$\frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$
Phase Shift	R_ϕ	Adds a phase of ϕ to the $\|1\rangle$ state	$\begin{bmatrix} 1 & 0 \\ 0 & e^{i\phi} \end{bmatrix}$
CNOT	X or $\oplus$	Controlled-NOT gate; flips the second qubit if the first qubit is $\|1\rangle$	$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$
Unitary	$U(\theta, \phi, \lambda)$	General unitary transformation for a single qubit	$\begin{bmatrix} \cos\left(\frac{\theta}{2}\right) & -e^{i\lambda}\sin\left(\frac{\theta}{2}\right) \\ e^{i\phi}\sin\left(\frac{\theta}{2}\right) & e^{i(\phi+\lambda)}\cos\left(\frac{\theta}{2}\right) \end{bmatrix}$
Rotation-X	$R_x(\theta)$	Rotation by angle θ around the X-axis	$\begin{bmatrix} \cos\left(\frac{\theta}{2}\right) & -i\sin\left(\frac{\theta}{2}\right) \\ -i\sin\left(\frac{\theta}{2}\right) & \cos\left(\frac{\theta}{2}\right) \end{bmatrix}$
Rotation-Y	$R_y(\theta)$	Rotation by angle θ around the Y-axis	$\begin{bmatrix} \cos\left(\frac{\theta}{2}\right) & -\sin\left(\frac{\theta}{2}\right) \\ \sin\left(\frac{\theta}{2}\right) & \cos\left(\frac{\theta}{2}\right) \end{bmatrix}$
Rotation-Z	$R_z(\theta)$	Rotation by angle θ around the Z-axis	$\begin{bmatrix} e^{-i\theta/2} & 0 \\ 0 & e^{i\theta/2} \end{bmatrix}$

Table 3.1.: Common quantum gates with descriptions, symbols, and matrix representations.

of quantum computing - often referred to as the noisy intermediate-scale quantum (NISQ) era [37, 28, 33]. NISQ devices are said to have between 50 to 100 qubits [26, 43], which, while impressive, remain far from what is required for large-scale, fault-tolerant quantum computing [44, 45].

Quantum systems can be realized using diverse qubit technologies, including spin systems [46], ion traps [47], neutral atoms [48], and superconducting circuits, the latter being the most prominent candidate [27]. Despite these advances, the noise from quantum gates and decoherence still limits the scalability and reliability of current quantum systems, leading to deviations from theoretical expectations [37, 33]. To address these challenges, researchers are focusing on developing approaches that maximize the utility of QCs despite these limitations. As a result, hybrid quantum-classical algorithms have gained prominence for their ability to tolerate the noisy conditions of NISQ devices while still leveraging quantum parallelism [49]. PQCs have emerged as useful tools in this context. A general QML model is shown in Figure 3.3, where classical data is encoded into quantum states via the feature map $U_{\phi(x)}$, followed by a trainable unitary transformation U_Θ. This basic structure underlies the functioning of PQCs.

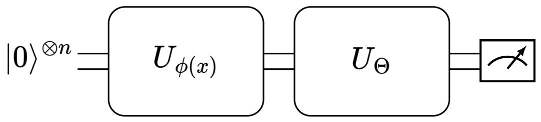


Figure 3.3.: A schematic setup of a QML model [50].

PQCs are QCs that contain tunable parameters, which can be optimized using classical optimization algorithms. These circuits emerged during the NISQ era because they were designed to minimize circuit depth and qubit requirements, thus limiting the exposure to noise and decoherence effects. A PQC consists of layers of quantum gates, with only specific gates parametrized by adjustable variables. The parameters are iteratively updated using classical optimization methods, such as gradient descent, to minimize a cost function relevant to a given problem, such as classification or regression [33]. This combination of classical optimization with quantum operations allows PQCs to perform complex tasks even on the limited qubit systems of NISQ devices. While both PQCs and VQCs rely on parameterized quantum gates and classical optimization, the literature refers to VQCs when the objective is to iteratively reduce a cost function. A general QML workflow is illustrated in Figure 3.4, which outlines the complete process. The classical data x is first passed through a feature map, represented by the unitary transformation $U_{\phi(x)}$, which encodes x into a quantum state. After encoding the classical data, the quantum state is processed by the VQC, which applies a trainable unitary transformation U_Θ. The output is then measured, producing a classical result z. The result is post-processed through a function $f(z)$, which is then fed into a classical optimizer. The optimizer updates the parameters θ, completing one iteration of the training loop. This iterative process continues to minimize a task-specific cost function.

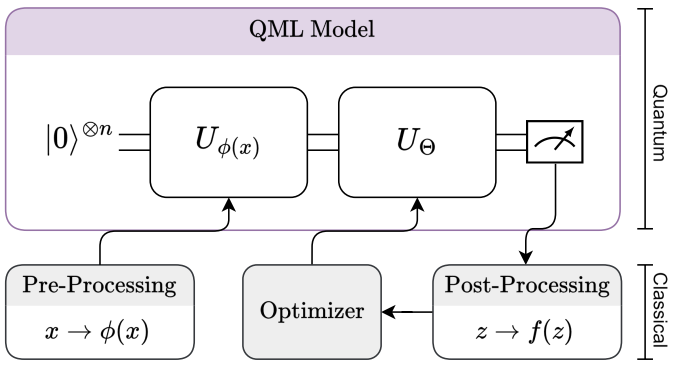


Figure 3.4.: Simplified workflow of a QML model [50].

3.2.1. QML Encoding Techniques

Various methods have been developed to encode classical data into quantum computers, mapping classical information to the Hilbert space of quantum states. This process, essential for ensuring the trainability of quantum models [51], represents the feature map $U_{\phi(x)}$, as shown in Figure 3.3.

Pauli feature mapping is an encoding technique that transforms classical data into quantum states using Pauli-Z rotations and controlled-Z (CZ) operations. Pauli-Z rotations modify the phase of individual qubits without altering their probability distributions, while CZ operations create entanglement by correlating pairs of qubits. This method is commonly utilized in algorithms such as quantum support vector machines and variational quantum classifiers [51]. Given a classical data vector $\mathbf{x} = [x_1, x_2, \ldots, x_n]$, the second-order Pauli feature map embeds this data into a quantum state by applying Pauli-Z rotations to each qubit and CZ entangling operations between qubits. The following unitary operation represents this mapping:

$$U_\phi(x) = \exp\left(i\left(\sum_i \phi_i(x) Z_i + \sum_{i<j} \phi_{ij}(x) Z_i Z_j\right)\right),$$

where Z_i is the Pauli-Z operator acting on the i-th qubit, and $\phi_i(x)$ and $\phi_{ij}(x)$ are data-dependent phase functions that encode classical data. These functions modify the phases of the qubits, embedding classical information into the quantum state [33]. For example, if the classical data vector is $\mathbf{x} = [x_1, x_2]$, the Pauli feature map produces the quantum state:

$$|\psi(x)\rangle = \exp(i\phi_1(x)Z_1)\exp(i\phi_2(x)Z_2)\exp(i\phi_{12}(x)Z_1Z_2)|0\rangle^{\otimes 2},$$

where $\phi_1(x)$ and $\phi_2(x)$ depend on x_1 and x_2, respectively, and $\phi_{12}(x)$ introduces entanglement between the qubits, correlating their states.

Another technique is amplitude encoding. This method is useful for embedding large datasets into a quantum state. The key concept behind amplitude encoding is normalizing the classical data vector and assigning its elements as the amplitudes of the quantum state's basis states. This encoding method can be employed in several quantum algorithms that rely on efficient data loading, such as QML or quantum differential equation solvers [52]. To illustrate amplitude encoding, consider a classical data vector $\mathbf{x} = [x_1, x_2, \ldots, x_n]$. By normalizing this vector such that $\sum_i |x_i|^2 = 1$ (cf. Equation 3.2 and 3.5), the conditions of a quantum state are satisfied. The classical data is then encoded into the quantum state as follows:

$$|\phi\rangle = x_1|00\rangle + x_2|01\rangle + \cdots + x_n|11\rangle.$$

Each classical value x_i is represented as an amplitude corresponding to the computational basis state $|i\rangle$ in this equation. If the number of data points is less than 2^N, where N is the number of qubits, padding can be applied to the data. This approach allows encoding up to 2^N data points into N qubits [51], exploiting quantum superposition for efficient data representation.

Lastly, Angle encoding is one of the most straightforward methods for embedding classical data into quantum states. In this approach, each classical data point is encoded as an angle of rotation applied to a qubit. In direct data encoding, the rotation angle θ_i is typically set by the value of the classical data x_i, such that $\theta_i = x_i$. This encoding is commonly achieved through rotation gates like R_x, R_y, or R_z, where the gate defines the axis of rotation, and the classical data determines the angle of rotation. Given an input data vector $\mathbf{x} = [x_1, x_2, \ldots, x_n]$, the corresponding quantum state is prepared using the following transformation:

$$|\psi(x)\rangle = \bigotimes_{i=1}^{n} R_y(x_i)|0\rangle.$$

Here, $R_y(x_i)$ represents a rotation around the y-axis (cf. Figure 3.1) by an angle x_i, encoding each data point as a rotation on the respective qubit. This method allows for encoding classical data with shallow circuit depth, minimizing computational complexity [53]. Figure 3.5 below illustrates a simple PQC that implements angle encoding using R_y rotation gates.

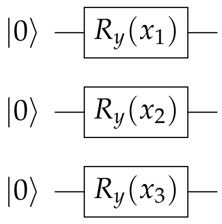


Figure 3.5.: A PQC implementing angle encoding with R_y rotation gates.

There are a variety of classical data encoding techniques available in QML, each with its own advantages and suitable use cases. These include basis encoding, Hamiltonian encoding, superdense encoding, and others, each providing different ways to map classical data onto quantum states. For a more comprehensive overview of these methods, refer to [54, 55, 56].

This chapter has provided a detailed introduction to the fundamentals of quantum computing, focusing on its applications within ML, particularly through using PQCs. Key quantum principles, such as superposition, entanglement, and quantum gates, were explored as the foundational elements enabling quantum computation's unique capabilities [26, 28]. The insights gained from quantum computing fundamentals and QML lay the groundwork for the next chapter, which introduces the primary focus of this thesis: QRL. This approach combines quantum computational advantages with classical RL techniques to address complex decision-making problems across diverse environments [33]. The subsequent discussion will expand on how VQCs are adapted to RL tasks, offering potential advancements over traditional approaches.

4. Quantum Reinforcement Learning

RL has seen considerable success in various fields, from gaming to robotics [41], by allowing agents to learn optimal policies through interactions with their environment. However, the growing complexity of modern problems - especially those involving large state-action spaces - poses significant challenges to classical RL [41]. With the advent of quantum computing, QRL has emerged as a promising paradigm that aims to utilize the computational power of quantum mechanics to accelerate and improve RL processes [57]. By exploiting the principles of superposition and entanglement, as discussed in the previous section, QRL has the potential to outperform classical RL methods in specific problem domains [49].

4.1. Quantum Kernels in RL

The QRL landscape encompasses a spectrum of approaches, ranging from quantum-inspired algorithms on classical hardware to fully quantum systems operating on universal, fault-tolerant quantum computers [33]. These methods vary in their integration of quantum and classical resources, offering different degrees of quantum-classical hybridization. With the rapid advancements in quantum hardware and the emergence of NISQ devices, hybrid quantum-classical approaches are particularly attractive for near-term applications, leveraging NISQ resources to enhance RL without requiring fully fault-tolerant quantum computers [58]. Table 4.1 provides an overview of these hybridization techniques and offers a comprehensive survey on QRL. The algorithm classes listed - namely *Quantum-Inspired RL Algorithms*, *VQC-based Function Approximation*, *RL Algorithms with Quantum Subroutines*, and *Full-Quantum RL* - each present a distinct balance between classical and quantum computation, tailored to available hardware and specific RL tasks. This section excludes tabular solutions (cf. Section 2.3.1) due to their infeasibility in addressing complex RL problems with large state or action spaces [23].

Quantum-Inspired RL Algorithms

At the classical end of the hybridization spectrum are quantum-inspired RL algorithms. While entirely executable on classical computing hardware, these algorithms are de-

	Classical Compute Resources	**NISQ Resources**	**Universal, Fault-Tolerant & Error-Corrected Quantum Processing Unit**	
	classical ←	**Degree of hybridization**		→ *quantum*
Algorithm Class	Quantum-inspired RL Algorithms	VQC-based Function Approximation	RL Algorithms with Quantum Subroutines	Full-Quantum RL
Subtype	Amplitude-Amplification-based Action Selection	Value-Function	Projective Simulation	Oracularized Environments
		Policy	Quantum Value Iteration	Quantum Gradient Estimation
		Actor-Critic	Quantum Policy Iteration	Quantum Policy Iteration
		Multi-Agent RL	Quantum Boltzmann-Machines for Function Approximation	
		Offline RL		

Table 4.1.: Overview of quantum and hybrid RL algorithms categorized by computational resources and degree of quantum-classical hybridization [33].

signed based on quantum mechanical principles such as superposition and amplitude amplification. They incorporate these concepts into classical RL models to improve task performance, such as exploration-exploitation balancing [49]. Despite their reliance on classical resources, these algorithms derive theoretical inspiration from quantum mechanics, influencing their design and operation.

One prominent subtype within this class is *Amplitude-Amplification-based Action Selection*. This method mirrors the concept of Grover's algorithm, where favorable actions are selectively amplified in probability based on the reward function [59]. The states and actions are embedded in a classical Hilbert space, and the amplitude of states that yield higher rewards is increased, thus improving the action selection process. Though this method is quantum-inspired, its implementation remains purely classical and does not require access to quantum hardware. Integrating these quantum concepts is more about enhancing classical models than providing actual quantum advantage [33].

VQC-based Function Approximation

As quantum hardware advances, VQCs have become prominent components of hybrid quantum-classical algorithms in RL tasks. In this hybrid framework, VQCs replace classical NNs in function approximation. The previously discussed learning methods in Section 2.2 can be adapted to use VQCs. The following three subtypes, namely *Policy*, *Value-Function*, and *Actor-Critic*, while serving the same objectives, are thus employed in a hybrid approximation.

Additionally, *Offline RL* uses pre-collected data to train RL agents without real-time interaction with the environment, similar to the model-based methods introduced in Section 2.2.1. QCs process these static datasets to derive value or policy approximations, extending the use of quantum resources in RL. This quantum adaptation of offline learning is well-suited for environments where data collection is costly or time-consuming, aligning with the advantages of classical model-based approaches regarding sample efficiency [33].

Lastly, using the same learning methods, the VQC-based approximation can also be employed in multi-agent reinforcement learning (MARL). MARL increases complexity by including several agents within one environment. Due to the different reward functions and policies learned by the various agents, uncertainty increases as the agents' actions are interdependent and more dynamic, rendering the state-action space nonstationary [60].

These QCs can be operated on NISQ devices by combining quantum components with classical optimization techniques. This allows the RL process to utilize quantum computation, even within current hardware constraints, without necessitating fully fault-tolerant quantum computers. These VQC-based methods, being adaptable to NISQ resources, already offer a glimpse into the potential for quantum advantage in RL, as they extend the classical learning frameworks described earlier with quantum-enhanced computation [61, 40].

RL Algorithms with Quantum Subroutines

Positioned further along the quantum-classical spectrum, *RL Algorithms with Quantum Subroutines* incorporate quantum algorithms as integral components within classical RL frameworks. These methods require more advanced quantum hardware and exploit quantum computing for policy iteration, value iteration, or sampling tasks. Unlike VQC-based approaches that utilize QCs as function approximators, these algorithms replace classical subroutines with their quantum equivalents.

One example is *Projective Simulation*, a model where learning is driven by stochastic processing of episodic memory through quantum random walks [45]. The agent operates in an environment where actions are rewarded based on sensor perceptions, and the core mechanism is the episodic and compositional memory (ECM), represented as a weighted, directed graph. Nodes in this graph, called clips, represent individual experiences and can be excited based on the agent's interaction with the environment. The learning process involves updating the ECM's structure by modifying connections between clips or adjusting weights, influencing future decisions [33]. This model-free approach leverages quantum random walks to simulate action selection, enabling faster exploration of the state-action space.

Another subtype, *Quantum Policy/Value Iteration*, replaces classical iterative methods to update policies or value functions. Techniques such as quantum natural policy gradients (QNPG) utilize quantum Fisher information matrices to precondition gradient updates, improving learning efficiency and potentially accelerating convergence [62]. By employing quantum properties, these algorithms enable RL agents to make more informed decisions faster, especially in high-dimensional or complex environments.

Quantum Boltzmann Machines (QBM) are another subtype within this class. These models extend classical Boltzmann machines by replacing classical spins with qubits, where the Hamiltonian governs the system's energy [63]. A key distinction is the dimensionality; the quantum version operates over a space exponentially larger than the classical one, corresponding to 2^N possible states for N qubits instead of N binary variables. The quantum framework allows QBMs to perform energy minimization and gradient descent more efficiently, contributing to faster learning. Although these methods still rely on classical resources for some computations, the incorporation of quantum subroutines for estimating distributions over quantum states moves QBMs closer to fully quantum approaches compared to VQC-based methods [63, 39].

Full-Quantum RL

At the far end of the quantum-classical hybridization spectrum are *Full-Quantum RL* algorithms, where all components of the RL process are executed on quantum hardware. These algorithms necessitate universal, fault-tolerant quantum processors, as they rely entirely on quantum computations [33]. In such settings, classical resources are minimal, if involved at all, and the RL agent operates fully within a quantum computational framework.

One subtype in this category is *Oracularized Environments*, where the RL agent interacts with a quantum oracle that simulates the environment [64]. This allows for faster state transitions and more efficient evaluations of policies, making the learning process quicker than in classical or hybrid environments [65].

Another subtype, *Quantum Gradient Estimation*, focuses on using quantum algorithms to compute the gradients needed for optimizing the parameters of QCs in policy networks. These quantum techniques offer more efficient gradient estimation than classical methods, particularly in high-dimensional parameter spaces [66].

The quantum-classical hybridization of RL algorithms reveals a diverse landscape where varying degrees of quantum integration are explored depending on the available hardware and specific RL tasks. From quantum-inspired classical methods to fully quantum RL systems, these approaches mark important steps toward realizing the potential of quantum computing in RL. As quantum hardware continues to evolve, the boundary between hybrid and fully quantum algorithms is expected to shift, paving the way for more widespread adoption of quantum techniques in ML. Despite these advances in hybrid quantum-classical models, most existing QRL research is limited to discrete action and state spaces. This limits the applicability of hybridizations in real-world problems where continuous action and state spaces are prevalent. In fields such as robotics [41], autonomous systems, and high-dimensional financial modeling, actions must be fine-tuned to respond to a broad range of possibilities rather than confined to a fixed set of choices. The following section presents state-of-the-art QRL applications developed over the past few years.

4.2. Related Work in VQC-based QRL

Building on the discussion of different quantum-classical hybridization techniques explored in the previous section, this section highlights state-of-the-art research and applications in VQC-based QRL. As VQCs are well-suited to the capabilities of current NISQ devices, they represent one of the most feasible approaches to implementing QRL in the near term. Much of the ongoing work in QRL focuses on integrating VQCs into hybrid models, taking advantage of quantum properties to enhance learning processes while still relying on classical optimization techniques. This section explores several prominent examples, underscoring the relevance of this approach to the contributions made in this thesis, which fall within the same domain.

[61] presents an early exploration of VQCs in deep reinforcement learning (DRL), adapting classical techniques like RB and target networks for quantum systems. The work demonstrates using VQCs as function approximators for Q-value learning, with the RB storing agent transitions as tuples and updating parameters via random batches from replay memory (cf. Section 2.3.2). The target network, updated periodically, stabilizes

training by providing a stationary Q-value function, enhancing training robustness. Their approach is tested in Frozen Lake and a cognitive radio setting. Frozen Lake, a simple, discrete environment part of the `OpenAI Gym` suite [67], provides a baseline for testing RL techniques, while the cognitive radio environment, though more complex, is noted as having comparable complexity. The QCs used in this study employed a four qubit system, representing 16 classical states via binary encoding and rotations. In the Frozen Lake experiment, the agent converged to an optimal policy after 200 episodes, with stable performance. The cognitive radio experiment demonstrated faster convergence, with quantum models showing robustness under noise. A key finding of the paper was the reduction in model parameters: the quantum model required only 28 parameters for Frozen Lake, compared to 64 in a classical Q-learning model, suggesting improved memory efficiency.

[41] addresses the challenge of applying QRL in continuous and discrete action spaces. The proposed algorithm, DDPG, utilizes VQC with separate reward and environment registers in different circuit stages. A combination of R_x, R_y, and R_z gates is used alongside entanglement, while classical optimization techniques are employed to minimize the mean square loss using an Adam optimizer [68]. The work focuses on amplitude encoding (cf. Section 4.1), where the amplitudes of a quantum state represent the environment's state. The algorithm is tested on tasks such as quantum state generation and eigenvalue problems, in addition to the Frozen Lake environment. The findings show the potential of VQCs to handle large state spaces, making them feasible for more complex RL problems.

[69] proposes a quantum-inspired experience replay mechanism, termed DRL-quantum-inspired experience replay (DRL-QER), to enhance the training process of DRL by improving the balance between exploration and exploitation. In contrast to traditional experience replay methods, DRL-QER adaptively selects experiences from the buffer based on the complexity of the transitions and the number of times they have been replayed. This is achieved through two operations: the preparation operation, which adjusts the probability amplitude of experiences according to their TD errors, and the depreciation operation, which ensures diversity by decreasing the probability of overused experiences. These quantum-inspired methods are implemented through classical simulations and applied to the widely used deep Q-network (DQN) framework. The study applies DRL-QER to various Atari 2600 games, such as Space Invaders, Breakout, Freeway, and River Raid, which feature high-dimensional state inputs and discrete action spaces. Experimental results demonstrate that DRL-QER achieves faster convergence and improved training efficiency compared to traditional methods like prioritized experience replay (PER) and deep curriculum reinforcement learning

(DCRL). Additionally, DRL-QER is tested with variants of DQN, including double DQN and dueling DQN architectures, demonstrating its compatibility with more advanced memory-based DRL algorithms.

[70] takes a different approach, focusing on controlling real-time distributed renewable energy systems (RES) rather than game-based environments. This study introduces quantum-inspired RL techniques for MARL systems to manage distributed energy storage and generation. Nine different quantum-inspired DRL algorithms, including quantum-inspired Q-learning, policy gradient, and actor-critic methods, are tested across two 100% RES scenarios. The results show that hybrid methods outperform classical algorithms regarding frequency deviation reduction and energy consumption. Notably, these methods offer better control under uncertain conditions, such as stochastic behavior in the energy grid, suggesting their practical application in complex, real-world systems.

In this work, [71] embeds a shallow VQC-based policy within the REINFORCE algorithm, an MC-based policy gradient method. The proposed approach utilizes a hardware-efficient ansatz with alternating layers of single-qubit rotation gates and entangling CNOT gates, with angle encoding employed to map continuous state spaces into quantum states. The model is tested in standard RL environments such as CartPole-v0 and Acrobot-v1, which involve continuous state spaces, discrete action spaces, and a quantum control task. The results show that the VQC models perform similarly to classical NNs, requiring significantly fewer trainable parameters. Additionally, the quantum models demonstrate greater resistance to barren plateaus [72], enhancing their trainability. Overall, the study illustrates the potential for VQCs to reduce the complexity of RL policies while maintaining strong performance in benchmarking tasks.

[23] explores different architectural choices and data encoding strategies and performs ablation studies to assess the VQC-based Q-learning approach. The VQC is based on a hardware-efficient ansatz, consisting of parameterized y- and z-rotations and a daisy chain of CZ gates. Continuous state spaces are encoded using x-rotations, and data re-uploading is used to improve expressivity. The model is tested on Frozen Lake (discrete state space) and CartPole (continuous state space), both with discrete action spaces. The algorithm is a quantum adaptation of DQN, where Q-values are updated with an ϵ-greedy policy and experience replay. The results show that VQCs perform comparably to classical NNs while requiring fewer parameters. Moreover, deeper circuits lead to faster convergence and more accurate Q-value approximations.

The work by [73] presents a hybrid quantum-classical approach to robot navigation tasks using the `PyBullet` real-time physics simulation engine [74] and a simulated TurtleBot2[1] robot. The study focuses on discrete RL problems, employing the double deep Q-network (DDQN) algorithm to train a robot to navigate grid mazes of increasing complexity, with environments of 3×3, 4×4, and 5×5 grid dimensions, as shown in Figure 4.1. In these environments, the robot starts in the upper-left corner and can perform three discrete actions: move forward, rotate left, or rotate right. The robot's potential paths to solve each environment are marked in Figure 4.1. The state space the agent observes includes three real-valued features: its position coordinates x, y and its z-orientation.[2]

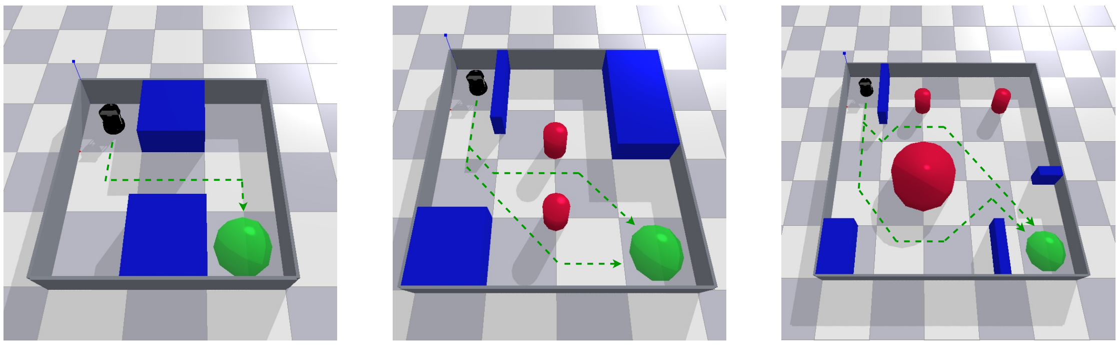
The proposed solution employs a DDQN algorithm, which reduces overestimation bias by utilizing two separate action-value functions, q and q', along with policy networks π and π'. These target networks (q' and π') are periodically updated with the parameters of the main networks, q and π, ensuring smoother and more reliable learning by preventing rapid fluctuations in value estimates. The objective of the task is to navigate the robot to the goal, located in the lower-right corner while avoiding obstacles. The reward structure penalizes collisions (-1) and unnecessary movements (-0.2), only providing incremental rewards for moving closer to the goal ($+0.1$) and a reward of $+10$ upon reaching the goal.

For the actor and critic, PQCs are used as function approximators. Figure 4.2 illustrates the general structure of the PQCs, highlighting two main components: the parameterized gates (purple) and the encoding gates (yellow). The encoding occurs in the orange section of the PQC, where data is uploaded into the circuit using the equation $x_{li} = \arctan(\lambda_{li} s_i)$, with λ_{li} as a trainable parameter, s_i representing the observed state and arctan as the activation function, as shown in Figure 4.3. The parameterized gates (purple) do not contain observation data but have trainable weights subject to classical optimization through the Adam optimizer.

The study instructs two variants of PQCs: PQC-1 and PQC-3. PQC-1 encodes one input feature per qubit using an R_x gate, whereas PQC-3 encodes all observed features on each qubit using R_x, R_y, and R_z gates. Figure 4.3 provides a detailed view of the PQC-1 and PQC-3 circuits, displaying how the encoding and parameterization are structured differently. The results indicated that PQC-3, which encodes all features on each qubit, exhibited faster convergence and better performance than PQC-1. Additionally, large

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

[2]At the time of writing, the state space has changed in their recent paper update $v3$ [73], where a dynamic environment was introduced that incorporates both local and global observations. The 3×3, 4×4, and 5×5 environments, however, were not tested with the new state space.



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

Figure 4.1.: Robotic Navigation Environments from [73]: 3×3 (a), 4×4 (b), and 5×5 (c) grid mazes with gray walls, blue and red static obstacles, and a green goal area. The robot starts from the upper-left corner in each environment. The dashed green line sketches potential paths the robot can take to solve the environment.

QCs, with up to 39 layers, were tested in more dynamic environments to assess the scalability and robustness of the quantum approach for complex robotic navigation tasks.

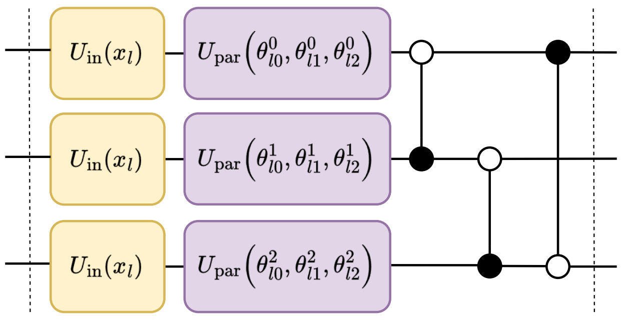

$U_{\text{in}}(x_l)$
$U_{\text{par}}\left(\theta_{l0}^0, \theta_{l1}^0, \theta_{l2}^0\right)$
$U_{\text{in}}(x_l)$
$U_{\text{par}}\left(\theta_{l0}^1, \theta_{l1}^1, \theta_{l2}^1\right)$
$U_{\text{in}}(x_l)$
$U_{\text{par}}\left(\theta_{l0}^2, \theta_{l1}^2, \theta_{l2}^2\right)$

Figure 4.2.: PQC ansatz layout from [73], showing one layer with repeated encoding of input data in the yellow blocks $U_{\text{in}}(x_l)$ and parameterized operations in the purple blocks $U_{\text{par}}(\theta)$. The structure includes an entangling layer connecting the qubits through CZ gates, as illustrated by the black dots and lines between qubits.

Most studies reviewed focus on idealized environments from the `OpenAI Gym` suite, such as Frozen Lake, CartPole, Grid World, and others. These environments typically

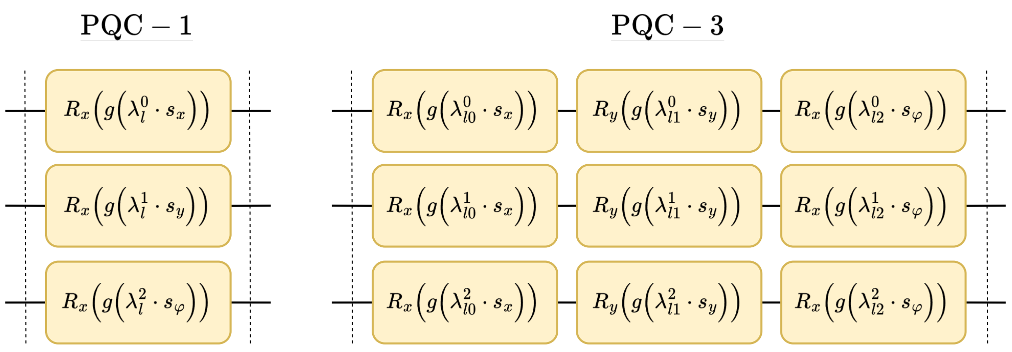

PQC − 1
PQC − 3
$R_x(g(\lambda_l^0 \cdot s_x))$
$R_x(g(\lambda_l^1 \cdot s_y))$
$R_x(g(\lambda_l^2 \cdot s_\varphi))$
$R_x(g(\lambda_{l0}^0 \cdot s_x))$
$R_y(g(\lambda_{l1}^0 \cdot s_y))$
$R_x(g(\lambda_{l2}^0 \cdot s_\varphi))$
$R_x(g(\lambda_{l0}^1 \cdot s_x))$
$R_y(g(\lambda_{l1}^1 \cdot s_y))$
$R_x(g(\lambda_{l2}^1 \cdot s_\varphi))$
$R_x(g(\lambda_{l0}^2 \cdot s_x))$
$R_y(g(\lambda_{l1}^2 \cdot s_y))$
$R_x(g(\lambda_{l2}^2 \cdot s_\varphi))$

Figure 4.3.: Input encoding $U_{\text{in}}(x_l)$ from [73] for the PQC architectures. PQC-1 (left) uses R_x gates with each qubit encoding one feature s_i, modulated by a trainable weight λ_l^i. PQC-3 (right) encodes all features s_i on each qubit using R_x and R_y gates, with separate trainable weights λ_l^i for each gate.

feature either discrete state or action spaces, or both, which limit the complexity of the tasks. While the `OpenAI Gym` suite is a widely used benchmark for RL algorithms, offering a variety of games with different combinations of continuous and discrete state-action spaces, most QRL approaches have concentrated on simplified scenarios. Few studies have explored more industrial-like environments involving continuous state and action spaces, leaving a gap in the application of QRL to more realistic tasks.

This thesis addresses the challenge of robot navigation in continuous action and state spaces, a scenario not thoroughly investigated in existing literature. While early QRL approaches have shown promise in discrete action spaces, they face difficulties in continuous domains due to dimensionality issues introduced by discretization [41]. To overcome this limitation, a quantum-enhanced DDPG algorithm is proposed to solve sequential decision problems in both classical and quantum domains.

This work extends the research of [73], which explored robot navigation in continuous state spaces, but with discrete actions. By introducing a continuous action space, the framework allows for smoother and more adaptive responses based on local observations without relying on an oracle entity. This approach integrates quantum algorithms with classical RL techniques, creating a more realistic and flexible approach. The following chapter will discuss the problem statement and the proposed solution, highlighting the application of this quantum-classical hybrid method to robot navigation tasks. This novel approach positions the work as a significant contribution to the field of QRL.

5. Problem Statement and Solution

This chapter addresses the main challenges motivating this thesis, emphasizing the constraints of existing QRL methodologies within the robotics domain. A comparative analysis is conducted between the approach of [73] and the framework proposed herein, focusing on distinctions in action space, state representation, and the complexity of the learning environment. While previous studies have predominantly employed discrete action spaces and global state representations, this thesis advances QRL to accommodate more complex scenarios characterized by continuous action spaces and localized sensor data. A solution utilizing the DDPG algorithm is proposed to overcome these challenges. Three methodological approaches are explored: a classical baseline, a hybrid model integrating classical layers with QCs, and a quantum approach wherein function approximation is handled solely by a VQC, supplemented by classical pre- and post-processing steps, including classical optimization as illustrated in Figure 3.4. Furthermore, this chapter details the software and libraries employed for implementation and assesses the scalability of quantum and hybrid quantum-classical circuits relative to task complexity and model size.

5.1. Challenges in QRL for Robot Navigation

As elaborated in Section 4.2, a substantial portion of QRL research utilizes Atari 2600 games as a testing framework [6]. Although certain Atari games incorporate continuous elements, many rely on discrete action and state spaces, limiting their applicability to more sophisticated tasks such as robot navigation. The inherent discretization in these environments does not adequately capture the continuous and dynamic interactions essential for real-world applications, rendering direct comparisons with industrial scenarios less relevant. Discretizing action spaces neglects important details about the relationships within the action domain, which are crucial for solving many real-world problems [5].

Additionally, Atari games often involve a superficial selection of features. Typically, features like environmental images or entity coordinates are directly input into NNs, resulting in a global state representation. This contrasts with the requirements of robot navigation tasks, where agents depend on localized observations derived from

sensor data. Global features, such as externally provided x, y coordinates, fail to reflect the autonomous observation generation necessary for navigation in unfamiliar environments. For instance, a robot navigating without external positioning systems must rely on local sensor inputs, such as LiDAR readings, to determine its path.
An illustrative example of this limitation is presented in [73], where agents in a robot navigation scenario receive global features like x, y coordinates within static environments. Although recent iterations of their work incorporate sensor data to represent local features in dynamic settings, global features such as the Euclidean distance to the goal and the angular difference between the robot's current heading and the direction toward the goal remain, resulting in a mix of local and global data.
Moreover, existing literature, including the work of [73], frequently emphasizes discretized state and action spaces, constraining the agent's capacity for nuanced control in tasks like navigation and steering. As such discretization simplifies the learning problem, it reduces the relevance of these approaches for practical, real-world applications that demand continuous control. Furthermore, although [73] provides comparisons between classical and various quantum models, there is a notable scarcity of research investigating the integration of classical NN layers with quantum function approximation - a hybrid approach. This methodology, wherein classical layers interface with QCs, has the potential to yield deeper insights into the distinct contributions of classical and quantum components within RL frameworks.

This thesis aims to address these limitations by implementing a DDPG approach with diverse ansatzes and varying circuit sizes, thereby significantly enhancing the complexity compared to the methodologies employed in [73]. Additionally, the task's complexity is increased by relying only on local features, continuous action and state spaces, and a comparative analysis of classical, hybrid, and quantum approaches across different circuit architectures and ansatzes. This holistic approach contributes to the advancement of QRL in robotic navigation by providing a more realistic and scalable testing environment.

5.2. Proposed Methodology

This methodology addresses the challenges in QRL for robot navigation by integrating advanced simulation environments, dynamic reward structures, and various computational models. It starts with designing a sophisticated RL environment that employs continuous action spaces and locally derived sensor data, such as LiDAR readings, to better emulate real-world robotic interactions. A refined reward system is implemented to encourage consistent progress while penalizing inefficiencies and promoting robust learning behaviors. The approach investigates three types of models for function approximation: classical NNs, hybrid quantum-classical architectures, and predominantly quantum VQCs. Each model is evaluated across different QC configurations and ansatzes to assess their performance and scalability. Further, hardware and software characteristics are considered to support extensive simulations and model training. By systematically comparing classical, hybrid, and quantum approaches, the methodology aims to disclose the strengths and limitations of integrating quantum computing into RL frameworks, ultimately advancing the field of quantum-enhanced robotics.

5.2.1. Design of RL Environment

This thesis extends the work of [73], as discussed in Section 4.2, by utilizing three static environments as test suites (cf. Figure 4.1) and a simulated TurtleBot2 robot implemented with the `PyBullet` real-time physics simulation engine [74], taken and modified from the GitHub repository[1] of [73]. The simulation operates at a frequency of 100 Hz for collision detection and motion dynamics. The TurtleBot2 features a differential drive system consisting of two wheels, each controlled by individual motors. Movement is achieved by setting velocities for each wheel: applying equal positive or negative velocities results in strict forward or backward motion, while equal but opposite velocities induce rotation in place. Differing wheel velocities produce curved trajectories.

Contrary to [73], which utilizes a discretized action space comprising three distinct actions: forward movement and left and right rotations by $40° – 50°$ (due to motion carryover between simulation steps), the current implementation adopts a continuous action space. This is achieved by allowing wheel velocities to vary continuously within the range of 0 to 10. Additionally, the state space has been significantly enhanced from the basic global coordinates (x and y) and local z-orientation to more complex, industrial-grade sensory inputs, organized in Figure 5.1.[2] These include LiDAR data, linear and angular velocities, and z-orientation. For each environment, the robot initiates its navigation from the upper left position and aims to reach the goal threshold located at the bottom right. This enhancement ensures that the state representation relies solely on local observations, better mirroring real-world robotic applications.

[1] https://github.com/dfki-ric-quantum/qdrl-turtlebot-env

[2] At the time of writing, [73] released paper version $v3$, which introduces a dynamic environment incorporating LiDAR data as state inputs, along with the Euclidean distance to the goal and the angular difference between the robot's current heading and the direction toward the goal (both global information). However, the static environments 3×3, 4×4, and 5×5 remained untested with the updated state space.

Consistent with the previous simulation steps, applying the maximum velocity of 10 to both wheels results in an approximate traversal distance of ~ 0.18 units per action, although a deterministic statement of distance cannot be made due to motion carryover affecting subsequent actions. Each action is executed over a duration of 0.5 seconds and is selected every 50 simulation steps, analogous to [73].

5.2.2. Computation of Reward Thresholds

The reward function employed in the RL setup is defined in Equation 5.1. Unlike the original implementation presented in [73], which utilizes a static reward of 0.1, the current approach adopts a dynamic reward structure based on the agent's progress toward the goal. Specifically, the reward is determined by the change in Euclidean distance to the goal between consecutive states, with the progress threshold lowered to 0.05 units from the original 0.1.

$$r(s_t, s_{t+1}) = \begin{cases} 10.0 & \text{if } \|s_{t+1} - \text{goal}\| \leq \text{goal threshold} \\ \|s_t - \text{goal}\| - \|s_{t+1} - \text{goal}\| & \text{if } \|s_t - \text{goal}\| - \|s_{t+1} - \text{goal}\| \geq 0.05 \\ -1.0 & \text{in case of collision} \\ -0.2 & \text{otherwise} \end{cases} \tag{5.1}$$

This modification ensures that the reward function dynamically reflects the agent's advancement toward the goal, providing more detailed feedback and encouraging consistent progress. The other components of the reward function, including collision penalties and default negative rewards, remain consistent with the original framework established in [73], thereby promoting trajectories that lead toward the goal.

Successful navigation within an environment is determined by meeting one of two predefined thresholds, t_1 and t_2, each representing different levels of effectiveness. The primary threshold, t_1, is manually determined, analogous to the method used by Hohenfeld [73]. In their work, the reward thresholds were set to 10.5, 11.0, and 10.0 for the 3×3, 4×4, and 5×5 environments, respectively. These values represent lower bounds derived from a range of successful navigation trajectories. Additionally, the authors established near-optimal step counts of 20, 30, and 45 steps for the respective environments. In the original implementation, the robot operated with a discrete action space comprising three actions: moving forward, left rotation, and right rotation. This required the robot to execute rotational actions that changed its orientation without altering its location, effectively consuming steps solely for rotation. To account for the increased mobility in the current implementation—which introduces a continuous action space allowing curved trajectories—the near-optimal step counts are adjusted by subtracting the steps previously needed for rotations. These adjusted step counts represent the near-optimal benchmarks for the continuous action space in this thesis and are used to determine the steps required to reach the goal while achieving the reward threshold t_1. The t_1 reward thresholds for the continuous action space were established accordingly through manual testing. The adjusted step counts and corresponding t_1 reward thresholds are as follows:

$$\begin{aligned}
3{\times}3: &\quad t_1\text{-steps: } 20-3=17, \quad t_1: \quad 12.0,\\
4{\times}4: &\quad t_1\text{-steps: } 30-2=28, \quad t_1: \quad 13.5,\\
5{\times}5: &\quad t_1\text{-steps: } 45-5=40, \quad t_1: \quad 14.5.
\end{aligned}$$

To further accommodate variations in agent performance, a secondary threshold t_2 is introduced. This threshold provides a more tolerant criterion for successful navigation by allowing an additional 50% of the steps required to achieve t_1. The t_2 threshold is calculated by adding half of the near-optimal step count to the t_1 threshold, with a penalty of -0.2 applied for each additional step. For example, the t_2 threshold is calculated as follows:

$$\begin{aligned}
3{\times}3: &\quad t_2\text{-steps: } 17\times1.5=25.5, \quad t_2: \quad 12.0-(8.5\times0.2)=12.0-1.7=10.3,\\
4{\times}4: &\quad t_2\text{-steps: } 28\times1.5=42, \quad t_2: \quad 13.5-(14\times0.2)=13.5-2.8=10.7,\\
5{\times}5: &\quad t_2\text{-steps: } 40\times1.5=60, \quad t_2: \quad 14.5-(20\times0.2)=14.5-4.0=10.5.
\end{aligned}$$

Under the t_2 threshold, the agent is considered to have successfully navigated the environment even if it does not consistently decrease the distance to the goal by at least 0.05 units per step. Achieving t_2 indicates that the agent has reached the goal, albeit with reduced efficiency compared to meeting the t_1 threshold. All derived reward and step thresholds are listed for clarity in Table A.1. Furthermore, each episode is limited to a maximum of 100 steps. If the agent fails to reach the goal within this limit, the environment is reset, and a collision penalty is applied. This constraint prevents episodes from running indefinitely and encourages the agent to identify effective paths within a reasonable number of actions.

Overall, the implementation of dual thresholds t_1 and t_2, along with step limitations, provides a balanced evaluation framework. It rewards efficient progress while penalizing prolonged or ineffective actions, encouraging the development of robust and efficient navigation strategies across various environmental complexities.

5.2.3. RL Models: Classical, Hybrid, and Quantum

This study explores three RL solutions: quantum, hybrid, and classical approaches. While still hybrid in implementation due to the classical optimizer, the quantum model operates without classical NN layers, relying solely on a VQC for function approximation. The hybrid approach combines classical and quantum function approximation by incorporating a classical layer preceding the VQC, allowing the model to leverage both types of computations. Finally, a purely classical NN serves as a baseline for comparison, providing a conventional architecture against which the performance of quantum-based models can be evaluated. All three approaches are built on the DDPG algorithm, detailed in Section 2.3.2, with the corresponding pseudo-code provided in Algorithm 1. The specific hyper-parameters used across the environments are presented in Table A.1, with a concise explanation provided in Appendix A.1. The overall structure of the QRL flow is illustrated by Figure 3.4.

Figure 5.1 presents the features (observation attributes) used across these three approaches, outlining the input structure that feeds into each model and highlighting the information available to the actor and critic networks. For the actor model, six features, f_1 through f_6, are employed, while the critic model extends this set by incorporating two additional action-related features, f_7 and f_8. The first three features, f_1, f_2, and f_3, are derived from LiDAR sensors that detect objects within a distance of 2 units. These sensors are positioned at fixed angles, separated by 45° $\left(\frac{\pi}{4}\text{ radians}\right)$, and aligned to face the robot's direction of movement. If no object is detected, the full sensor range is used. Feature f_4 represents the robot's orientation, denoted as its z-orientation, while features f_5 and f_6 capture the motion dynamics based on the TurtleBot's differential drive system. Specifically, f_5 is the linear velocity, and f_6 is the angular velocity of the robot. In addition to these observation features used by the actor, the critic model includes the left and right wheel velocities, represented by features f_7 and f_8, respectively. This addition aligns with the setup of the DDPG algorithm, where the critic evaluates the actions chosen by the actor (policy). Thus, the critic requires access to both the observation features and the actions to assess the effectiveness of the chosen actions in the given state. Finally, all features undergo a pre-processing step, normalizing them to specific ranges as indicated in the Pre-Processing box, ensuring consistency across inputs.

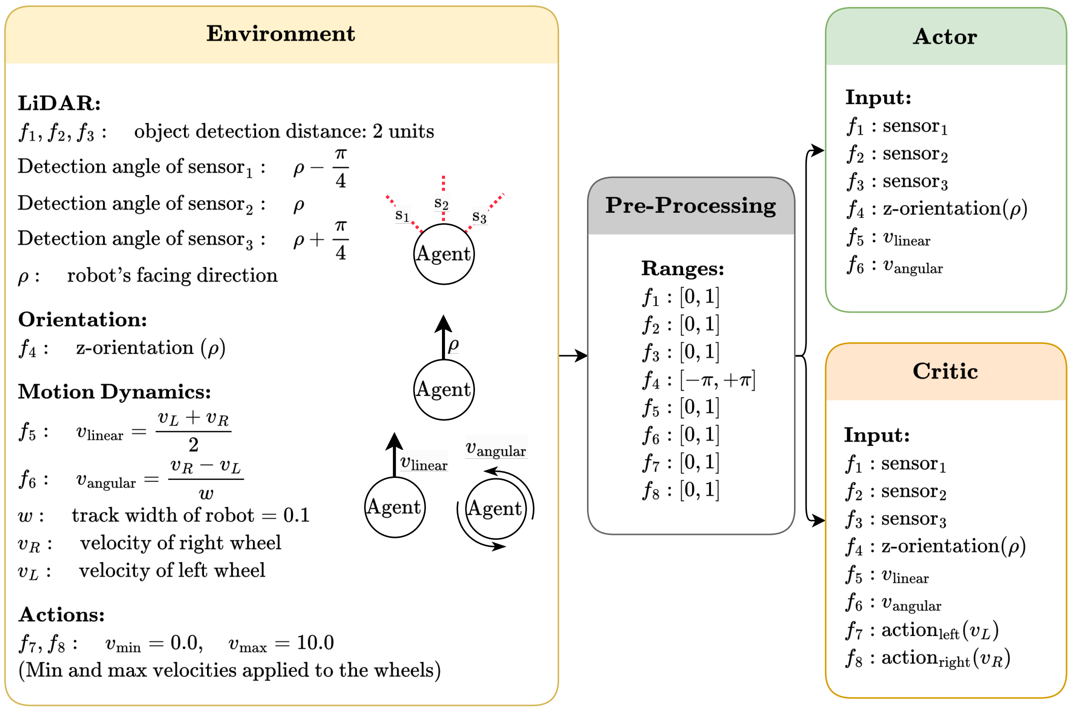

Environment
LiDAR:
f_1, f_2, f_3 : object detection distance: 2 units
Detection angle of sensor$_1$: $\rho - \frac{\pi}{4}$
Detection angle of sensor$_2$: ρ
Detection angle of sensor$_3$: $\rho + \frac{\pi}{4}$
ρ : robot's facing direction
s_1
s_2
s_3
Agent
Orientation:
f_4 : z-orientation (ρ)
ρ
Agent
Motion Dynamics:
f_5 : $v_{\text{linear}} = \frac{v_L + v_R}{2}$
f_6 : $v_{\text{angular}} = \frac{v_R - v_L}{w}$
w : track width of robot $= 0.1$
v_R : velocity of right wheel
v_L : velocity of left wheel
v_{linear}
Agent
v_{angular}
Agent
Actions:
f_7, f_8 : $v_{\min} = 0.0$, $v_{\max} = 10.0$
(Min and max velocities applied to the wheels)
Pre-Processing
Ranges:
$f_1 : [0, 1]$
$f_2 : [0, 1]$
$f_3 : [0, 1]$
$f_4 : [-\pi, +\pi]$
$f_5 : [0, 1]$
$f_6 : [0, 1]$
$f_7 : [0, 1]$
$f_8 : [0, 1]$
Actor
Input:
f_1 : sensor$_1$
f_2 : sensor$_2$
f_3 : sensor$_3$
f_4 : z-orientation(ρ)
f_5 : v_{linear}
f_6 : v_{angular}
Critic
Input:
f_1 : sensor$_1$
f_2 : sensor$_2$
f_3 : sensor$_3$
f_4 : z-orientation(ρ)
f_5 : v_{linear}
f_6 : v_{angular}
f_7 : action$_{\text{left}}(v_L)$
f_8 : action$_{\text{right}}(v_R)$

Figure 5.1.: Representation of the input features used to train the actor and critic networks. The input for the actor network consists of the object detection distances from the LiDAR sensors (f_1 to f_3), where the full detection range is used if no object is found within the maximum distance, the robot's z-orientation (f_4), and its linear (f_5) and angular velocities (f_6). For the critic network, the additional left (f_7) and right (f_8) wheel velocities are included as actions. All features are pre-processed before being input into the networks, with the corresponding feature ranges shown.

The classical NN architecture used in this study, based on the implementation of `CleanRL ddpg_continuous_action.py` [75] and modified to meet the requirements of this specific RL problem, consists of a critic network and an actor network. The critic receives an input that combines the observation and action dimensions, resulting in an input size of eight, with six observation features and two action features. Its architecture includes three fully connected (linear) layers: the first layer maps the input to an intermediate representation, followed by a second layer, and finally, an output layer with a single unit representing the Q-value. The activation function, rectified linear unit (ReLU), defined as $\text{ReLU}(x) = \max(0, x)$, is applied after the first two layers to introduce non-linearity, while the final layer uses a linear activation to produce the Q-value. Additionally, the action input is normalized by dividing by 10.0 before concatenation with the observation input.

The actor network, with an input dimension of six observation features, follows a similar structure with three fully connected layers. The first two layers apply ReLU activations, while the final layer uses *tanh* to constrain the action output within $[-1, 1]$. The output, representing the action space's two dimensions (left and right wheel velocities), is rescaled to a positive range by applying an environment-specific bias and scaling factor. This range is then clipped to align with the maximum velocity limit of 10, as depicted in the figure, ensuring the actions conform to the continuous action space constraints. In the scope of this thesis, the classical models are annotated with c, with the number of neurons in the first and second hidden layers specified as subscripts.

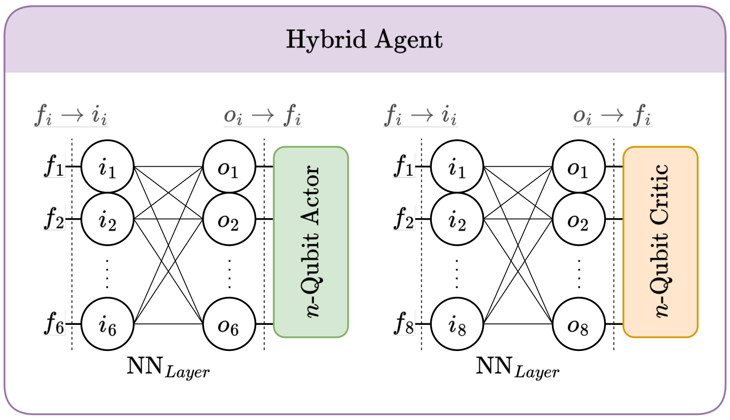


Figure 5.2.: Hybrid agent illustrating the processing of features f_1 to f_6 for the actor and f_1 to f_8 for the critic (cf. Figure 5.1). The features are passed through a fully connected NN$_{\text{Layer}}$, where the input and output correspond to the number of features. The NN's output is then forwarded into the respective quantum circuit q_{Circuit}. The quantum ansatz used in the circuit is the same as that presented in Figure 5.3. The actor model utilizes 6 features, while the critic model processes 8 features.

To enable VQC-based function approximation, the classical `CleanRL` implementation was customized and modified accordingly, merging the work from [73] with [75]. In this setup, both the hybrid and quantum approaches incorporate a VQC but differ in how they process input features and their degree of hybridization. The hybrid approach includes a classical layer before the QC. Specifically, the input features first pass through a simple classical NN with fully connected input and output layers. The number of neurons in the input and output layers matches the observation dimensions: six for the actor and eight for the critic. This configuration essentially maps the input features to an identical output size, with the processed output serving as the input to the VQC. This process is illustrated in Figure 5.2. In both hybrid and quantum approaches, the VQC structure depends on the number of qubits and layers used. A schematic representation of the VQC is shown in Figure 5.3. The VQC design is inspired by the architectures presented in [73] and [23], but employs a different encoding scheme to better accommodate the specific requirements of this study. The n-qubit actor and critic models begin with an initial layer of parameterized rotation gates, R_x, R_y, and R_z, without feature encoding (white). This is followed by an entanglement chain composed of CZ gates, where each upper qubit (empty circle) serves as the control, and the subsequent qubit (filled-in circle) acts as the target. The last qubit controls the first, ensuring full entanglement across all qubits. Each repeating layer comprises feature encoding gates, variational gates, and entanglement. The feature encoding gates (orange) involve alternating between R_x and R_y gates, iterating over the six features for the actor and eight for the critic. For each encoding gate, classical features are multiplied by corresponding trainable parameters, λ, and circulated horizontally according to the modulo operation, mod 6 for the actor and mod 8 for the critic, with the latter including action features. After the feature encoding gates, a variational set of gates identical to the initial layer is applied, concluded by entanglement using CZ gates. This sequence forms one complete layer of the circuit, which is repeated according to the specified number of layers. All parameters applied in the circuit are individual to the respective gates and are not repeated or reused. Lastly, a measurement is performed on the first qubit for the actor and on the first and second qubits for the critic. Throughout this work, quantum and hybrid models are annotated by q and h, respectively, with the number of qubits and layers written as subscripts.

The ansatz illustrated in Figure 5.3 distinguishes between the actor and critic networks due to their differing numbers of encoded features. With an increase in layers and, consequently, circuit size, the total weights of each network also vary. For the actor network, the weights are composed as follows:

The initial layer includes 3 trainable weights per qubit for initialization. Each subsequent layer contributes $3 \times$ #qubits weights from the variational gates and #features $\times$ #qubits $\times$ #layers weights from the encoding process. In the actor network, where #features $= 6$, output scaling is applied for both the left and right actions, in line with the approach outlined in [23] and [73]. This results in the total number of weights being

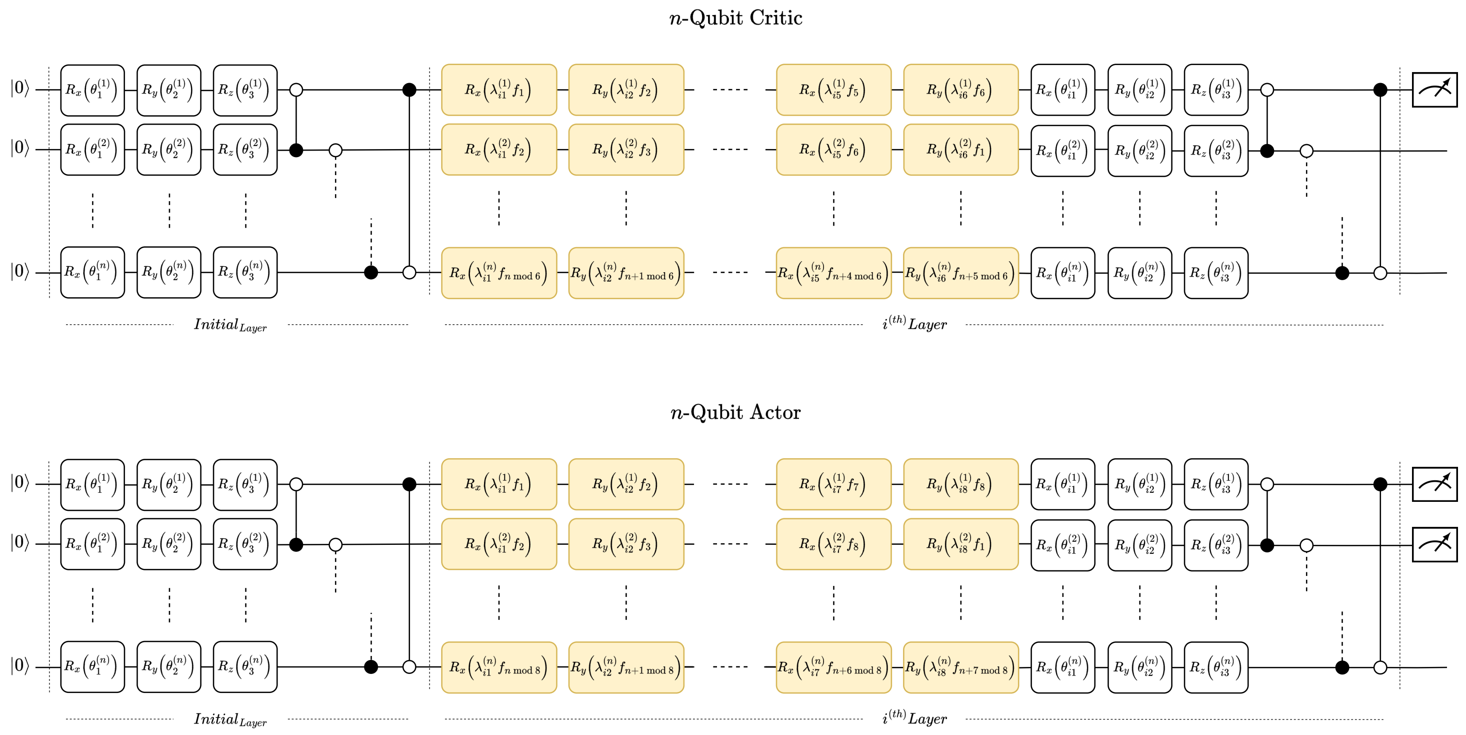

n-Qubit Critic
|0⟩
$R_x(\theta_1^{(1)})$
$R_y(\theta_2^{(1)})$
$R_z(\theta_3^{(1)})$
$R_x(\lambda_{i1}^{(1)} f_1)$
$R_y(\lambda_{i2}^{(1)} f_2)$
$R_x(\lambda_{i5}^{(1)} f_5)$
$R_y(\lambda_{i6}^{(1)} f_6)$
$R_x(\lambda_{i1}^{(n)} f_{n \bmod 6})$
$R_y(\lambda_{i2}^{(n)} f_{n+1 \bmod 6})$
$R_x(\lambda_{i5}^{(n)} f_{n+4 \bmod 6})$
$R_y(\lambda_{i6}^{(n)} f_{n+5 \bmod 6})$
$Initial_{Layer}$
$i^{(th)} Layer$
n-Qubit Actor
$R_x(\lambda_{i7}^{(1)} f_7)$
$R_y(\lambda_{i8}^{(1)} f_8)$
$R_x(\lambda_{i1}^{(n)} f_{n \bmod 8})$
$R_y(\lambda_{i2}^{(n)} f_{n+1 \bmod 8})$
$R_x(\lambda_{i7}^{(n)} f_{n+6 \bmod 8})$
$R_y(\lambda_{i8}^{(n)} f_{n+7 \bmod 8})$

Figure 5.3.: A general n-qubit actor and critic model used for function approximation. The gates (R_x, R_y, R_z) represent variational gates (white) containing trainable parameters (θ) without data encoding. The orange gates represent the data encoding of the classical features (cf. Figure 5.1). In both the actor and critic circuits, the classical features are multiplied by corresponding trainable parameters (λ). The features are circulated as indicated by the modulo operation, mod 6 for the actor and mod 8 for the critic, with the critic containing two additional features (cf. Figure 5.1). The encoding gates alternate between R_x and R_y gates, incrementing the features. The qubits are entangled via CZ gates, where the upper qubit is the control (empty circle), and the subsequent qubit is the target (filled-in). The last qubit is the control qubit for the first (target) qubit, ensuring full entanglement across the qubits. Only the first qubit is measured for the critic, while the first two qubits are measured for the actor.

calculated as follows:

$$\theta_{\text{Actor}} = 3 \times \#\text{qubits} + 3 \times \#\text{layers} \times \#\text{qubits} + 6 \times \#\text{layers} \times \#\text{qubits} + 2.$$

For the critic network, where #features $= 8$, the structure is similar, with an additional single output scaling parameter for the Q-values. Therefore, the total weight count for the critic network is:

$$\theta_{\text{Critic}} = 3 \times \#\text{qubits} + 3 \times \#\text{layers} \times \#\text{qubits} + 8 \times \#\text{layers} \times \#\text{qubits} + 1.$$

The hybrid approach introduces a classical layer alongside the quantum components, contributing additional weights to the actor and critic models. For the actor, the classical layer has 6 inputs and 6 outputs, resulting in 36 weights plus 6 biases, totaling 42 parameters. The remaining parameters, including the QC and output scaling, are identical to those in the quantum ansatz calculation. This gives the total weight count for the actor as:

$$\theta_{\text{Actor}} = 42 + 3 \times \#\text{qubits} + 3 \times \#\text{layers} \times \#\text{qubits} + \#\text{features} \times \#\text{layers} \times \#\text{qubits} + 2.$$

For the critic network, the classical layer has 8 inputs and 8 outputs, resulting in 64 weights plus 8 biases, totaling 72 parameters. The QC and output scaling structure remain the same as in the quantum ansatz, giving the total weight count for the critic network as:

$$\theta_{\text{Critic}} = 72 + 3 \times \#\text{qubits} + 3 \times \#\text{layers} \times \#\text{qubits} + \#\text{features} \times \#\text{layers} \times \#\text{qubits} + 1.$$

Table 5.1 summarizes the trainable parameters for all models used in this thesis. NNs with a similar number of parameters as the quantum models were chosen as a classical baseline for comparison. Additionally, an advanced classical model with hidden layers of size $(256, 256)$ was included to provide a performance benchmark, allowing for a comparison of parameter complexity and effectiveness relative to the quantum and hybrid approaches.

5.2.4. Hardware, Software, and Computational Resources

The workload for this project was distributed over a cluster of machines, most of which were equipped with an AMD Ryzen 9 5950X processor featuring 16 cores and 32 threads, paired with an NVIDIA GeForce RTX 3090 GPU. Some machines were equipped with less performant GPUs, such as the NVIDIA GeForce GTX 1080. The simulated static environment used to train and evaluate the robot was constructed using `PyBullet` [74], a real-time physics simulation engine. The environment was

Table 5.1.: Distribution of trainable parameters (θ) for quantum (q), hybrid (h), and classical (c) models across actor and critic networks. For quantum and hybrid models, θ_Q includes variational and data encoding parameters. Output scaling parameters are included as classical trainable weights, with two for the actor and one for the critic. The $c_{256,256}$ model is included for baseline comparison. The table presents θ_Q, θ_C, and θ_{Total} for both actor and critic networks, along with their total counts.

	Actor			Critic			
Model	θ_Q	θ_C	θ_{Actor}	θ_Q	θ_C	θ_{Critic}	θ_{Total}
$q_{4,3}$	120	2	122	144	1	145	267
$q_{4,5}$	192	2	194	232	1	233	427
$q_{8,3}$	240	2	242	288	1	289	531
$q_{8,5}$	384	2	386	464	1	465	851
$h_{4,3}$	120	44	164	144	73	217	381
$h_{4,5}$	192	44	236	232	73	305	541
$h_{8,3}$	240	44	284	288	73	361	645
$h_{8,5}$	384	44	428	464	73	537	965
$c_{7,7}$	0	121	121	0	127	127	248
$c_{10,10}$	0	202	202	0	211	211	413
$c_{12,12}$	0	266	266	0	277	277	543
$c_{16,16}$	0	418	418	0	433	433	851
$c_{256,256}$	0	68,098	68,098	0	68,353	68,353	136,451

developed following the `OpenAI Gym` framework [67], which provides standardized interfaces for RL environments. QCs were simulated using `PennyLane` [76], and all circuit simulations were noise-free. The boilerplate code for the DDPG algorithm was taken from the `CleanRL` library [75]. This code was modified to incorporate QCs instead of classical NNs and hybrid approaches combining classical and quantum models. `CleanRL` utilizes the `PyTorch` library [77] for all ML tasks, including model optimization using the Adam Optimizer. The replay buffer employed in this project was sourced from the `Stable Baselines3` library [78]. The GitHub repository for the environment and simulation code was obtained and modified from the original project repository by [73], as noted in their publication.

The wall-clock times of the experiments averaged over all configurations and environment sizes, are presented in Table 5.2. The results are differentiated between the hybrid and quantum ansatz, with time reported in thousands of steps per day. While data was collected across all environment sizes (3×3, 4×4, 5×5), the time differences between environments were marginal. The machine cluster used for these experiments was shared for multiple purposes, leading to varying loads and availability throughout the process. This factor likely explains the counterintuitive observation that the hybrid method, which includes an additional classical layer prior to the QC (cf. Figure 5.2), exhibited an increase in steps per day compared to the pure quantum approach. The variability in cluster usage should be taken into account, and these results should, therefore, be interpreted as rough estimations rather than precise benchmarks.

Table 5.2.: Training steps in 24h (in thousands) for different configurations of quantum (q) and hybrid (h) models based on wall-clock time.

	$q_{4,3}$	$h_{4,3}$	$q_{4,5}$	$h_{4,5}$	$q_{8,3}$	$h_{8,3}$	$q_{8,5}$	$h_{8,5}$
Steps/24h (x 1000)	20.6	23.3	13.5	14.8	15.8	13.6	9.4	7.7

In summary, the proposed methodology effectively addresses the identified challenges by implementing the DDPG algorithm with diverse ansatzes and varying circuit sizes. This approach incorporates continuous action spaces and local feature extraction, increasing the complexity of the RL setup to better emulate real-world robotic navigation scenarios. By exploring classical, hybrid, and quantum approaches, the methodology establishes a comprehensive framework for evaluating the performance and scalability of quantum-enhanced RL models. The integration of different weights, ansatzes, and circuit architectures facilitates a robust analysis of each model's contributions and limitations. Lastly, the consideration of hardware and software characteristics ensures the feasibility of the methods.
This structured approach contributes to the advancement of QRL in the context of robotic navigation by providing a more realistic and scalable testing environment. The subsequent section presents the results of benchmarking hybrid and quantum models with 4 and 8 qubits and 3 and 5 layers, respectively, against a classical baseline with a similar number of weights. These comparisons aim to elucidate the effectiveness of quantum enhancements in RL and their potential advantages over traditional classical models.

6. Results

This chapter presents the outcomes of a comprehensive series of experiments designed to evaluate and compare the performance of VQC-based quantum and hybrid models, as well as classical NN models, within different environmental configurations. The results are structured to provide insights into both training and testing performances, encouraging a complete comparative analysis against the findings of [73]. The subsequent sections detail the experimental setup, training performance, testing performance, and overall comparative analysis across various environments and models. The chapter concludes with trends and interpretations of the different ansatzes.

6.1. Experimental Setup and Remarks

For each configuration and model detailed in the preceding chapter, three distinct random seeds were employed across the 3×3, 4×4, and 5×5 environments. The quantum and hybrid models featured configurations with either 4 or 8 qubits and were implemented using 3 or 5 layers. Training each quantum or hybrid model with three seeds per environment resulted in a total of $4 \times 3 \times 3 \times 2 = 72$ quantum-based runs. Additionally, five classical NNs were trained as baseline models, each utilizing three random seeds across all environments, contributing another $5 \times 3 \times 3 = 45$ runs. Furthermore, eleven additional classical NNs ($c_{4,4}$, $c_{8,4}$, $c_{8,8}$, $c_{16,8}$, $c_{32,16}$, $c_{32,32}$, $c_{64,32}$, $c_{64,64}$, $c_{128,64}$, $c_{128,128}$, and $c_{256,128}$) were trained on each environment with three distinct seeds. However, these were excluded from the thesis to maintain focus and prevent excessive runs, as the included runs sufficiently facilitate comparison without overfilling the thesis. Consequently, out of a possible $16 \times 3 \times 3 = 144$ classical runs, only 45 runs were included in the performance comparisons, resulting in a cumulative total of 117 experimental runs.

In addition to training performance, 117 models - including all quantum and hybrid models as well as five classical NNs ($c_{7,7}$, $c_{10,10}$, $c_{12,12}$, $c_{16,16}$, and $c_{256,256}$) - were subjected to 10 evaluation runs each, resulting in a total of 1,170 test runs to assess the effectiveness of the trained policies. These five classical NNs were selected based on their weight configurations: the first four models ($c_{7,7}$, $c_{10,10}$, $c_{12,12}$, and $c_{16,16}$) possess weight configurations comparable to the VQC used in the quantum and hybrid ansatz (cf. Figure 5.1), ensuring a fair comparison. The fifth model ($c_{256,256}$) was included to provide a benchmark through an exhaustive classical NN. The hyper-parameter settings for each trained model are detailed in Table A.1 and further elaborated in Sections A.1 and 2.3.2. A comparative performance analysis of quantum (q), hybrid (h), and classical (c) models, including the classical baseline model ($c_{256,256}$), is presented in Tables 6.1, 6.2, and 6.3, corresponding to each environment.

Model selection for evaluation was based on the exponential moving average (EMA) peak with a decay factor of 0.99. The model checkpoint closest to this EMA peak at each timestep was chosen for the evaluation runs. Each performance table includes the model identifier, total number of weights (θ), the number of successful runs out of 30 total runs (SR/30), and statistical measures for both rewards and steps. Specifically, the tables report the minimum, maximum, median, mean, and standard deviation (Std) for rewards and steps, where step counts are calculated only from successful runs. Models that did not achieve successful runs are denoted with *nan* for step-related metrics. Best-performing metrics within each model category are highlighted in bold to facilitate comparison. Also, rows corresponding to top-performing models - determined primarily by the number of successful runs and reward metrics - are shaded in green, while models achieving fewer than five successful runs are highlighted in red and excluded from metric highlighting. It should be noted that the designation of top-performing models is subject to interpretation based on these criteria. This structured presentation facilitates a clear and comprehensive performance comparison across different model types and environments.

To ensure clarity and uniformity in the presentation of results, all training plots adhere to a consistent layout and design. Figures ranging from 6.1 to 6.10 illustrate the training performance of the respective models. The legends differentiate between quantum (q), hybrid (h), and classical (c) models, specifying the number of qubits and layers for quantum and hybrid configurations and the number of neurons in the hidden layers for classical models. Each model's performance is averaged over three seeds, depicted by a solid line in the model-specific color, where a simple moving average (SMA) with a window size of 25 is applied adaptively to ensure accurate representation throughout the training process, preventing skewed averages in the early stages when data points are sparse. The corresponding standard deviation is denoted as a shaded area. Additionally, the thresholds t_1 and t_2, as defined in Section 5.2.2, are indicated by dashed horizontal lines to denote success criteria specific to each environment. The t_1 threshold also limits the upper standard deviation, preventing the visualization of rewards that exceed plausible maximum values. Listed in Table A.1, the replay memory buffer is initialized with 5000 steps. This allows the DDPG algorithm to sample and learn from an initial set of experiences, marked by a vertically dashed line in all plots as "Learning starts". Across all figures, the x-axis represents the time steps executed during training, totaling 120,000 steps per model, while the y-axis reflects the rewards achieved in one episode by the agent. The uniform styling is maintained throughout, except for the grid plot (Figure 6.10), where some labels are omitted to enhance readability; however, the notation remains consistent and clear.

6.2. Training Results

This section provides an evaluation of the training performances for quantum, hybrid, and classical models across the 3×3, 4×4, and 5×5 environments. The thresholds t_1 (near-optimal) and t_2 (sufficient) success criteria, previously defined in Section 5.2.2, are used as reference points to compare and assess model performance, providing insight into the effectiveness and learning ability of each approach. The grid plot shown in Figure 6.10 provides an overview of the training performances across all configurations, organized by similar model weights. While a more coherent comparison of these models is discussed later alongside the overall results, this plot provides a summary of the training performances covered in this section. Each plot in the grid is also presented and analyzed individually within the quantum, hybrid, and classical categories. Thus, these plots do not introduce new insights into the training performance.

6.2.1. Classical Performance

Figures 6.1, 6.2, and 6.3 display the training performance of the classical models $c_{7,7}$, $c_{10,10}$, $c_{12,12}$, $c_{16,16}$, and $c_{256,256}$ in the 3×3, 4×4, and 5×5 environments, respectively.

In the 3×3 environment, all models exhibit a rapid initial increase in average reward shortly after the commencement of training, followed by a subsequent decline. The $c_{256,256}$ model demonstrates the highest stability and consistently achieves both the t_1 and t_2 thresholds. Conversely, $c_{12,12}$ reaches the t_2 threshold and nearly attains t_1, maintaining near-threshold performance despite having fewer neurons than $c_{16,16}$. The $c_{10,10}$ and $c_{16,16}$ models show higher variation in mean rewards in the 3×3 environment, getting close to the t_2 threshold but ultimately failing to reach it. The $c_{7,7}$ model exhibits instability and performs the worst, failing to reach the t_2 threshold and concluding with a below-zero mean reward. In the 4×4 environment, all classical models, except for $c_{7,7}$, successfully achieve the t_2 threshold and demonstrate learning capability. The $c_{256,256}$ model consistently reaches the t_1 threshold, while $c_{10,10}$ and $c_{12,12}$ come close. Among the classical models, $c_{256,256}$ demonstrates the greatest stability, whereas $c_{12,12}$ and $c_{16,16}$ are sufficiently stable, with $c_{7,7}$ being the least effective. For the 5×5 environment,

$c_{256,256}$ achieves the t_1 threshold but demonstrates limited stability. Both $c_{12,12}$ and $c_{16,16}$ exhibit similar volatility in terms of robustness and reward attainment, although only $c_{16,16}$ reaches the t_2 threshold. Models $c_{7,7}$ and $c_{10,10}$ fluctuate around zero rewards, indicating inadequate learning.

Overall, all classical models encounter challenges in solving the 3×3 and 5×5 environments, with increased neuron counts generally correlating with improved performance. Models with more neurons ($c_{12,12}$, $c_{16,16}$, $c_{256,256}$) achieve better and, most of the time, more stable results, whereas $c_{7,7}$ consistently underperforms across all environments. Despite not reaching the t_2 threshold in the 3×3 environment, $c_{10,10}$ demonstrates a slightly more robust performance than $c_{7,7}$, though with a lower average reward than $c_{16,16}$, especially in the 5×5 environment, positioning it as a sufficient performer in the 3×3 and 4×4 environments. Furthermore, excluding the $c_{256,256}$ model, the $c_{12,12}$ model stands out as the best-performing classical model overall.

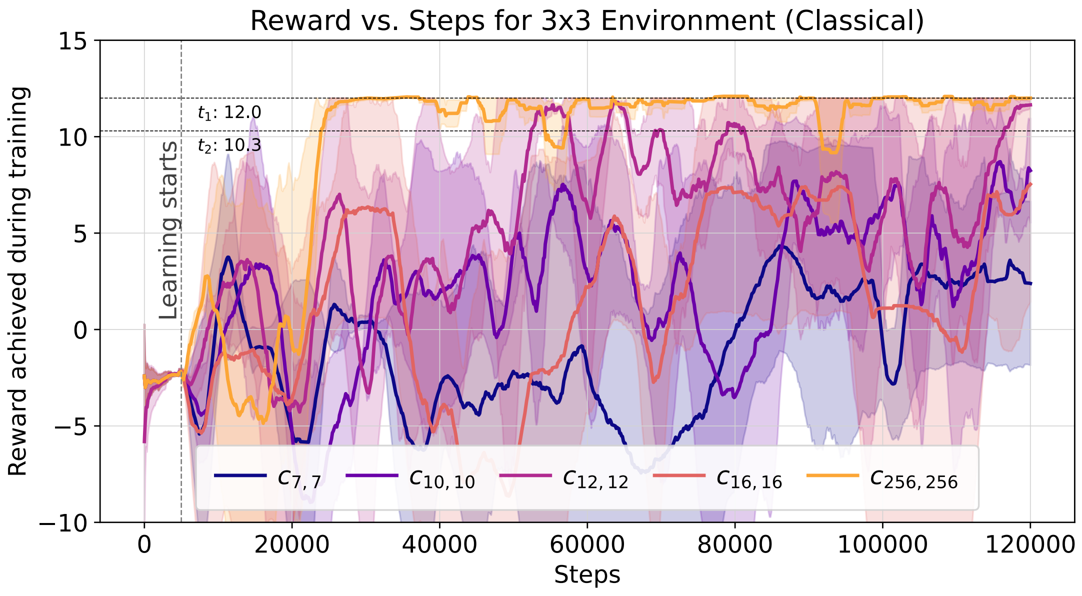


Figure 6.1.: Training performance of all classical models in the 3×3 environment.

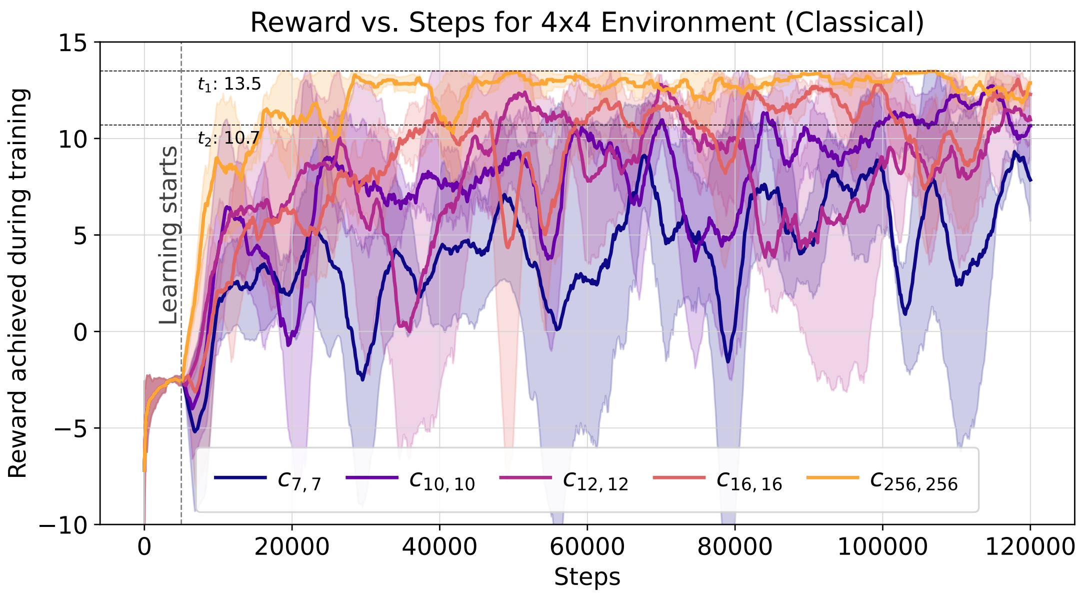


Figure 6.2.: Training performance of all classical models in the 4×4 environment.

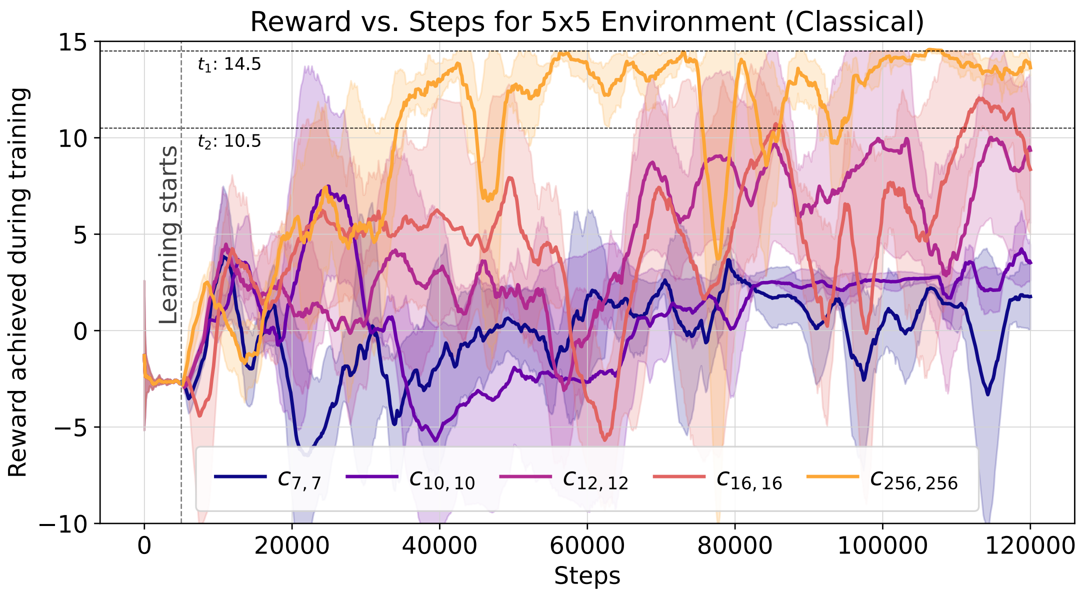


Figure 6.3.: Training performance of all classical models in the 5×5 environment.

6.2.2. Quantum Performance

Figures 6.4, 6.5, and 6.6 illustrate the training performance of the quantum models $q_{4,3}$, $q_{4,5}$, $q_{8,3}$, and $q_{8,5}$ in the 3×3, 4×4, and 5×5 environments, respectively.

In the 3×3 environment, only the $q_{8,5}$ model demonstrates significant learning capacity by reaching the t_2 threshold and approaching the t_1 threshold, albeit with considerable variability and a drop in mean reward at the end of training. The remaining quantum models ($q_{4,3}$, $q_{4,5}$, and $q_{8,3}$) remain stagnant with average rewards near zero, indicating a lack of effective learning. Among these low-performing models, $q_{4,5}$ maintains a mean reward marginally closer to the zero-reward line compared to the others. In the 4×4 environment, both $q_{8,5}$ and $q_{4,5}$ attain the t_2 threshold. The $q_{8,5}$ model also reaches the t_1 threshold multiple times, even by 60,000 steps - half of the training steps - though not persistently. The $q_{8,3}$ model initially shows promising performance by learning more rapidly than $q_{4,5}$. Still, it experiences a substantial decline around 100,000 steps, resulting in the lowest average reward at the end of training among the quantum models. The $q_{4,5}$ model maintains steady growth in average reward, while $q_{8,5}$ reaches higher thresholds but lacks consistency. The least complex model, $q_{4,3}$, fails to demonstrate concrete learning progress. In the 5×5 environment, all quantum models generally fluctuate around a mean reward of zero. The $q_{8,5}$ model stands out by raising the mean reward to approximately five, surpassing the t_2 threshold in some of its runs, though this is not consistent across all training sessions. The models $q_{4,3}$, $q_{4,5}$, and $q_{8,3}$ exhibit similarly ineffective performance in this more complex environment.

Overall, the quantum models demonstrate limited learning ability across the tested environments. While $q_{8,5}$ shows potential by achieving higher thresholds in the 3×3 and 4×4 environments, its performance is characterized by significant variability. The remaining quantum models ($q_{4,3}$, $q_{8,3}$, and $q_{4,5}$) generally fail to demonstrate significant learning, particularly in the 3×3 and 5×5 environments. However, $q_{4,5}$ performs well in the 4×4 environment. Among the four models, $q_{8,5}$ performs the best overall, while $q_{4,5}$, despite excelling only in the 4×4 environment, is considered the second-best.

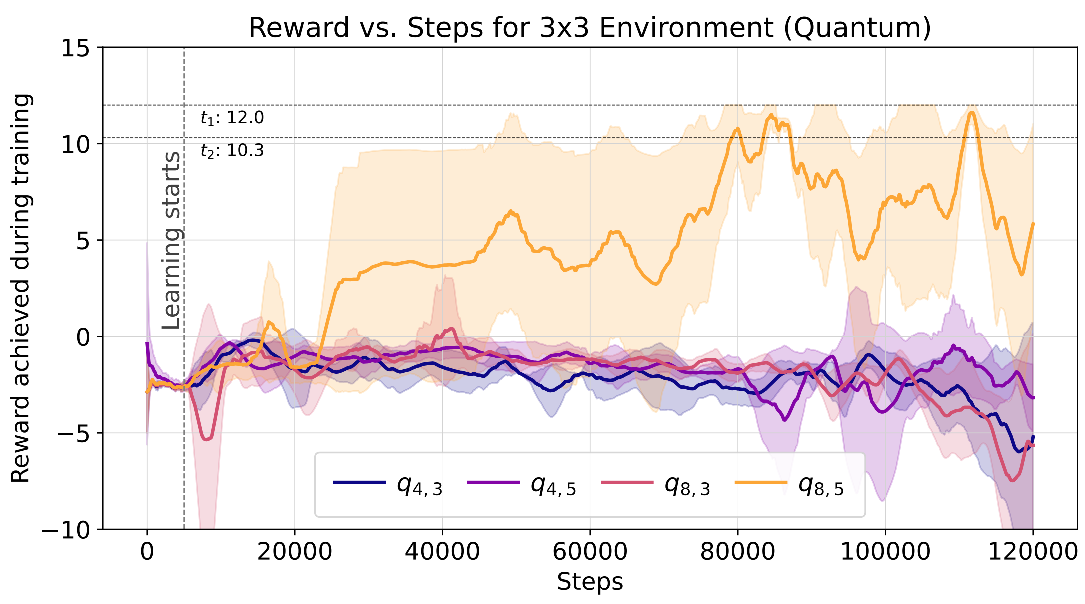


Figure 6.4.: Training performance of all quantum models in the 3×3 environment.

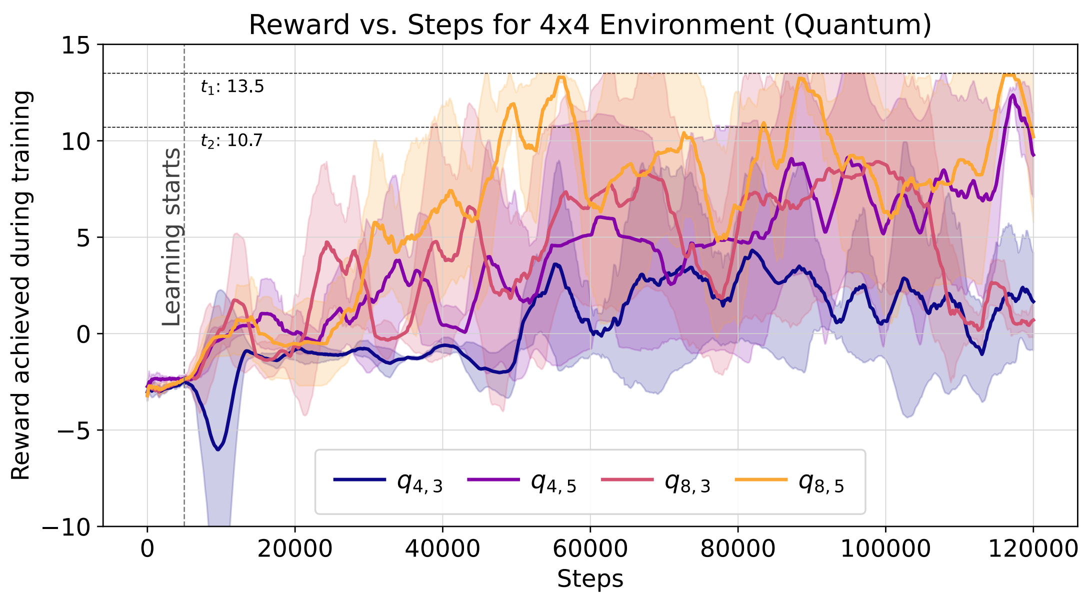


Figure 6.5.: Training performance of all quantum models in the 4×4 environment.

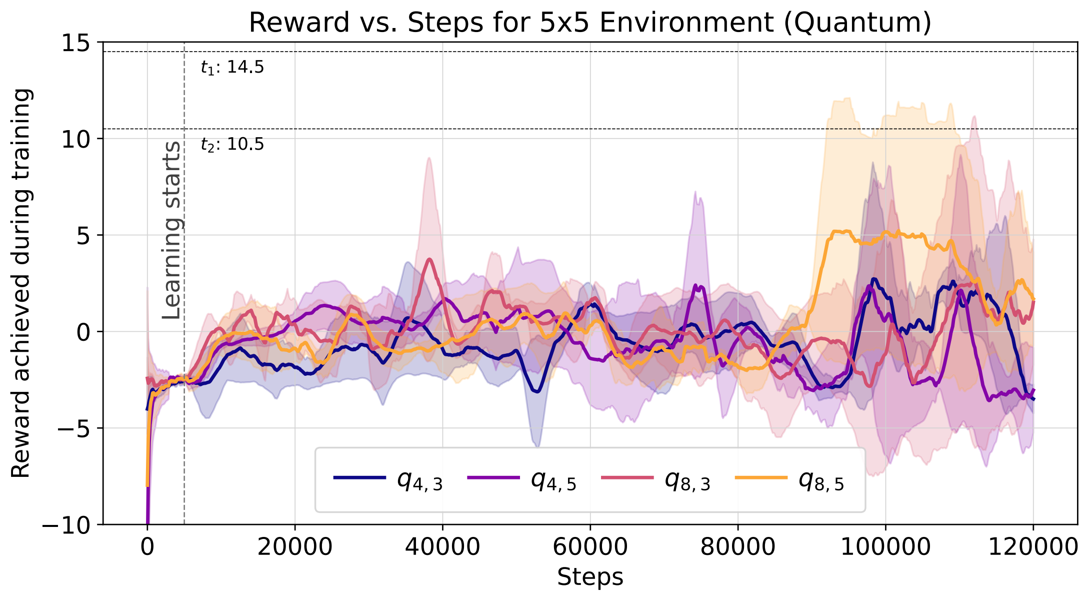


Figure 6.6.: Training performance of all quantum models in the 5×5 environment.

6.2.3. Hybrid Performance

Figures 6.7, 6.8, and 6.9 present the training performance of the hybrid models $h_{4,3}$, $h_{4,5}$, $h_{8,3}$, and $h_{8,5}$ in the 3×3, 4×4, and 5×5 environments, respectively.

In the 3×3 environment, none of the hybrid models' mean rewards reach the established thresholds. $h_{4,3}$ shows an initial rapid increase in mean reward but subsequently experiences a sharp decline and persistent oscillations throughout the training, concluding with the worst performance. The models $h_{4,5}$ and $h_{8,3}$ stabilize over time and show consistent growth in mean reward, with $h_{4,5}$ performing slightly better. The $h_{8,5}$ model maintains a steady increase in average reward, briefly declines during the later stages of training, but ultimately recovers and settles marginally below $h_{4,5}$. In the 4×4 environment, the hybrid models $h_{4,3}$, $h_{4,5}$, and $h_{8,3}$ reach the t_2 threshold, with $h_{8,5}$ coming close. All models exhibit a steep initial increase in mean reward, followed by a period of stagnation. The $h_{4,3}$ model learns the quickest but demonstrates significant fluctuations, ending with an average reward of around five. The $h_{4,5}$ model shows consistent learning initially but experiences a sharp drop in the later stages of training, resulting in the lowest final performance. Conversely, $h_{8,3}$ and $h_{8,5}$ display the most consistent training progress, with $h_{8,5}$ approaching the t_2 threshold and achieving the highest average reward near ten. In the 5×5 environment, hybrid models $h_{4,3}$ and $h_{8,3}$

perform poorly, oscillating around an average reward of zero with a negative trend in the later stages of training, though $h_{8,3}$ is more stable than $h_{4,3}$. The $h_{8,5}$ model achieves the second-highest average reward, albeit with fluctuations. The $h_{4,5}$ model outperforms the others by surpassing the t_2 threshold early on, although its performance is unstable and eventually declines slightly below the threshold.

Overall, among the hybrid models, $h_{4,3}$ exhibits the poorest average performance, despite briefly crossing the t_2 threshold in the 4×4 environment but failing to maintain it. The $h_{8,5}$ model nearly achieves the t_2 threshold in the 4×4 environment and generally outperforms $h_{8,3}$ and $h_{4,3}$ in mean reward, though it lacks robustness across environments. $h_{8,3}$ delivers comparable performance to $h_{4,5}$ in the 3×3 environment and reaches t_2 in the 4×4 environment but is also prone to instability. The most effective model is $h_{4,5}$, as it uniquely surpasses two t_2 thresholds and demonstrates strong learning in the 5×5 environment. Nonetheless, it underperforms in the 4×4 environment compared to $h_{8,5}$ and $h_{8,3}$. Overall, $h_{4,5}$ appears to perform best in training on average and excels in the largest environment (5×5), though it is not consistent across all models. $h_{8,3}$ performs better in the smaller environments, similarly to $h_{8,5}$.

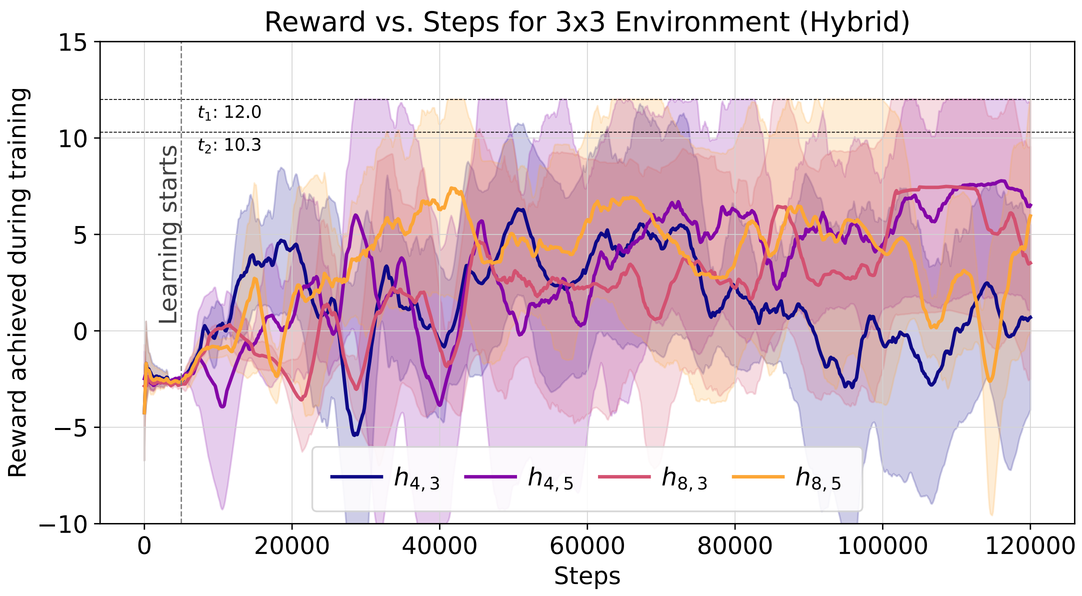


Figure 6.7.: Training performance of all hybrid models in the 3×3 environment.

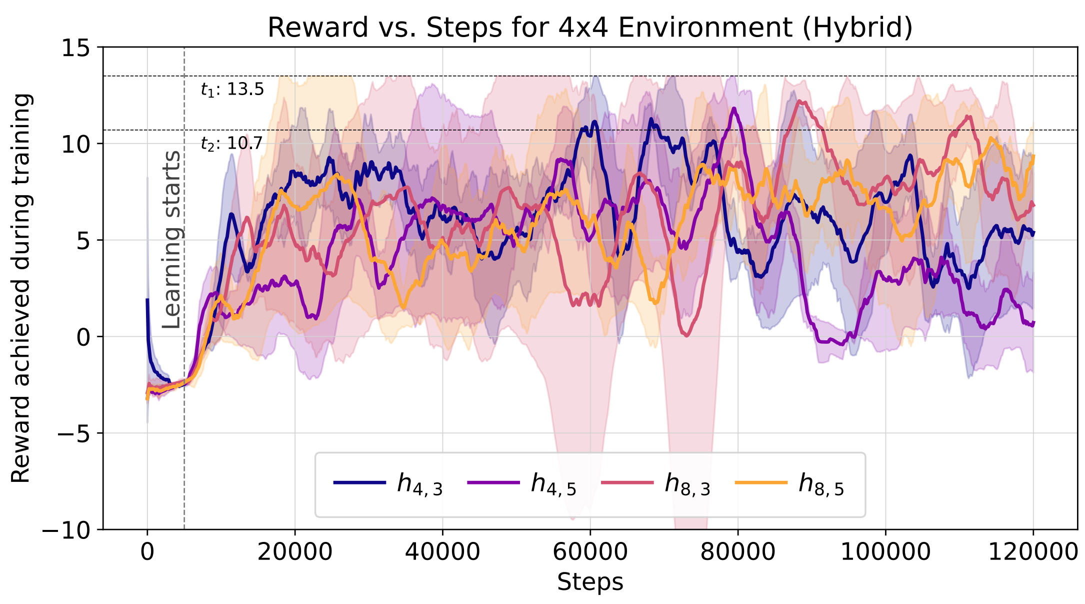


Figure 6.8.: Training performance of all hybrid models in the 4×4 environment.

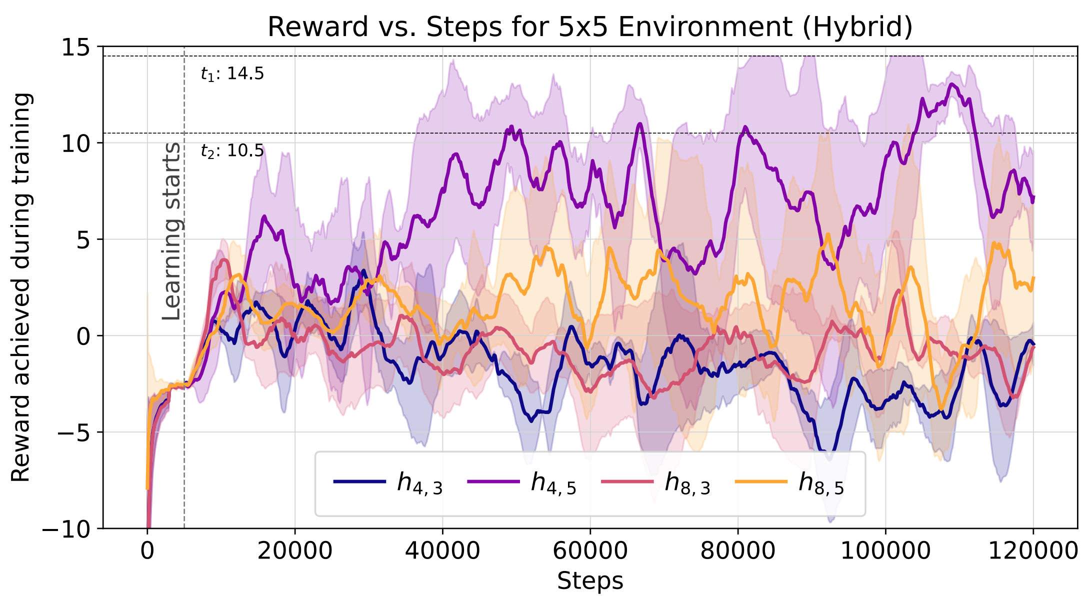


Figure 6.9.: Training performance of all hybrid models in the 5×5 environment.

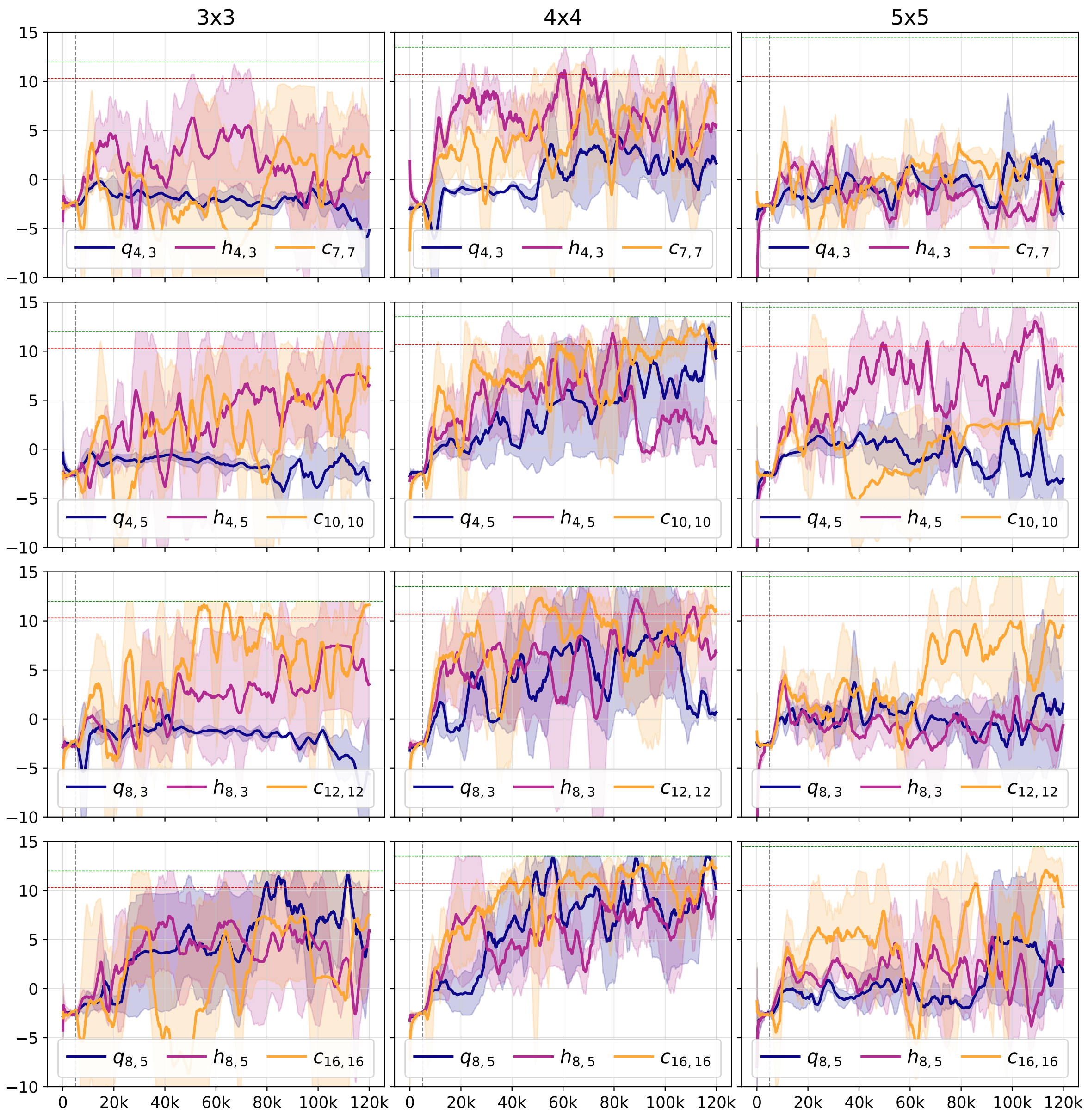


Figure 6.10.: Training performance grid plot of quantum, hybrid, and similarly weighted classical models across all environments. Thresholds t_1 and t_2 are colored in green and red, respectively.

6.3. Test Results

Following the analysis of training performances in the previous section, this section presents the test results, focusing on the performance of quantum, hybrid, and classical models across the 3×3, 4×4, and 5×5 environments. The evaluation is based on the performance data provided in Tables 6.1, 6.2, and 6.3. Models were selected for testing based on their training performance and their respective EMA peaks at each timestep, as detailed in Section 6.1. The test results are compared in relation to the t_1 and t_2 thresholds established in Section 5.2.2, and summarized in Table 6.4.

6.3.1. 3×3 Environment

A clear performance hierarchy emerges across model types in the 3×3 environment (Table 6.1). The $q_{8,5}$ model demonstrates superior performance, achieving a 100% success rate (30/30) with the highest median (12.10) and mean (12.11) rewards and the lowest standard deviation (0.06). This performance surpasses all other models, including the exhaustive classical model $c_{256,256}$, while using significantly fewer weights (851 vs. 136,451). The $q_{8,3}$ model is less successful (8/30), while $q_{4,3}$ and $q_{4,5}$ fail to achieve any successful runs, indicating a potential threshold effect in quantum model complexity for this environment. Hybrid models demonstrate consistent performance, with all configurations achieving a success rate of 60% or higher. The $h_{8,3}$ model leads with 22/30 successful runs and the highest mean reward (8.70) among hybrids. The $h_{4,5}$ model attains the highest reward median (11.47) and the lowest mean number of steps (23.95) among successful hybrid runs. Both $h_{4,5}$ and $h_{8,3}$ can be justified as top-performing category runs, depending on the assessment criteria. Due to the higher success rate and average reward, $h_{8,3}$ is selected as the best configuration. Classical models show robust performance across all configurations with a marginally lower average success rate than the hybrid runs. $c_{12,12}$ attains the highest success rate (25/30) and mean reward (9.84) among non-exhaustive classical models. The $c_{7,7}$ model, despite having the fewest weights (248), manages a 20/30 success rate and the lowest variability in steps (standard deviation of 0.0), demonstrating consistent path-finding when successful. Except for $c_{10,10}$, which has the worst performance, all other non-exhaustive models perform well, with only minor differences.

The quantum model $q_{8,5}$ outperforms all other configurations and models, excelling in almost all reward and steps-related metrics as well as success rate. Compared to the exhaustive classical model $c_{256,256}$, both $q_{8,5}$ and $c_{256,256}$ have almost identical test performances, showing only minor differences, the most significant being the minimum steps recorded for a successful run, with $c_{256,256}$ and $q_{8,5}$ having 15 and 20, respectively. Considering the more than two orders of magnitude difference in weights for the classical model, this suggests a significant difference in computational resource requirements and network complexity. Furthermore, $q_{8,5}$ achieves better results than any of the hybrid models in the 3×3 environment, indicating a substantial advantage for quantum approaches in small-scale environments. Hybrid models display similar test efficiency, with lower median reward and step metrics, though more consistent across all models.

Table 6.1.: Performance comparison of quantum (q), hybrid (h), and classical (c) models in the 3×3 environment. $c_{256,256}$ is included for reference but is not considered in the selection of top-performing models. Each configuration was evaluated with three seeds, conducting 10 runs per seed based on the peak EMA (decay factor = 0.99), totaling 30 runs per model. The table includes the model identifier, total weights (θ), successful runs (SR) out of 30 (SR/30), and statistics (min, max, median, mean, Std) for rewards and steps (steps calculated from SR only). Models without successful runs display *nan* for steps. Best metrics are highlighted in bold, top-performing models are shaded green, and models with fewer than five successful runs are marked in red and excluded from metric highlighting.

			Reward					**Steps (SR-based)**				
Model	θ	**SR/30**	**Min**	**Max**	**Median**	**Mean**	**Std**	**Min**	**Max**	**Median**	**Mean**	**Std**
$q_{4,3}$	267	0	-1.22	-0.27	-0.28	-0.53	0.38	*nan*	*nan*	*nan*	*nan*	*nan*
$q_{4,5}$	427	0	-1.46	0.62	0.34	-0.05	0.63	*nan*	*nan*	*nan*	*nan*	*nan*
$q_{8,3}$	531	8	-1.73	12.09	-0.24	2.64	5.21	24	28	27.0	26.75	1.39
$q_{8,5}$	851	**30**	**12.05**	**12.38**	**12.10**	**12.11**	**0.06**	**20**	**23**	**21.0**	**21.37**	**1.25**
$h_{4,3}$	381	18	-0.34	**12.16**	11.05	7.16	5.60	**23**	29	**23.0**	25.17	2.55
$h_{4,5}$	541	20	-6.11	12.07	**11.47**	6.35	7.84	**23**	**26**	24.0	**23.95**	**1.05**
$h_{8,3}$	645	**22**	**0.35**	12.10	11.34	**8.70**	**4.99**	24	29	27.0	27.41	1.44
$h_{8,5}$	965	20	-0.29	12.11	10.20	7.27	5.48	**23**	32	26.0	26.60	2.84
$c_{7,7}$	248	20	0.18	12.40	**12.05**	8.22	5.57	**19**	**19**	**19.0**	**19.00**	**0.00**
$c_{10,10}$	413	14	-0.82	12.32	0.65	5.55	6.17	**19**	21	20.0	19.71	0.61
$c_{12,12}$	543	**25**	**0.54**	12.41	11.73	**9.84**	**4.24**	**19**	22	20.0	20.56	1.45
$c_{16,16}$	851	20	-1.64	**12.42**	**12.05**	7.60	6.48	**19**	22	20.0	20.05	1.10
$c_{256,256}$	136,451	30	12.01	12.41	12.10	12.12	0.09	15	21	19.0	18.13	1.70

6.3.2. 4×4 Environment

The analysis of the 4×4 environment (Table 6.2) reveals a shift in performance dynamics. Both $q_{4,5}$ and $q_{8,5}$ achieve perfect success rates (30/30), with $q_{4,5}$ demonstrating the highest overall performance across all models and configurations. It exhibits the highest values for minimum (13.41), maximum (13.80), median (13.49), and mean (13.50) rewards among the quantum models, coupled with the lowest standard deviation (0.06). The $q_{8,5}$ model follows closely, with nearly identical performance metrics. In contrast, the least complex model, $q_{4,3}$, performs poorly, achieving only 2 successful runs out of 30, while $q_{8,3}$ manages a decent 20/30 success rate but with slightly lower overall reward and step statistics. Hybrid models continue to show consistent performance, with all variants achieving at least 20/30 successful runs. The $h_{8,5}$ model is the top performer, leading with 28/30 successful runs and the highest median (13.43) and mean (11.77) rewards among hybrids. $h_{4,3}$ records the lowest mean number of steps (33.10) among all models with substantial success rates. However, all hybrid models display moderate to good results, except in step-related metrics, where models with a higher number of layers tend to show weaker path-finding ability. Similar to the hybrids, classical models achieve at least 20/30 successful runs. Among non-exhaustive classical models, $c_{12,12}$ excels with 26/30 successful runs and competitive reward metrics. $c_{12,12}$ achieves the highest mean reward (11.28), while $c_{16,16}$ performs best in step metrics, with a median and mean step count of 29.50 and the lowest step standard deviation (0.51). $c_{10,10}$ performs well with 24/30 successful runs and matches $q_{4,5}$ and $q_{8,3}$ for the highest reward median (13.49). However, it performs worse in step-related rewards and has the lowest minimum reward across all configurations and environments (-15.72).

In this environment, the quantum models $q_{4,5}$ and $q_{8,5}$ demonstrate superior performance, surpassing even the exhaustive classical model in reward metrics while using far fewer weights, similar to the 3×3 environment. This suggests that quantum approaches may also offer significant advantages in medium-scale environments. Hybrid and classical models, while slightly underperforming compared to the quantum models in reward metrics, exhibit more stable performance, particularly in step consistency and lower variability across runs. Hybrid models show comparable performance to the slightly better-performing classical models with similar weight counts, though they fall short compared to the pure quantum models.

6.3.3. 5×5 Environment

The 5×5 environment (Table 6.3) presents the largest and most challenging setting. Quantum models show mixed results in this environment. The $q_{4,3}$ model achieves

Table 6.2.: Performance comparison of quantum (q), hybrid (h), and classical (c) models in the 4×4 environment. $c_{256,256}$ is included for reference but is not considered in the selection of top-performing models. Each configuration was evaluated with three seeds, conducting 10 runs per seed based on the peak EMA (decay factor = 0.99), totaling 30 runs per model. The table includes the model identifier, total weights (θ), successful runs (SR) out of 30 (SR/30), and statistics (min, max, median, mean, Std) for rewards and steps (steps calculated from SR only). Models without successful runs display *nan* for steps. Best metrics are highlighted in bold, top-performing models are shaded green, and models with fewer than five successful runs are marked in red and excluded from metric highlighting.

			Reward					Steps (SR-based)				
Model	θ	**SR/30**	**Min**	**Max**	**Median**	**Mean**	**Std**	**Min**	**Max**	**Median**	**Mean**	**Std**
$q_{4,3}$	267	2	-1.11	13.44	0.36	1.49	3.35	30	30	30.0	30.00	0.0
$q_{4,5}$	427	**30**	**13.41**	**13.80**	**13.49**	**13.50**	**0.06**	31	41	**35.0**	35.93	3.89
$q_{8,3}$	531	20	-1.83	13.79	**13.49**	8.79	6.71	34	41	36.5	36.95	**3.05**
$q_{8,5}$	851	**30**	12.95	13.51	13.45	13.31	0.22	**28**	**36**	**35.0**	**33.10**	3.49
$h_{4,3}$	381	21	-1.07	13.51	13.30	9.31	6.18	**30**	39	**35.0**	**33.10**	3.11
$h_{4,5}$	541	20	-0.50	13.49	11.65	8.15	6.05	34	46	42.0	40.75	5.46
$h_{8,3}$	645	20	**0.15**	13.51	13.05	8.96	6.24	34	**36**	36.0	35.55	**0.60**
$h_{8,5}$	965	**28**	-2.24	**13.56**	**13.43**	**11.77**	**3.85**	39	44	40.0	40.61	1.75
$c_{7,7}$	248	20	-1.70	**13.84**	13.43	8.44	7.26	29	32	32.0	31.40	0.94
$c_{10,10}$	413	24	-15.72	13.83	**13.49**	7.67	11.01	32	83	37.0	37.0	10.09
$c_{12,12}$	543	**26**	0.28	**13.84**	13.39	**11.28**	**4.43**	**28**	33	31.0	30.58	2.25
$c_{16,16}$	851	20	**0.70**	13.80	13.43	9.24	6.11	29	**30**	**29.50**	**29.5**	**0.51**
$c_{256,256}$	136,451	30	12.79	13.80	13.40	13.31	0.27	28	36	30.0	30.93	2.24

the highest success rate among quantum models (13/30) and the highest mean reward (6.50), but its median reward (2.70) suggests inconsistent performance. $q_{8,3}$ and $q_{8,5}$ follow with 10 successful runs out of 30. $q_{8,5}$ achieves the highest median reward (2.94) among quantum models. $q_{4,5}$ performs the worst with only 6/30 successful runs. The best step metrics come from the $q_{8,5}$ model, with a standard deviation of 0.00 and both the median and mean steps at 52.0. The $h_{4,5}$ model emerges as the top performer across all configurations and models, with 28/30 successful runs, the highest mean reward (13.12), and a competitive median reward (14.20). It also achieves the lowest median (50.0) and mean (51.71) steps among all VQC-based models. The $h_{4,3}$ model also performs well, with 18/30 successful runs and the highest median reward (14.32) across all models. $h_{4,3}$ is sufficient in performance (14/30) but underperforms in other metrics. $h_{8,3}$ performs the worst, with only one successful run. Classical models also display mixed performances but are generally more stable. The $c_{16,16}$ model leads with 26/30 successful runs, a high median reward (14.08), and a mean reward of 11.99. $c_{12,12}$ achieves 20/30 successful runs and the highest median reward (14.63) across all models and configurations. The exhaustive $c_{256,256}$ model maintains its perfect 30/30 success rate and achieves the highest overall median (14.90) and mean (14.81) rewards, reasserting its advantage in even the most challenging environment.

In the 5×5 environment, hybrid models, particularly $h_{4,5}$, demonstrate superior performance, outperforming most classical and all quantum models in success rate, reward metrics, and overall performance. This suggests that hybrid approaches may offer a balanced solution for larger-scale environments. Classical models show a more consistent performance, with the exhaustive $c_{256,256}$ continuing to excel, though at the cost of substantially higher computational requirements. The classical models, in general, are more efficient in path-finding in the 5×5 environment, as indicated by the generally lower mean and median steps.

6.4. Cross-Environment Training and Test Performance Analysis

This section summarizes the key findings of the quantum, hybrid, and classical ansatzes in both the training and testing performances across all environments. The presented information is drawn from the training results (cf. Section 6.2), particularly from Figure 6.10, which shows a grid with columns representing the different environments (3×3, 4×4, and 5×5) and rows corresponding to models from the ansatzes with a comparable number of weights. The test results (cf. Section 6.3) are mainly focused on the performance tables (cf. Tables 6.1 - 6.3) and the summary of thresholds reached (cf. Table 6.4). After summarizing the results, this section proceeds to interpret the training and testing performances, identifying trends across different ansatzes, models, and environments. The distribution and total number of weights for each architecture are documented in Table 5.1 and are also listed in the test performance tables.

6.4.1. Training Summary

eThe mean reward training performance of the quantum models in the 3×3 environment tends to show limited learning ability, as indicated in Figure 6.10. Specifically, the quantum models with only three layers ($q_{4,3}$ and $q_{8,3}$) perform worse than the setups with five layers. Increasing the number of weights by using more qubits does not seem to impact performance as much as increasing the number of layers. Only the model with 8 qubits and 5 layers ($q_{8,5}$) demonstrates acceptable performance, while

Table 6.3.: Performance comparison of quantum (q), hybrid (h), and classical (c) models in the 5×5 environment. $c_{256,256}$ is included for reference but is not considered in the selection of top-performing models. Each configuration was evaluated with three seeds, conducting 10 runs per seed based on the peak EMA (decay factor = 0.99), totaling 30 runs per model. The table includes the model identifier, total weights (θ), successful runs (SR) out of 30 (SR/30), and statistics (min, max, median, mean, Std) for rewards and steps (steps calculated from SR only). Models without successful runs display *nan* for steps. Best metrics are highlighted in bold, top-performing models are shaded green, and models with fewer than five successful runs are marked in red and excluded from metric highlighting.

			Reward					Steps (SR-based)				
Model	θ	**SR/30**	**Min**	**Max**	**Median**	**Mean**	**Std**	**Min**	**Max**	**Median**	**Mean**	**Std**
$q_{4,3}$	267	**13**	-2.37	14.71	2.70	**6.50**	7.10	56	62	62.0	60.62	2.63
$q_{4,5}$	427	6	**0.79**	**14.97**	2.79	4.56	**5.16**	54	59	55.0	56.00	2.00
$q_{8,3}$	531	10	-0.43	14.72	1.97	5.68	6.31	56	58	57.5	57.10	0.99
$q_{8,5}$	851	10	-5.63	14.96	**2.94**	4.97	7.61	**52**	**52**	**52.0**	**52.00**	**0.00**
$h_{4,3}$	381	18	-1.83	**14.93**	**14.32**	9.52	6.33	55	63	57.0	56.72	**1.84**
$h_{4,5}$	541	**28**	**2.63**	14.90	14.20	**13.12**	**2.97**	**46**	**60**	**50.0**	**51.71**	5.56
$h_{8,3}$	645	1	-0.52	14.83	1.04	1.75	2.70	48	48	48.0	48.00	*nan*
$h_{8,5}$	965	14	-4.64	13.89	1.46	4.65	7.19	55	74	55.0	60.64	7.96
$c_{7,7}$	248	10	-1.76	15.14	2.86	5.85	6.65	42	56	45.5	46.20	4.89
$c_{10,10}$	413	14	**0.38**	13.77	3.11	6.85	6.03	51	65	51.0	55.00	6.41
$c_{12,12}$	543	20	-1.98	**15.22**	**14.63**	10.03	6.69	**37**	57	**42.0**	**43.35**	4.65
$c_{16,16}$	851	**26**	-7.07	14.89	14.08	**11.99**	**5.87**	45	**52**	50.0	49.54	**1.73**
$c_{256,256}$	136,451	30	14.37	15.22	14.90	14.81	0.24	39	45	44.0	42.93	2.38

the remaining, less complex models underperform. In the 4×4 environment, a similar tendency is observed. Again, $q_{4,5}$ and especially $q_{8,5}$ perform better than the models with fewer layers in terms of mean reward, with the 8-qubit models learning faster than their 4-qubit counterparts but also showing higher standard deviation. In general, all quantum models, except for $q_{4,3}$, demonstrate good performance, with $q_{8,3}$ showing learning ability, though there is a drop in performance near the end of training. The performance of the quantum models in the 5×5 environment is less clear. Here, the quantum models struggle to adapt to the environment, with only moderate results observed for $q_{8,5}$. Both 4-qubit models underperform, while $q_{8,3}$ shows marginally better performance. In this environment, models with more qubits and layers tend to perform

Table 6.4.: Performance of quantum (q), hybrid (h), and classical (c) models across 3×3, 4×4, and 5×5 environments, assessed against the t_1 and t_2 threshold (cf. Section 5.2.2). Crosses (×) indicate models where the mean test reward successfully met the specified threshold, as documented in Tables 6.1, 6.2, and 6.3.

		3×3		4×4		5×5	
Model	$\boldsymbol{\theta}$	$\boldsymbol{t_1}$ (12.0)	$\boldsymbol{t_2}$ (10.3)	$\boldsymbol{t_1}$ (13.5)	$\boldsymbol{t_2}$ (10.7)	$\boldsymbol{t_1}$ (14.5)	$\boldsymbol{t_2}$ (10.5)
$q_{4,3}$	267						
$q_{4,5}$	427			×	×		
$q_{8,3}$	531						
$q_{8,5}$	851	×	×		×		
$h_{4,3}$	381						
$h_{4,5}$	541						×
$h_{8,3}$	645						
$h_{8,5}$	965				×		
$c_{7,7}$	248						
$c_{10,10}$	413						
$c_{12,12}$	543				×		
$c_{16,16}$	851						×
$c_{256,256}$	136,451	×	×		×	×	×

better, particularly $q_{8,5}$, where the variation suggests that some seeds successfully solve the environment.

A similar analysis for the hybrid models, based on their training performance in Figure 6.10, shows that all models except for $h_{4,3}$ perform reasonably well in the 3×3 environment. Notably, $h_{4,5}$ and $h_{8,3}$ exhibit stable and constant growth in mean reward during the latter half of training. While $h_{8,5}$ outperforms the least complex model $h_{4,3}$, it shows large oscillations, recovering at the end with moderate performance. A similar observation is made in the 4×4 environment. The overall performance of the hybrid models increases in this setup. $h_{4,3}$ starts with a strong learning curve but concludes with only a moderate mean reward. All hybrid configurations display steeper learning curves than the quantum models, but their performance is less stable, with oscillations in mean reward across all models—less pronounced in $h_{8,5}$. The best performances in this environment are achieved by $h_{8,3}$ and $h_{8,5}$. In the 5×5 environment, $h_{4,5}$ clearly outperforms the other hybrid models, showing the best performance overall. $h_{8,5}$ also performs well, though not as strongly as $h_{4,5}$, but significantly better than the remaining models. In contrast, $h_{8,3}$ and $h_{4,3}$ show weaker performance and do not demonstrate a noteworthy learning ability in this environment. A higher number of layers appears to be beneficial, with models that have fewer qubits and more layers ($h_{4,5}$) showing the best results.

Classical models were chosen based on a similar number of weights as the VQC in the hybrid and quantum models. In the 3×3 environment, all classical models outperform the quantum models, except for $c_{16,16}$, which is outperformed by $q_{8,5}$. In this case, the quantum model exceeds both the classical and hybrid models regarding stability and average mean reward. The less complex classical models, $c_{7,7}$ and $c_{10,10}$, display unstable but comparable performance to the hybrid models. Only $c_{12,12}$ clearly achieves higher average results than the hybrid models, but with less stability. The results are less definitive in the 4×4 environment. Classical models conclude training with the highest mean reward, but they also exhibit more oscillations compared to the quantum models and, except for $h_{4,5}$ and $c_{10,10}$, also more than the hybrid models. Nevertheless, all classical models show good performance, with $c_{10,10}$ and $c_{16,16}$ performing best in the 4×4 environment. Finally, the classical models also face challenges in the 5×5 environment. The $c_{7,7}$ model performs similarly to the quantum and hybrid models, $c_{10,10}$ performs slightly worse than $h_{4,5}$ and marginally better than $q_{4,5}$, and $c_{12,12}$ outperforms both $q_{8,3}$ and $h_{8,3}$. Finally, $c_{16,16}$ performs better than both $q_{8,5}$ and $h_{8,5}$ but also exhibits large fluctuations in mean reward.

6.4.2. Test Summary

Similarly to the training results, a clear trend is observed where increasing model complexity corresponds to higher test performance. This trend is particularly evident in the 3×3 environment, where the quantum models with 8 qubits outperform all other models. Notably, the $q_{8,5}$ model surpasses both hybrid and classical models, including the exhaustive $c_{256,256}$, which has more than two orders of magnitude greater weights. In the 4×4 environment, quantum models with 5 layers, whether with 4 or 8 qubits, exhibit strong performance, with the $q_{8,3}$ model also yielding competitive results. The simplest quantum configuration fails to achieve success in this environment. In the 5×5 environment, this trend is less pronounced, as the $q_{4,3}$ model achieves the most successful runs. However, the $q_{8,3}$ and $q_{8,5}$ models perform comparably, indicating that the general observation from earlier environments still holds. Table 6.4 summarizes these observations, highlighting the advantage of more complex quantum models, especially those with a greater number of layers, in the 3×3 and 4×4 environments.

Hybrid models demonstrate more consistent and robust performance across all environments, generally aligning more closely with classical results than with quantum. However, as shown in Table 6.4, hybrid models do not consistently reach the t_1 and t_2 thresholds. The hybrid models display stronger learning ability in the more complex 3×3 and 5×5 environments. The complexity of each environment is empirically demonstrated in Figure 6.10, which shows that all models achieved their best results in the 4×4 environment, followed by the 3×3, and lastly the 5×5 environment. Higher complexity in hybrid models, particularly those with more qubits and layers, reflects a trend similar to that of quantum models, though the impact is somewhat less pronounced. This is reinforced by Table 6.4, where only the $h_{4,5}$ and $h_{8,5}$ models meet the t_2 thresholds in the 4×4 and 5×5 environments, respectively.

For classical models, testing results demonstrate a more linear and predictable increase in performance with increasing complexity. Classical models show a clear, calculable progression, where higher complexity generally yields better results. Nevertheless, some minor variations are observed, as seen in Tables 6.1 and 6.2, where the smaller $c_{12,12}$ model outperforms the $c_{16,16}$ model in terms of successful runs in both the 3×3 and 4×4 environments, though not necessarily in reward or step metrics.

6.4.3. Interpretation and Trends of Summaries

This section examines the performance trends of quantum, hybrid, and classical models across different environment complexities in robot navigation tasks, contextualizing the findings with those from [73]. The analysis highlights key patterns and observations regarding the capabilities of each model type in these environments.

Increasing VQC complexity, particularly through additional layers, enhances training and test performance, indicating that more complex circuit structures outperform simpler models. This improvement is observable in both training and testing phases, aligning with the findings of [73], which indicate that a quantum-based approach demonstrates learning capability and effectively solves the robotic navigation RL environment. Additionally, [73] notes that adding more layers by incorporating additional data encoding gates per qubit improves performance, which corresponds with the observation of this thesis that increased VQC complexity leads to better outcomes. More specifically, configurations with 8 qubits showed, on average, better results but were marginal compared to the performance improvement of adding more layers. A similar trend is noted in hybrid ansatzes; however, the impact is less pronounced compared to purely quantum models. This may be attributed to the configuration of the hybrid ansatz, where a classical layer precedes the VQC. The classical component potentially mitigates the slower learning capabilities of the VQC, as indicated by training plots where the hybrid model positions itself between classical and quantum models, offering balanced performance, particularly in more complex environments. Although hybrid models possess slightly more parameters than their quantum and classical counterparts, this interpretation requires empirical validation. Furthermore, a quantum advantage is observed in the number of weights when comparing the test results to the exhaustive $c_{256,256}$ model, emphasizing the efficiency of quantum approaches in parameter usage, especially when additional data encoding layers are introduced. This finding aligns with [73], demonstrating that quantum models can achieve comparable performance with significantly fewer trainable parameters.

Analyzing the environments reveals that training and test results indicate that the complexity of the environment, specifically the function approximation capability of the models, does not follow a straightforward hierarchy with increasing environment size. The 5×5 environment is the most challenging, followed by the 3×3 environment, while the 4×4 environment is the least difficult and consistently yields the best results across all model types. This pattern suggests that environmental complexity significantly influences model performance. Deeper QCs excel in simpler environments, while their advantages decline in more complex settings. Hybrid models also perform better in smaller environments, with their limitations becoming apparent in more complex scenarios. Classical models demonstrate the least variability in performance across different environments, indicating superior scalability for larger and more complex tasks. Additionally, performance tables reveal that, on average, quantum models perform well in reward-related metrics for successful configurations. However, their step-related metrics are inferior to those of hybrid models and even more so compared to classical models, suggesting that the VQC approach may be lacking in achieving near-optimal steps. Nevertheless, hybrid and quantum approaches demonstrate the ability to solve the environments under study, even with the added complexity of continuous action and state spaces.

To conclude, this chapter outlines several important findings that highlight the performance patterns of quantum, hybrid, and classical models in robot navigation tasks across different levels of environmental complexity:

- Quantum and hybrid approaches are capable of solving the tested environments with local features and continuous action and state spaces.
- Increased VQC complexity, particularly through adding more layers, leads to

better training and test performance, with an increase in qubits also contributing positively, though with less impact.

- Classical models exhibit more consistent performance and linear scalability across different environment complexities.
- Hybrid ansatzes offer balanced performance between classical and quantum models, especially in complex environments.
- The training environments' complexities are not proportional to size, as the 5×5 environment is the most challenging, while the 4×4 environment is the least.
- Successful quantum models excel in reward-related metrics but underperform in step-related metrics compared to hybrid and classical models, with classical models performing best in step-related metrics.
- A potential quantum advantage in parameter efficiency is observed in some environments.

7. Conclusion and Future Work

This thesis builds upon the work of [73] by investigating the applicability of quantum reinforcement learning using the deep deterministic policy gradient (DDPG) algorithm, an actor- ritic approach, in a more complex robot navigation framework. This study explores a simulated robot navigation setup characterized by continuous action and state spaces across three distinct environments. Each environment incorporates only local features, aligning with the requirements of realistic industrial implementations of robot navigation in various scenarios. A variational quantum circuit (VQC), analogous to those in [73] and [23], was employed, utilizing a horizontal-circulating data encoding.

A comprehensive series of experiments was designed to evaluate and compare the performance of VQC-based function approximation in both quantum and hybrid configurations against a classical baseline. The quantum and hybrid models comprised configurations utilizing 4 and 8 qubits, with 3 and 5 layers, respectively. In total, 24 configurations were examined, including 4 quantum, 4 hybrid, and 16 classical models. Among these, all VQC-based and 5 classical models were additionally tested across three environments of different complexity. Each configuration was tested using three random seeds, resulting in a total of 117 models. These models underwent 10 evaluation runs, culminating in 1170 evaluation runs. Both quantum and hybrid models successfully trained deterministic policies capable of navigating the various tested environments, revealing distinct characteristics and behaviors. Notably, an increase in performance was observed for more complex VQC-based models, where an increase in both qubit count and layer size enhanced performance, with the latter exerting a more significant positive impact. Models $q_{8,5}$ and $q_{4,5}$ exhibited outstanding results in the majority of environments, surpassing the reward metrics of classical and hybrid setups with comparable weight configurations. Furthermore, these quantum models outperformed a classical model with two orders of magnitude more parameters, namely $c_{256,256}$. This observation indicates better parameter efficiency for certain quantum models in less complex environments. Hybrid models demonstrated more consistent performance across all environments, albeit with lower peak performance than their quantum counterparts. Compared to quantum models, hybrids' more rapid learning ability may be attributed to the preceding classical neural network, integrating characteristics of both quantum and classical components. Classical components exhibited more robust and predictable performance scaling in relation to an increase in weights, as experienced in [73]. In contrast, quantum models did not demonstrate a consistent performance increase when scaling weights, as evidenced by the superior performance of $q_{4,5}$ relative to $q_{8,3}$. While extensive manual testing was conducted during the development and assessment of VQCs, the resources for hyper-parameter testing were limited due to the prolonged training times required for larger quantum circuits. Training VQCs is more sensitive to hyper-parameter adjustments than classical models, a phenomenon noted in this study and affirmed by other research, such as [79] and [73]. The hardware and noise constraints of quantum computers, combined with insufficient error mitigation in the noisy intermediate-scale quantum (NISQ) era, render the implementation of the VQC utilized in this thesis impractical on real quantum hardware. Consequently, the performance of this VQC cannot be verified or tested on actual quantum systems. While this work's DDPG employs Gaussian noise as an exploration strategy, the simulated environment introduces additional noise due to motion carryover; however, the VQC was applied as an idealized, noise-free circuit.

A potential avenue for enhancing results lies in the application of more advanced Critic-Actor methods, such as twin-delayed DDPG (TD3), which addresses the inherent challenge of overestimating the value function, leading to sub-optimal policies. TD3 seeks to mitigate this issue by learning multiple value functions and selecting the lowest value for policy updates, along with less frequent updates of the policy and target networks [80]. While this work offers insights into the scalability of VQCs within the domain of quantum reinforcement learning (QRL), further extensive tests are called for, particularly with increased layer sizes of data encoding gates. Experiments with 4 qubits and 7 layers, utilizing an identical VQC setup, yielded promising results but were not included due to time constraints. Future work could also benefit from a more thorough analysis of barren plateaus, a phenomenon analogous to the vanishing gradient problem encountered in classical algorithms [72]. Although quantum algorithms have demonstrated superiority over classical implementations in specific domains (cf. Chapter 3) and quantum machine learning (QML) [39, 40, 41], the applicability and utility of quantum-enhanced reinforcement learning in the robotics domain remain unresolved.

A. Appendix

A.1. Hyper-Parameter Details for DDPG

This section provides an overview of the hyper-parameters used in the experiments, as given in Table A.1, followed by a brief explanation. Although some of these parameters are addressed in Section 2.3.2, they are listed here in full for completeness.

Weight Initialization Scale: This scaling factor is applied during the initialization of the model's randomly chosen weights. It influences the starting point of the training process, potentially affecting the convergence speed and stability.

Random Seed: The seed value initializes the random number generator, ensuring the experimental results are reproducible. Consistent seeding allows for the replication of results across different runs.

Total Timesteps: Defines the total number of timesteps for which the experiments are conducted, determining the duration of the training phase.

Learning Rate: The learning rate for the optimizer is responsible for updating the weights of the circuit. It dictates the step size during gradient descent, balancing convergence speed against the risk of overshooting minima.

Output Scaling Learning Rate: The learning rate for the optimizer tasked with adjusting the output scaling. This learning rate is only used for the quantum and hybrid ansatzes.

Replay Buffer Size: The size of the replay memory buffer used to store experience tuples for training. A larger buffer size allows the model to learn from a more diverse set of past experiences, enhancing generalization.

Learning Start Timestep: The number of timesteps to collect experience before commencing the learning process. This allows the replay buffer to be pre-filled with diverse experiences.

Discount Factor: The discount factor in the DDPG algorithm determines the importance of future rewards compared to immediate rewards. A value close to 1 emphasizes long-term rewards, encouraging the agent to consider future benefits.

Target Network Update Rate: This coefficient controls the rate at which the target networks are updated towards the main networks. A smaller value ensures smoother and more stable updates.

Batch Size: The number of samples drawn from the replay memory buffer during each training step. A larger batch size can provide a more accurate gradient estimate but requires more computational resources.

Exploration Noise Scale: The scale of the Gaussian noise (with a mean of 0) added to the policy's actions, where the exploration noise controls the standard deviation of the added noise. This encourages the agent to explore the action space and helps prevent premature convergence to suboptimal policies.

Policy Update Frequency: The frequency at which the policy network is updated. A delayed update strategy can help stabilize training by ensuring the policy does not change too rapidly.

Table A.1.: Hyper-parameters, thresholds and number of near-optimal (NO) steps for the classical and quantized DDPG algorithm across different environments (3×3, 4×4, 5×5). Thresholds are defined in Section 5.2.2.

Hyper-Parameter	**3×3**	**4×4**	**5×5**
Number of Seeds	3	3	3
Evaluation Runs per Seed	10	10	10
Total Timesteps	120,000	120,000	120,000
Learning Rates			
Circuit	0.001	0.001	0.001
Output Scaling (VQC)	0.01	0.01	0.01
Classical	0.001	0.001	0.001
θ Initialization Scaling	1.0	1.0	1.0
Replay Buffer Size	20,000	20,000	20,000
Discount Factor (γ)	0.99	0.99	0.99
Target Smoothing (τ)	0.005	0.005	0.005
Batch Size	64	64	64
Exploration Noise	0.05	0.05	0.05
Learning Starts	5,000	5,000	5,000
Policy Update Frequency	2	2	2
Metrics			
Threshold t_1	12.0	14.5	13.5
Threshold t_2	10.3	10.7	10.5
Number of NO-Steps (t_1)	17	28	40
Number of NO-Steps (t_2)	25.5 (26)	42	60

List of Figures

List of Tables

List of Abbreviations

A2C	Advantage Actor-Critic
A3C	Asynchronous Advantage Actor-Critic
AI	Artificial Intelligence
CNOT	Controlled-NOT (Gate)
CZ	Controlled-Z (Gate)
DDPG	Deep Deterministic Policy Gradient
DDQN	Double Deep Q-Network
DQN	Deep Q-Learning/Network
DRL	Deep Reinforcement Learning
DRL-PER	Prioritized Experience Replay
ECM	Episodic and Compositional Memory
EMA	Exponential Moving Average
HER	Hindsight Experience Replay
I2A	Imagination-Augmented Agents
MARL	Multi-Agent Reinforcement Learning
MB	Model-Based
MBMF	Model-Based Model-Free
MBVE	Model-Based Value Expansion
MC	Monte Carlo
MCTS	Monte Carlo Tree Search
MDP	Markov Decision Process
ML	Machine Learning
MSBE	Mean Squared Bellman Error
NISQ	Noisy Intermediate-Scale Quantum
NN	Neural Network
NO	Near-Optimal
OU	Ornstein-Uhlenbeck
POMDP	Partially Observable Markov Decision Process
PPO	Proximal Policy Optimization
PQC	Parameterized Quantum Circuit
QBM	Quantum Boltzmann Machine

QC	Quantum Circuit
Q-L	Q-Learning
QML	Quantum Machine Learning
QNPG	Quantum Natural Policy Gradients
QR-DQN	Quantile Regression Deep Q-Learning/Network
QRL	Quantum Reinforcement Learning
RB	Replay Buffer
RES	Renewable Energy System
ReLU	Rectified Linear Unit
RL	Reinforcement Learning
SAC	Soft Actor-Critic
SARSA	State-Action-Reward-State-Action
SMA	Simple Moving Average
TD	Temporal Difference
TD3	Twin-Delayed Deep Deterministic Policy Gradient
TRPO	Trust Region Policy Optimization
VQA	Variational Quantum Algorithms
VQC	Variational Quantum Circuit

Bibliography

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. *Playing Atari with Deep Reinforcement Learning*. arXiv:1312.5602 [cs]. Dec. 2013.

[2] D. Heimann, H. Hohenfeld, F. Wiebe, and F. Kirchner. *Quantum Deep Reinforcement Learning for Robot Navigation Tasks*. arXiv:2202.12180 [quant-ph]. July 2022.

[3] B. Singh, R. Kumar, and V. P. Singh. "Reinforcement learning in robotic applications: a comprehensive survey." en. In: *Artificial Intelligence Review* 55.2 (Feb. 2022), pp. 945–990. ISSN: 1573-7462. DOI: 10.1007/s10462-021-09997-9.

[4] S. Gu, T. Lillicrap, I. Sutskever, and S. Levine. *Continuous Deep Q-Learning with Model-based Acceleration*. arXiv:1603.00748 [cs]. Mar. 2016.

[5] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. *Continuous control with deep reinforcement learning*. 2019. arXiv: 1509.02971 [cs.LG].

[6] J. S. Obando-Ceron and P. S. Castro. "Revisiting Rainbow: Promoting more Insightful and Inclusive Deep Reinforcement Learning Research." eng. In: *arXiv.org* (2021). ISSN: 2331-8422.

[7] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. *Attention Is All You Need*. 2023. arXiv: 1706.03762 [cs.CL].

[8] E. Ghasemian. "Stationary states of a dissipative two-qubit quantum channel and their applications for quantum machine learning." In: *Quantum Machine Intelligence* 5.1 (Feb. 2023), p. 13. ISSN: 2524-4914. DOI: 10.1007/s42484-023-00096-2.

[9] D. Maheshwari, B. Garcia-Zapirain, and D. Sierra-Sosa. "Quantum Machine Learning Applications in the Biomedical Domain: A Systematic Review." In: *IEEE Access* 10 (2022), pp. 80463–80484. DOI: 10.1109/ACCESS.2022.3195044.

[10] P. W. Shor. "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer." In: *SIAM Journal on Computing* 26.5 (Oct. 1997), pp. 1484–1509. ISSN: 1095-7111. DOI: 10.1137/s0097539795293172.

[11] L. K. Grover. *A fast quantum mechanical algorithm for database search*. 1996. arXiv: quant-ph/9605043 [quant-ph].

[12] R. S. Sutton. *Reinforcement Learning: An Introduction*. The MIT Press, 2018.

[13] M. Morales. *Grokking deep reinforcement learning*. Manning Publications, 2020.

[14] Y. Matsuo, Y. LeCun, M. Sahani, D. Precup, D. Silver, M. Sugiyama, E. Uchibe, and J. Morimoto. "Deep learning, reinforcement learning, and world models." In: *Neural Networks* 152 (2022), pp. 267–275. ISSN: 0893-6080. DOI: https://doi.org/10.1016/j.neunet.2022.03.037.

[15] OpenAI. *Spinning Up in Deep RL*. https://spinningup.openai.com/. 2020.

[16] H. Kurniawati. "Partially observable markov decision processes and robotics." In: *Annual Review of Control, Robotics, and Autonomous Systems* 5.1 (2022), pp. 253–277.

[17] Y. Wu, Y. Ye, J. Hu, P. Zhao, L. Liu, G. Strbac, and C. Kang. "Chance Constrained MDP Formulation and Bayesian Advantage Policy Optimization for Stochastic Dynamic Optimal Power Flow." In: *IEEE Transactions on Power Systems* 39.5 (2024), pp. 6788–6791. DOI: 10.1109/TPWRS.2024.3430650.

[18] F. Scharf, F. Helfenstein, and J. Jäger. "Actor vs Critic: Learning the Policy or Learning the Value." eng. In: *Reinforcement Learning Algorithms: Analysis and Applications* (), pp. 123–133. ISSN: 1860-949X.

[19] R. Bellman. "The theory of dynamic programming." In: *Bulletin of the American Mathematical Society* 60.6 (1954), pp. 503–515.

[20] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis. *Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm*. 2017. arXiv: 1712.01815 [cs.AI].

[21] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. *Asynchronous Methods for Deep Reinforcement Learning*. 2016. arXiv: 1602.01783 [cs.LG].

[22] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. *Proximal Policy Optimization Algorithms*. 2017. arXiv: 1707.06347 [cs.LG].

[23] A. Skolik, S. Jerbi, and V. Dunjko. "Quantum agents in the Gym: a variational quantum algorithm for deep Q-learning." In: *Quantum* 6 (May 2022), p. 720. ISSN: 2521-327X. DOI: 10.22331/q-2022-05-24-720.

[24] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller. "Deterministic policy gradient algorithms." In: *International conference on machine learning*. Pmlr. 2014, pp. 387–395.

[25] L.-J. Lin. *Reinforcement learning for robots using neural networks*. eng. 1992.

[26] J. Preskill. "Quantum Computing in the NISQ era and beyond." In: *Quantum* 2 (Aug. 2018), p. 79. ISSN: 2521-327X. DOI: 10.22331/q-2018-08-06-79.

[27] P. Krantz, M. Kjaergaard, F. Yan, T. P. Orlando, S. Gustavsson, and W. D. Oliver. "A quantum engineer's guide to superconducting qubits." In: *Applied Physics Reviews* 6.2 (June 2019). ISSN: 1931-9401. DOI: 10.1063/1.5089550.

[28] I. Chuang and M. Nielsen. "Introduction to quantum mechanics." eng. In: *Quantum Computation and Quantum Information*. United Kingdom: Cambridge University Press, 2010. ISBN: 9781107002173.

[29] D. Bhattacharyya and J. Guha. *Quantum Optics and Quantum Computation*. 2053-2563. IOP Publishing, 2022. ISBN: 978-0-7503-2715-2. DOI: 10.1088/978-0-7503-2715-2.

[30] H. Haverkort and L. Toma. "Quadtrees and Morton Indexing." eng. In: *Encyclopedia of Algorithms*. New York, NY: Springer New York, pp. 1637–1642. ISBN: 9781493928637.

[31] D. Aerts and M. Sassoli de Bianchi. "The extended Bloch representation of quantum mechanics: Explaining superposition, interference, and entanglement." In: *Journal of Mathematical Physics* 57.12 (Dec. 2016), p. 122110. ISSN: 0022-2488. DOI: 10.1063/1.4973356. eprint: https://pubs.aip.org/aip/jmp/article-pdf/doi/10.1063/1.4973356/16082020/122110_1_online.pdf.

[32] S. Filatov and M. Auzinsh. "Towards Two Bloch Sphere Representation of Pure Two-Qubit States and Unitaries." In: *Entropy* 26.4 (2024). ISSN: 1099-4300. DOI: 10.3390/e26040280.

[33] N. Meyer, C. Ufrecht, M. Periyasamy, D. D. Scherer, A. Plinge, and C. Mutschler. *A Survey on Quantum Reinforcement Learning*. 2024. arXiv: 2211.03464 [quant-ph].

[34] H. M. Wiseman and G. J. Milburn. *Quantum Measurement and Control*. eng. 1st ed. Vol. 9780521804424. Cambridge: Cambridge University Press, 2010. ISBN: 0521804426.

[35] S. Rajeev and M. Lahiri. "Single-qubit measurement of two-qubit entanglement in generalized Werner states." eng. In: *Physical review. A* 108.5 (2023). ISSN: 2469-9926.

[36] D. Loss and D. P. DiVincenzo. "Quantum computation with quantum dots." In: *Physical Review A* 57.1 (Jan. 1998), pp. 120–126. ISSN: 1094-1622. DOI: 10.1103/physreva.57.120.

[37] D. M. Tran, D. V. Nguyen, B. H. Le, and H. Q. Nguyen. "Experimenting quantum phenomena on NISQ computers using high level quantum programming." eng. In: *EPJ quantum technology* 9.1 (2022). ISSN: 2662-4400.

[38] E. Rieffel and W. Polak. *Quantum computing : a gentle introduction / Eleanor Rieffel and Wolfgang Polak*. eng. 1st ed. Scientific and engineering computation. Cambridge, Mass.: MIT Press, 2011. ISBN: 0-262-52667-0.

[39] S. Jerbi, L. M. Trenkwalder, H. Poulsen Nautrup, H. J. Briegel, and V. Dunjko. "Quantum Enhancements for Deep Reinforcement Learning in Large Spaces." In: *PRX Quantum* 2 (1 Feb. 2021), p. 010328. DOI: 10.1103/PRXQuantum.2.010328.

[40] *Parametrized Quantum Policies for Reinforcement Learning*. Zenodo, Jan. 2022. DOI: 10.5281/zenodo.5833370.

[41] S. Wu, S. Jin, D. Wen, D. Han, and X. Wang. *Quantum reinforcement learning in continuous action space*. 2023. arXiv: 2012.10711 [quant-ph].

[42] D. P. DiVincenzo. "The Physical Implementation of Quantum Computation." eng. In: *Fortschritte der Physik* 48.9-11 (2000), pp. 771–783. ISSN: 0015-8208.

[43] S. Sim, P. D. Johnson, and A. Aspuru-Guzik. "Expressibility and Entangling Capability of Parameterized Quantum Circuits for Hybrid Quantum-Classical Algorithms." In: *Advanced quantum technologies (Online)* 2.12 (2019), n/a. ISSN: 2511-9044.

[44] A. G. Fowler, M. Mariantoni, J. M. Martinis, and A. N. Cleland. "Surface codes: Towards practical large-scale quantum computation." In: *Physical Review A* 86.3 (Sept. 2012). ISSN: 1094-1622. DOI: 10.1103/physreva.86.032324.

[45] O. Menegasso Pires, E. Inacio Duzzioni, J. Marchi, and R. De Santiago. "Quantum Circuit Synthesis Using Projective Simulation." eng. In: *Inteligencia artificial* 24.67 (2021), pp. 90–101. ISSN: 1137-3601.

[46] V. Privman, D. Mozyrsky, and I. D. Vagner. "Quantum computing with spin qubits in semiconductor structures." In: *Computer Physics Communications* 146.3 (2002). Quantum Computing for Physical Modeling, pp. 331–338. ISSN: 0010-4655. DOI: https://doi.org/10.1016/S0010-4655(02)00424-1.

[47] W. K. Hensinger. "Quantum information: Microwave ion-trap quantum computing." eng. In: *Nature (London)* 476.7359 (2011), pp. 155–156. ISSN: 1476-4687.

[48] T. M. Graham, Y. Song, J. Scott, C. Poole, L. Phuttitarn, K. Jooya, P. Eichler, X. Jiang, A. Marra, B. Grinkemeyer, M. Kwon, M. Ebert, J. Cherek, M. T. Lichtman, M. Gillette, J. Gilbert, D. Bowman, T. Ballance, C. Campbell, E. D. Dahl, O. Crawford, N. S. Blunt, B. Rogers, T. Noel, and M. Saffman. "Multi-qubit entanglement and algorithms on a neutral-atom quantum computer." eng. In: *Nature* 604.7906 (2022), pp. 457–462. ISSN: 0028-0836.

[49] D. Dong, C. Chen, H. Li, and T.-J. Tarn. "Quantum Reinforcement Learning." In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 38.5 (2008), pp. 1207–1220. DOI: 10.1109/TSMCB.2008.925743.

[50] S. Rethinasamy, R. Agarwal, K. Sharma, and M. M. Wilde. "Estimating distinguishability measures on quantum computers." In: *Physical Review A* 108.1 (July 2023). ISSN: 2469-9934. DOI: 10.1103/physreva.108.012409.

[51] D. Sierra-Sosa, S. Pal, and M. Telahun. "Data rotation and its influence on quantum encoding." eng. In: *Quantum information processing* 22.1 (2023). ISSN: 1573-1332.

[52] J. Gonzalez-Conde, T. W. Watts, P. Rodriguez-Grasa, and M. Sanz. "Efficient quantum amplitude encoding of polynomial functions." eng. In: *Quantum* 8 (2024), p. 1297. ISSN: 2521-327X.

[53] E. Ovalle-Magallanes, D. E. Alvarado-Carrillo, J. G. Avina-Cervantes, I. Cruz-Aceves, and J. Ruiz-Pinales. "Quantum angle encoding with learnable rotation applied to quantum–classical convolutional neural networks." eng. In: *Applied soft computing* 141 (2023), p. 110307. ISSN: 1568-4946.

[54] M. A. Khan, M. N. Aman, and B. Sikdar. "Beyond Bits: A Review of Quantum Embedding Techniques for Efficient Information Processing." eng. In: *IEEE access* 12 (2024), pp. 46118–46137. ISSN: 2169-3536.

[55] M. Weigold, J. Barzen, F. Leymann, and M. Salm. "Expanding Data Encoding Patterns For Quantum Algorithms." In: *2021 IEEE 18th International Conference on Software Architecture Companion (ICSA-C)*. 2021, pp. 95–101. DOI: 10.1109/ICSA-C52384.2021.00025.

[56] M. Weigold, J. Barzen, F. Leymann, and M. Salm. "Encoding patterns for quantum algorithms." eng. In: *IET quantum communication* 2.4 (2021), pp. 141–152. ISSN: 2632-8925.

[57] L. Kunczik. *Reinforcement learning with hybrid quantum approximation in the NISQ context / Leonhard Kunczik*. eng. Wiesbaden, Germany: Springer Vieweg, 2022. ISBN: 9783658376154.

[58] A. Sequeira, L. Paulo Santos, and L. Soares Barbosa. "Trainability issues in quantum policy gradients." eng. In: *Machine learning: science and technology* 5.3 (2024), p. 35037. ISSN: 2632-2153.

[59] L. Grover. "Synthesis of quantum superpositions by quantum computation." eng. In: *Physical review letters* 85.6 (2000), pp. 1334–1337. ISSN: 0031-9007.

[60] A. Wong, T. Bäck, A. V. Kononova, and A. Plaat. "Deep multiagent reinforcement learning: challenges and directions." eng. In: *The Artificial intelligence review* 56.6 (2023), pp. 5023–5056. ISSN: 0269-2821.

[61] S. Y.-C. Chen, C.-H. H. Yang, J. Qi, P.-Y. Chen, X. Ma, and H.-S. Goan. "Variational Quantum Circuits for Deep Reinforcement Learning." In: *IEEE Access* 8 (2020), pp. 141007–141024. DOI: 10.1109/ACCESS.2020.3010470.

[62] A. Sequeira, L. P. Santos, and L. S. Barbosa. "On Quantum Natural Policy Gradients." eng. In: *IEEE transactions on quantum engineering* 5 (2024), pp. 1–11. ISSN: 2689-1808.

[63] M. H. Amin, E. Andriyash, J. Rolfe, B. Kulchytskyy, and R. Melko. "Quantum Boltzmann Machine." eng. In: *Physical review. X* 8.2 (2018), p. 021050. ISSN: 2160-3308.

[64] V. Dunjko, J. M. Taylor, and H. J. Briegel. "Framework for learning agents in quantum environments." eng. In: *arXiv (Cornell University)* (2015). ISSN: 2331-8422.

[65] A. Hamann, V. Dunjko, and S. Wölk. "Quantum-accessible reinforcement learning beyond strictly epochal environments." In: *Quantum Machine Intelligence* 3.2 (Aug. 2021). ISSN: 2524-4914. DOI: 10.1007/s42484-021-00049-7.

[66] W. J. ZENG, G. MAZZOLA, S. E. WOERNER, and N. STAMATOPOULOS. *QUANTUM ADVANTAGE USING QUANTUM CIRCUIT FOR GRADIENT ESTIMATION*. eng. 2024.

[67] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. *OpenAI Gym*. 2016. arXiv: 1606.01540 [cs.LG].

[68] D. P. Kingma and J. Ba. "Adam: A Method for Stochastic Optimization." In: *CoRR* abs/1412.6980 (2014).

[69] Q. Wei, H. Ma, C. Chen, and D. Dong. "Deep Reinforcement Learning With Quantum-Inspired Experience Replay." In: *IEEE Transactions on Cybernetics* 52.9 (2022), pp. 9326–9338. DOI: 10.1109/TCYB.2021.3053414.

[70] D. Liu, Y. Wu, Y. Kang, L. Yin, X. Ji, X. Cao, and C. Li. "Multi-agent quantum-inspired deep reinforcement learning for real-time distributed generation control of 100% renewable energy systems." In: *Engineering Applications of Artificial Intelligence* 119 (2023), p. 105787. ISSN: 0952-1976.

[71] A. Sequeira, L. P. Santos, and L. S. Barbosa. "Policy gradients using variational quantum circuits." eng. In: *Quantum Machine Intelligence/Quantum machine intelligence* 5.1 (2023). ISSN: 2524-4906.

[72] J. R. McClean, S. Boixo, V. N. Smelyanskiy, R. Babbush, and H. Neven. "Barren plateaus in quantum neural network training landscapes." In: *Nature Communications* 9.1 (Nov. 2018). ISSN: 2041-1723. DOI: 10.1038/s41467-018-07090-4.

[73] H. Hohenfeld, D. Heimann, F. Wiebe, and F. Kirchner. "Quantum Deep Reinforcement Learning for Robot Navigation Tasks." In: *IEEE Access* 12 (2024), pp. 87217–87236. ISSN: 2169-3536. DOI: 10.1109/access.2024.3417808.

[74] E. Coumans and Y. Bai. *PyBullet, a Python module for physics simulation for games, robotics and machine learning*. Version 3.0. 2016–2021.

[75] S. Huang, R. F. J. Dossa, C. Ye, J. Braga, D. Chakraborty, K. Mehta, and J. G. Araújo. "CleanRL: High-quality Single-file Implementations of Deep Reinforcement Learning Algorithms." In: *Journal of Machine Learning Research* 23.274 (2022), pp. 1–18.

[76] V. Bergholm, J. Izaac, M. Schuld, C. Gogolin, S. Ahmed, V. Ajith, M. S. Alam, G. Alonso-Linaje, B. AkashNarayanan, A. Asadi, J. M. Arrazola, U. Azad, S. Banning, C. Blank, T. R. Bromley, B. A. Cordier, J. Ceroni, A. Delgado, O. D. Matteo, A. Dusko, T. Garg, D. Guala, A. Hayes, R. Hill, A. Ijaz, T. Isacsson, D. Ittah, S. Jahangiri, P. Jain, E. Jiang, A. Khandelwal, K. Kottmann, R. A. Lang, C. Lee, T. Loke, A. Lowe, K. McKiernan, J. J. Meyer, J. A. Montañez-Barrera, R. Moyard, Z. Niu, L. J. O'Riordan, S. Oud, A. Panigrahi, C.-Y. Park, D. Polatajko, N. Quesada, C. Roberts, N. Sá, I. Schoch, B. Shi, S. Shu, S. Sim, A. Singh, I. Strandberg, J. Soni, A. Száva, S. Thabet, R. A. Vargas-Hernández, T. Vincent, N. Vitucci, M. Weber, D. Wierichs, R. Wiersema, M. Willmann, V. Wong, S. Zhang, and N. Killoran. *PennyLane: Automatic differentiation of hybrid quantum-classical computations*. 2022. arXiv: 1811.04968 [quant-ph].

[77] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. 2019. arXiv: 1912.01703 [cs.LG].

[78] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann. "Stable-Baselines3: Reliable Reinforcement Learning Implementations." In: *Journal of Machine Learning Research* 22.268 (2021), pp. 1–8.

[79] M. Franz, L. Wolf, M. Periyasamy, C. Ufrecht, D. D. Scherer, A. Plinge, C. Mutschler, and W. Mauerer. "Uncovering instabilities in variational-quantum deep Q-networks." In: *Journal of the Franklin Institute* 360.17 (Nov. 2023), pp. 13822–13844. ISSN: 0016-0032. DOI: 10.1016/j.jfranklin.2022.08.021.

[80] S. Fujimoto, H. van Hoof, and D. Meger. *Addressing Function Approximation Error in Actor-Critic Methods*. 2018. arXiv: 1802.09477 [cs.AI].