OSTBAYERISCHE
TECHNISCHE HOCHSCHULE
REGENSBURG

BACHELOR THESIS

Benjamin Zec

Conception and Integration of Accessible Hardware Demonstrators for Quantum Computing and Quantum Reinforcement Learning

January 14, 2026

Faculty:	Informatik und Mathematik
Study Programme:	Allgemeine Informatik
Supervisor:	Prof. Dr. Wolfgang Mauerer
Secondary Supervisor:	Prof. Dr. Maike Stern

Abstract

Quantum Computing (QC) offers the potential for substantial computational speed-ups by exploiting the principles of quantum mechanics. Several quantum algorithms have already been shown to outperform classical methods. However, their practical usefulness on current Noisy Intermediate-Scale Quantum (NISQ) devices is still limited due to hardware noise and the small number of available qubits. Even with these limitations, variational hybrid quantum–classical algorithms are viewed as a promising way to achieve early quantum advantages on existing hardware.

This bachelor's thesis investigates compact hardware demonstrators that make quantum concepts accessible and examines the feasibility of Quantum Machine Learning (QML) embedded systems. The first part focuses on an empirical study of Variational Quantum Deep Q-Learning (VQ-DQL) in embedded systems. For this purpose, a VQ-DQL agent was developed that controls an Anki Overdrive car on a custom-designed track. The training results show that the quantum agent can reach a performance level similar to a classical Deep Q-Network (DQN).

The second part presents a compact Quantum circuit demonstrator based on an affordable ESP32 microcontroller. The system is designed for accessibility and hands-on experimentation. It uses Radio Frequency Identification (RFID)-modules as a physical interface for selecting quantum gates and simulates the state of a five-qubit system in real time. The successful implementation and evaluation of basic quantum circuits, including the creation of entangled states such as a five-qubit Bell state, confirm the hands-on functionality and practicality of the demonstrator.

The findings of this thesis show that low-cost and accessible quantum learning tools are feasible. They also highlight future progresses, especially in improving the scalability of such demonstrators and developing more effective optimization techniques for quantum machine learning algorithms.

Kurzfassung

QC bietet das Potenzial für erhebliche Geschwindigkeitsvorteile, da es die Prinzipien der Quantenmechanik nutzt. Einige Quantenalgorithmen konnten bereits theoretisch zeigen, dass sie klassischen Methoden überlegen sind. Ihre praktische Anwendbarkeit ist auf heutigen NISQ Geräten jedoch weiterhin begrenzt, was vor allem auf Hardware-Rauschen und die geringe Anzahl verfügbarer Qubits zurückzuführen ist. Trotz dieser Einschränkungen gelten variationale hybride quanten-klassische Algorithmen als ein -versprechender Ansatz, um bereits mit bestehender Hardware einen frühen quantenbedingten Vorteil zu erzielen.

Diese Bachelorarbeit untersucht kompakte Hardware-Demonstratoren, die quantenmechanische Konzepte anschaulich vermitteln, und analysiert zugleich die Umsetzbarkeit von QML eingebetteten Systemen. Der erste Teil widmet sich einer empirischen Untersuchung von VQ-DQL in eingebetteten Systemen. Zu diesem Zweck wurde ein VQ-DQL-Agent entwickelt, der ein Anki-Overdrive-Fahrzeug auf einer Strecke steuert. Die Trainingsergebnisse zeigen, dass der Quanten-Agent ein Leistungsniveau erreichen kann, das mit einem klassischen DQN vergleichbar ist.

Im zweiten Teil wird ein kompakter Quantenschaltkreisdemonstrator vor- gestellt, der auf einem kosten- günstigen ESP32-Mikrocontroller basiert. Das System wurde speziell für leichte Zugäng- lichkeit und praxisorientiertes Lernen konzipiert. RFID-Module dienen als physische Schnittstelle zur Aus- wahl von Quantengattern, während das Gerät in Echtzeit den Zustand eines Fünf-Qubit-Systems simuliert. Die erfolgreiche Implementierung und Überprüfung grund- legender Quantenschaltkreise, darunter auch die Erzeugung verschränkter Zustände wie eines Fünf-Qubit Bellzustands, bestätigt die Funktionalität und praktische Nutzbarkeit des Demonstrators.

Die Ergebnisse dieser Arbeit zeigen, dass kostengünstige und leicht zugäng- liche Lernplattformen für QC realisierbar sind. Zudem eröffnen sie zukünftige Fortschritte, insbesondere im Hinblick auf die Skalierbarkeit solcher Demonstratoren und die Entwicklung effektiverer Optimierungsstrategien für quantenmaschinellen Lernenalgorithmen.

Contents

1 Introduction

The field of QC aims to exploit the properties of quantum mechanics in order to achieve computational speed-ups. It has indeed been shown that certain problems can theoretically be solved more efficiently using quantum algorithms than classical algorithms. Two well-known examples are Grover's algorithm [1] and Shor's algorithm [2], which respectively provide a quadratic speed-up for the exploration of unstructured search spaces and efficiently solve prime factorization and discrete logarithm problems.

While the promise of quantum computing is vast, the current generation of quantum computers, called NISQ systems, is prone to errors, such as decoherence caused by interactions with the environment, and is limited by a few available qubits [3]. These hardware imperfections limit the capabilities of current quantum devices and impose restrictions on the size of problems that can be solved on them. Nonetheless, some quantum approaches offer potential advantages over classical approaches, even in the NISQ-era, with variational hybrid quantum-classical algorithms being particularly well-suited for near-term gate-based quantum machines [4], [5]. These algorithms perform only a limited number of steps on a quantum computer, with the remaining steps executed on classical machines, making it more feasible to take advantage of QC within current hardware constraints. A promising class of hybrid methods is QML, especially Quantum Reinforcement Learning (QRL), which operates on small data sets but explores large search spaces in which quantum computing may provide advantages. [6], [7].

This exploration is significant given that despite substantial advancements in classical Reinforcement Learning (RL) over the past decade, achieving superhuman performance even on simple tasks still requires vast computational resources, making the search for quantum speed-ups in this domain a natural area of research [8]. However, a major challenge facing the entire field is the lack of intuitive, engaging, and accessible educational tools that can lower the barrier to entry for non-experts and cultivate new talent [9]. Current quantum cirriculum approaches often remain too abstract and difficult to grasp for newcomers, hindering wider understanding [9]. The primary goal of this thesis is to address this crucial gap by focusing on the development and implementation of accessible hardware demonstrators for QC and quantum-enhanced reinforcement learning.

This thesis pursues this goal by first analyzing and comparing the performance of VQ-DQL [6], [10], [11] with its classical counterpart, Deep Q-Learning (DQL), utilizing the real-world control problem of driving an Anki Overdrive car. This real-world application serves as a gamified use case to explain the core concepts of RL and its quantum enhancement in a tangible way.

Secondly, the work details the design and implementation of a compact QC hardware demonstrator using an affordable embedded microcontroller platform, such as the ESP32. This demonstrator is specifically built to provide a hands-on, real-world interface, using physical components to represent gates, allowing audiences to physically interact with and intuitively explore the core principles of a quantum circuit, thus creating a much-needed [9], enjoyable and practical entry point to this complex field.

Structure This thesis is beginning with Chapter 2, which provides the necessary background of QC. Chapter 3 introduces the fundamentals of RL, Q-Learning and its quantum analog. Chapter 4 presents the hardware components and its communication protocols used in this work. Chapter 5 describes the development of the two quantum demonstrators, including the methodology for implementing the self-learning agent and the design of the microcontroller-based quantum simulator. Chapter 6 discusses the results obtained from both demonstrators and gives an outlook on future research directions. Finally, Chapter 7 summarizes the key findings and contributions of this thesis.

2 Background on Quantum Computing

This section provides a concise introduction to the fundamental concepts of QC. It starts by introducing the basic unit of quantum information, the qubit and superpositions in Sec. 2.1 and expanding the concept of multi-qubit systems and entanglement in Sec. 2.2. Sec. 2.3 then describes quantum gates, which are the building blocks for quantum computation. Sec. 2.3.2 discusses universal gate sets, which are essential for constructing arbitrary quantum circuits. Finally, Sec. 2.4 introduces Variational Quantum Algorithm (VQA), which are particularly relevant for hybrid quantum-classical approaches.

2.1 Quantum Bits

The quantum bit, or qubit, represents the fundamental unit of quantum information. Unlike classical bits, a qubit harnesses the principle of superposition, allowing it to exist in a linear combination of the two basis states.

For single qubits, these computational basis states are symbolized by $|0\rangle$ and $|1\rangle$, forming a basis $\mathcal{B}$, allowing them to be represented [12] as

$$\mathcal{B} = \{|0\rangle, |1\rangle\} = \left\{ \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix} \right\}, \tag{2.1}$$

as a superposition or in other words a linear combination of basis states:

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle\,, \tag{2.2}$$

with its complex amplitudes $\alpha, \beta \in \mathbb{C}$.

When a measurement is performed in the standard basis $|0\rangle\,, |1\rangle$, the superposition collapses to one of the computational basis states, where the state $|0\rangle$ is obtained with probability $|\alpha|^2$, and $|1\rangle$ with probability $|\beta|^2$. The normalization condition

$$|\alpha|^2 + |\beta|^2 = 1 \tag{2.3}$$

ensures that the total measurement probability equals one [13].

The notation above used to described quantum states is known as the **Dirac notation**. Within this framework, a quantum state is represented by a vector called a ket, symbolized as $|\psi\rangle$, where ψ is simply a label for the specific state vector. The complex conjugate transpose of a ket $|\psi\rangle$ is called a bra, denoted by $\langle\psi|$.

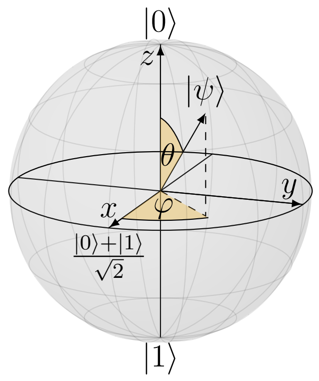


Figure 2.1: Representation of a qubit by the Bloch sphere [14].

Every pure single-qubit state can equivalently be represented as a point on the surface of the Bloch sphere, depicted in Fig. 2.1. By choosing suitable real parameters $\gamma, \theta, \varphi \in \mathbb{R}$ Eq. 2.2 can be written [13] as

$$|\psi\rangle = e^{i\gamma}\left(\cos\left(\frac{\theta}{2}\right)|0\rangle + e^{i\varphi}\sin\left(\frac{\theta}{2}\right)|1\rangle\right). \tag{2.4}$$

Here, θ corresponds to the polar angle and φ to the azimuthal angle on the Bloch sphere, while $e^{i\gamma}$ represents the global phase which has no observable physical effect and therefore be disregarded in the following.

2.2 Multiple Qubits and Entanglement

Classical bits can be combined through simple concatenation to form longer bitstrings. For example, two bits can represent four possible states, namely 00, 01, 10, 11. However, unlike classical bits, a pair of qubits can exist in a superposition of all four possible basis states.

A two qubit quantum system can therefore be represented as [13]

$$|\psi\rangle = \alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle, \tag{2.5}$$

where $\alpha_{ij} \in \mathbb{C}$ with $i, j \in \{0, 1\}$ are the probability amplitutes for the basis states and their squared amplitudes $|\alpha_{ij}|^2$ must sum to one, due to the normalization condition.

Due to the principle of superposition, an n-qubit system can encode and process information that scales as $O(2^n)$. In contrast, a classical system is limited to scaling of $O(n)$. This exponential increase in computational complexity is one of the indicators of a potential quantum advantage [12].

To construct the state space of a multi-qubit system, individual vector spaces V and W can be combined using the tensor product [13]. The resulting space has basis vectors of the form $|i\rangle \otimes |j\rangle$, where $|i\rangle$ and $|j\rangle$ denote orthonormal basis states of V and W, respectively. Any state in the combined space can then be expressed as a linear combination of these tensor product basis states.

The tensor product for vectors and matrices is implemented using the Kronecker product. For example, using the computational basis representation $|0\rangle$, the two-qubit state $|00\rangle$ is computed as:

$$|00\rangle = |0\rangle \otimes |0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \otimes \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \cdot 1 \\ 1 \cdot 0 \\ 0 \cdot 1 \\ 0 \cdot 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \tag{2.6}$$

which is a computational state of a two-qubit system.

More generally, for an n-qubit system, any quantum state consisting of 2^n probability amplitudes can be written as

$$|\psi\rangle = \sum_{i=0}^{2^n-1} \alpha_i |i\rangle, \qquad \alpha_i \in \mathbb{C}, \qquad \sum_i |\alpha_i|^2 = 1, \tag{2.7}$$

where 2^n is the dimension of the state space. Thus, the state vector of an n-qubit system formally contains 2^n amplitudes. However, only generic quantum states require an explicit specification of all amplitudes, whereas special cases, such as basis states with a single non-zero amplitude, allowing an efficient classical representation. For convenience, tensor products of multiple single-qubit states such as

$$|v_1\rangle \otimes |v_2\rangle \otimes \cdots \otimes |v_n\rangle \tag{2.8}$$

are commonly written in the compact form

$$|v_1 v_2 \cdots v_n\rangle. \tag{2.9}$$

In an ideal, noise-free quantum system, the state of the computer is fully described by a single normalized statevector, known as a pure state. Real quantum hardware, however, is subject to noise and decoherence effects [3]. These noise proccesses lead to mixed states, which are statistical ensembles of pure states, which can be described by density matrices [13]. This representation requires more memory rescources, compared to pure states, as a density matrix for an n-qubit system has dimensions $2^n \times 2^n$, making mixed-state simulations substantially more computationally expensive. Since the focus of this work lies on simulating ideal quantum circuits, all simulations in this thesis are restricted to pure states.

QC also introduces a concept called entanglement which has no classical counterpart [13]. The Bell state or EPR pair [15], [16] is a well-known example of an entangled state in a two qubit system, defined as:

$$|\Phi^+\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle), \tag{2.10}$$

which has the property that measuring one qubit instantaneously determines the state of the other. Taking the state from Eq. 2.10, the second qubit yields the same result as the first, resulting in perfect correlations between the two qubits. This phenomena is the foundation for various QC protocols and algorithms [13].

2.3 Quantum Computation

This section provides an overview of the basic principles of quantum computation, with a particular emphasis on how quantum gates are used to transform quantum states. It starts by defining a quantum gate and its quantum circuit model.

Quantum Gates An important part of QC is the manipulation of qubit states. This can be achieved by rotating the state vector in the complex vector space using transformation, such as applying matrices. To ensure that the transformation presevers the normalization condition from Eq. 2.3, the corresponding matrix U must fulfill the unitary condition

$$U^{\dagger}U = UU^{\dagger} = I, \tag{2.11}$$

where $U^{\dagger}$ is the conjugate transpose of U and I is the identity matrix [13]. Quantum operations (except measurements) are unitary operations, and therefore, are reversible. This is in contrast to classical logic gates, which are often irreversible.

Quantum Circuits Deutsch introduced 1989 the quantum circuit model [17]. This model is a description of a collection of qubits with gates acting on each qubit in a fixed sequence. Similar to classical algorithms, a quantum register prepared in an initial state serves as the input, while quantum gates act on the qubits. The output is received by measuring the qubits. There are other model representations of QC, like adiabatic quantum computing [18], but this is out of the scope of this thesis.

In the gate-based model introduced by Deutsch, qubits are represented as wires running from left to right. Quantum gates operate on the qubits by being applied to the corresponding wires.

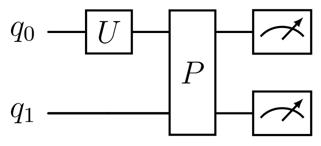


Figure 2.2: Example of a Quantum Circuit.

Fig. 2.2 illustrates an example quantum circuit in which a U gate is applied to qubit q_0, followed by a two-qubit gate P acting on qubits q_0 and q_1, after which both qubits are measured.

2.3.1 Single-Qubit Gates

Classical computing employs only one non-trivial single-bit gate, the NOT gate, which flips a bit, that is, transforming 0 to 1 and vice versa. QC, in contrast to its classical counterpart, employs multiple single-qubit gates that manipulate quantum states. The quantum analog of the classical NOT gate is the Pauli-X gate [13], defined as

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}. \tag{2.12}$$

For a general single-qubit state $\alpha |0\rangle + \beta |1\rangle$, the action of the X gate with $\alpha, \beta \in \mathbb{C}$ and $|\alpha|^2 + |\beta|^2 = 1$ is given by

$$X\alpha |0\rangle + X\beta |1\rangle = X(\alpha|0\rangle + \beta|1\rangle) = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} \beta \\ \alpha \end{bmatrix} = \beta|0\rangle + \alpha|1\rangle, \tag{2.13}$$

resulting in a negation of the amplitudes. The X gate is a part of the set of Pauli gates, which also includes the Pauli-Y and Pauli-Z gates. These widely used matrices [13] are defined as

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \quad Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}, \quad Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}. \tag{2.14}$$

Pauli matrices are not only unitary but also Hermitian [19], satisfying

$$A = A^\dagger, \tag{2.15}$$

which, described by [19], implies that their eigenvalues are real. Hermitian operators correspond to observable physical quantities in quantum mechanics, therefore they are often referred to as observables [13]. Post-measurement, the quantum state collapses to one of the eigenstates of the operator being measured. Pauli-Z is particularly important for measurements in the computational basis, as its operater has the eigenvectors $|0\rangle$ and $|1\rangle$ with eigenvalues +1 and -1 respectively, therefore making it essential in the context of hybrid quantum-classical algorithms, including QRL.

In addition to the Pauli operators, their corresponding rotational gates play a central role in quantum state manipulation. Rotational gates implement continuous transformations on the Bloch sphere, allowing qubit states to be adjusted by arbitrary angles. For a given Pauli operator $A \in \{X, Y, Z\}$, the rotation operator around axis A by an angle $\theta \in \mathbb{R}$ is defined as

$$R_A(\theta) = e^{-i\frac{\theta}{2}A}. \tag{2.16}$$

Explicitly, the three rotational gates are given by

$$R_X(\theta) = \begin{pmatrix} \cos\left(\frac{\theta}{2}\right) & -i\sin\left(\frac{\theta}{2}\right) \\ -i\sin\left(\frac{\theta}{2}\right) & \cos\left(\frac{\theta}{2}\right) \end{pmatrix}, \tag{2.17}$$

$$R_Y(\theta) = \begin{pmatrix} \cos\left(\frac{\theta}{2}\right) & -\sin\left(\frac{\theta}{2}\right) \\ \sin\left(\frac{\theta}{2}\right) & \cos\left(\frac{\theta}{2}\right) \end{pmatrix}, \tag{2.18}$$

$$R_Z(\theta) = \begin{pmatrix} e^{-i\frac{\theta}{2}} & 0 \\ 0 & e^{i\frac{\theta}{2}} \end{pmatrix}. \tag{2.19}$$

Rotational Gates are frequently used in a Variational Quantum Circuit (VQC), as they can be parameterized by optimizable parameters θ, described in more detail in Sec. 2.4.

In addition to rotation gates, the Hadamard gate H and the T gate are fundamental single-qubit gates in QC. The Hadamard gate creates superposition states from computational basis states, while the T gate applies a $\pi/4$ phase rotation, enabling finer control over qubit phases [13]. Their matrix representations are given by

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \tag{2.20}$$

$$T = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix}. \tag{2.21}$$

2.3.2 Multi-Qubit Gates and Universal Gate Sets

In QC, multi-qubit gates are fundamental for creating correlations and entanglements between qubits, which are essential resources for quantum algorithms. Among the most commonly used two-qubit gates is the CNOT (Controlled-NOT) gate, operating between two seperate qubits defined as:

$$\text{CNOT}\colon\ |c\rangle\,|t\rangle \mapsto |c\rangle\,|t \oplus c\rangle\,, \tag{2.22}$$

where $|c\rangle$ is the control qubit, $|t\rangle$ is the target qubit, and $\oplus$ denotes addition modulo 2. The CNOT gate flips the target qubit if the control qubit is in the $|1\rangle$ state, leaving it unchanged if the control qubit is $|0\rangle$ [13]. This gate is essential for generating entangled states, such as a Bellstate (see Eq. 2.10) :

$$|\Phi^+\rangle = \frac{1}{\sqrt{2}}\left(|00\rangle + |11\rangle\right) = \left(\text{CNOT}_{1,2} \otimes H\right)|00\rangle\,, \tag{2.23}$$

where $\text{CNOT}_{x,y}$ indicates that x is the control and y the target qubit.

The CNOT gate can be represented in matrix form as:

$$\text{CNOT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}. \tag{2.24}$$

Generally speaking any single-qubit gate unitary U can be used as a target gate, where the target qubit is only modified if the control qubit is in the $|1\rangle$ state [13]. The symbols for CNOT and controlled-Pauli-Z (CZ) gates, which are used in VQCs to ensure correlations, are depicted in Fig. 2.3.

(a) CNOT **(b)** CZ

Figure 2.3: Circuit symbols for the controlled-NOT ($CNOT$) and the controlled-Pauli-Z (CZ) gate. The control qubit is indicated by a black dot, whereas the target qubit is indicated by the $\oplus$ symbol for the $CNOT$ gate and by the $\times$ symbol for the CZ gate.

In QC, a *universal gate set* is a collection of gates from which any unitary operation, and thus any quantum computation, can be approximated to arbitrary accuracy. One used universal gate set consists of the single-qubit gates $\{H, X, Y, Z, T\}$ together with the two-qubit CNOT gate [20]. While Clifford gates alone can be efficiently simulated classically [21], the addition of the T gate allows for universal quantum computation and access to the full computational power of quantum systems. Alternatively, the Pauli gates can be replaced by the phase gate S, yielding the widely used Clifford+T gate set [20]. However, this formulation is beyond the scope of this thesis, as the use of Pauli gates provides a more intuitive interpretation in terms of axis rotations for the design of the demonstrator in Sec. 5.2.

Another common universal set is based on the single-qubit rotation gates $\{R_X, R_Y, R_Z\}$ together with CNOT. These gates allow continuous parameterization of qubit rotations, making them particularly suitable for VQAs, such as those used in hybrid quantum-classical approaches including QRL. Both gate sets are capable of constructing any multi-qubit unitary, demonstrating the universality of these gates for quantum computation. This concludes the building block necessary to understand variational hybrid quantum-classical algorithms, which are introduced in the next section.

2.4 Variational Hybrid Quantum-Classical Algorithms

This section introduces variational hybrid quantum-classical algorithms, which are central to the approaches for QRL discussed later in Sec. 3.4.

Variational hybrid quantum-classical algorithms represent a class of quantum algorithms that combine parameterized quantum circuits with classical optimization routines. These algorithms are widely used to approximate target functions [22] in tasks such as QRL, which is described in Sec. 3.4.

A VQA aims to find a set of optimal parameters $\vec{\theta}$ that minimizes a cost function C depending on the parameters:

$$\vec{\theta}^* = \arg\min_{\vec{\theta}} C(\vec{\theta}), \tag{2.25}$$

where $\vec{\theta}^*$ represent the optimal parameters for the cost function. A VQA employs a parameterized quantum circuit, often referred to as a VQC . Such circuits contain gates with adjustable parameters, typically realized through single-qubit rotations around the pauli axes. During training, an input state is prepared and processed by the VQC, after which measurements are performed on the output state. The circuit parameters are then updated using classical optimization methods, such as the Adam optimizer [23], to minimize a predefined cost function. This hybrid quantum-classical loop is iterated until the VQC approximates the target function to the desired quality.

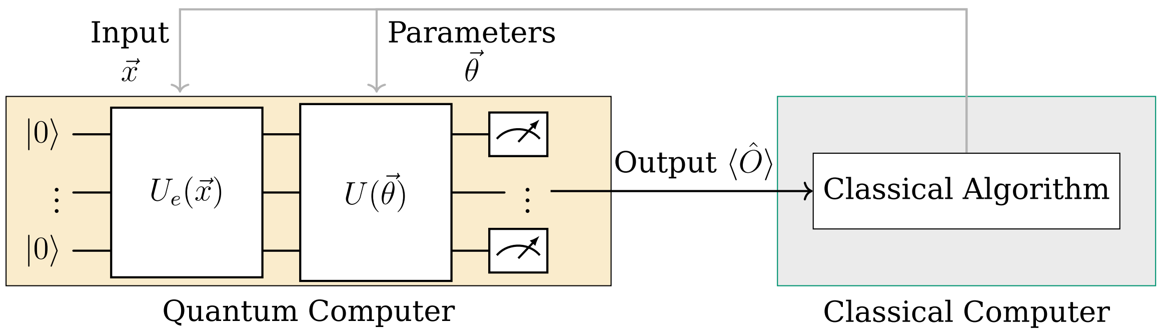


Figure 2.4: Structure of a quantum-classical algorithm using a VQC [14]

An VQC consists of three parts, seen in Fig. 2.4. The first part prepares classical input data $\vec{x}$ to represent a quantum state. The input is encoded into a initial state $|\psi_{\text{in}}(\vec{x})\rangle$, which is by convention assumed to be $|0\rangle^{\otimes n}$ [12], using an encoding unitary $U_e(\vec{x})$:

$$|\psi_{\text{in}}(\vec{x})\rangle = U_e(\vec{x})\,|0\rangle^{\otimes n}. \tag{2.26}$$

After encoding, the state is processed by a parameterized unitary $U(\vec{\theta})$, referred to as a variational layer, representing the trainable part of the circuit:

$$|\psi_{\text{out}}(\vec{x},\vec{\theta})\rangle = U(\vec{\theta})\,|\psi_{\text{in}}(\vec{x})\rangle. \tag{2.27}$$

Finally, measurements of an observable $\hat{O}$ are performed on the output state to obtain classical information. The expectation value of $\hat{O}$, later discussed in Sec. 2.4.2, is used to evaluate the cost function.

The classical optimizer then updates the parameters $\vec{\theta}$ iteratively to minimize $C(\vec{\theta})$. The classical part feeds input data $\vec{x}$ to the encoding component and parameters $\vec{\theta}$ to the variational layer [6]. The following three subsections provide a more detailed explanation of quantum encodings used to transform classical data to quantum states, data extraction of quantum states and the calculation of gradients used for the cost function evaluation.

2.4.1 Quantum Encodings

In order to process classical information on a quantum computer, the data must first be transformed into a quantum representation. This transformation is realised through encoding unitaries, which map classical input vectors $\vec{x}$ to quantum states $|\psi_{\text{in}}(\vec{x})\rangle = U_e(\vec{x})\,|0\rangle^{\otimes n}$. Several encoding strategies have been proposed in the literature, each offering a different trade-off between circuit complexity, qubit efficiency, and representational density. In this work, two widely used methods [24], [25] are listed in the following:

Amplitude Encoding enables compact representation of classical data by mapping real-valued features, such as integers or floating-point numbers, onto the amplitudes of a quantum state. Using n qubits, it is possible to encode up to 2^n classical values simultaneously, while ensuring that the amplitudes satisfy the normalisation condition Eq. 2.7. This encoding therefore offers the highest information density among common approaches. However, preparing a general quantum state with arbitrary amplitudes via the corresponding unitary $U_e^{(a)}$ requires an exponential number of operations to encode 2^n data values, making it infeasible for current NISQ-devices [25], [26]. Consequently, amplitude encoding is typically limited to small data dimensions or theoretical studies. Hence, we do not consider it in our study.

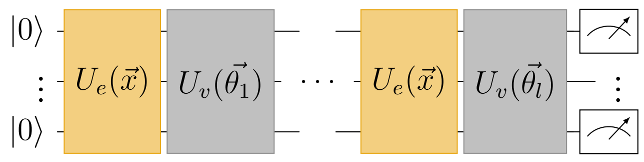
Angle Encoding in contrast, maps classical values to the rotation angles of parameterised quantum gates. The corresponding unitary $U_e^{(\angle)}(\vec{x})$ applies one [10], [25] or more [27], [28] Pauli-rotation gates to each qubit:

$$U_e^{(\angle)}(\vec{x}) = \prod_{i=1}^{n} R_{\sigma_i}(x_i), \tag{2.28}$$

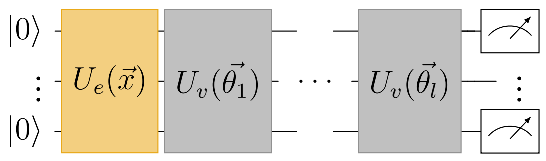
where $R_{\sigma_i}(x_i) = e^{-ix_i\sigma_i/2}$ and σ_i denotes a Pauli operator (X, Y, or Z). Since rotation gates are periodic, input values must be scaled to a finite interval, typically $[0, 2\pi]$ or smaller. Various schemes have been proposed to perform this scaling. Skolik et al. [29] present Continous (C) encoding, which computes rotation angles as the arctan of the respective input value [6]. Angle encoding requires only one qubit per feature, leading to shallow circuits and low resource requirements, which makes it particularly well suited for near-term quantum hardware and applications such as QRL. For these reasons, angle encoding is employed in this work.

The standard encoding strategy in a VQC follows a structure analogous to a classical Neural Network (NN): the input is embedded once into the circuit using an encoding unitary, after which the variational layers are applied until measurement, depicted in Fig. 2.5b.

However, data re-uploading [30] has been proposed as a method to increase the expressivity of VQCs, that is their ability to represent groups of functions [10]. In gate-based quantum circuits, there is no theoretical limitation on how often input features can be reintroduced. This allows the encoding unitary to be inserted multiple times throughout the circuit, as illustrated in Fig. 2.5a. Reintroducing the input information at several points in the circuit enables the model to capture more complex relationships[31], thereby enhancing its representational power.



(a) VQC architecture with data re-uploading.



(b) Standard VQC architechture.

Figure 2.5: VQC architectures with and without data re-uploading for an encoding unitary $U_e(\vec{x})$ representing the classical data $\vec{x}$ and a variational unitary $U_v(\vec{\theta}_i)$, parameterised by $\vec{\theta}_i$ with $i \in [1, l]$ and l variational layers.

2.4.2 Quantum Data Extraction

To retrieve classical information from a quantum state $|\psi\rangle$, one typically evaluates the expectation value $\langle\hat{O}\rangle$ [32] of a hermitian observable $\hat{O}$, which is defined as

$$\langle\hat{O}\rangle = \frac{\langle\psi|\hat{O}|\psi\rangle}{\langle\psi|\psi\rangle}. \tag{2.29}$$

Because all quantum states in this work are assumed to be normalized, $\langle\psi|\psi\rangle$ is equal to one and can therefore be ommitted. As quantum measurements are probabilistic by nature, the probability of each measured value is bound to the amplitudes of the corresponding basis state. Therefore the expectation value of an observable cannot be properly obtained from a single circuit measurement but must be estimated by averaging measurement outcomes over multiple repetitions. As prior stated, due to simplicity and hardware efficiency, Pauli operations are frequently chosen as measurement observables in VQAs [32].

2.4.3 Quantum Gradient Estimation

To solve the optimization problem, defined in Eq. 2.25, which aims to find optimal pramеters $\vec{\theta}$, the parameters can be trained using gradient-based approaches. In this context, a VQC is considered where the parameterized unitary $U(\vec{\theta})$, which depends on m real gate parameters $\vec{\theta}$, transforms the initial state $|\psi\rangle$, followed by a measurement of the observable $\hat{O}$. The output of such a parameterized quantum circuit can be written as the expectation value of an observable $\hat{O}$ with respect to the state prepared by the circuit. Formally, this defines a real-valued function [31] $f : \mathbb{R}^m \to \mathbb{R}$:

$$f(\vec{\theta}) := \langle \hat{O} \rangle = \langle \psi \,|\, U^\dagger(\vec{\theta}) \hat{O} U(\vec{\theta}) \,|\, \psi \rangle, \tag{2.30}$$

where $U(\vec{\theta})$ denotes a parameterized unitary transformation acting on an input state $|\psi\rangle$. To minimize a cost function $C(\vec{\theta})$ based on Eq. 2.30, gradient-based optimisation methods can be applied. However, classical finite-difference approximations (or altogether gradient-free methods) are unfeasible in near-term quantum devices [33].

Consider a parameterised quantum gate of the form

$$R_\sigma(\mu) = e^{-i\mu\sigma/2}, \tag{2.31}$$

where σ is a Pauli operator and $\mu \in \vec{\theta}$ is a single trainable parameter. If each generator σ has only two distinct eigenvalues, then the partial derivative of $f(\vec{\theta})$ with respect to $\mu \in \vec{\theta}$ can be evaluated [22] analytically via the parameter-shift rule [4], [33]:

$$\frac{\partial f(\vec{\theta})}{\partial \mu} = \frac{1}{2} \left[f(\vec{\theta}_{\mu,+\frac{\pi}{2}}) - f(\vec{\theta}_{\mu,-\frac{\pi}{2}}) \right], \tag{2.32}$$

where $\vec{\theta}_{\mu,s}$ denotes that parameter μ is parameter shifted by s, whereas the parameter $(\mu + s) \in \vec{\theta}_{\mu,s}$. The parameter-shift rule enables the analytical computation of gradients [31]. For a general cost function $C(\vec{\theta})$, which depends on the expectation values produced by the VQC, the gradient of $C(\vec{\theta})$ can therefore be obtained by applying the chain rule [6], [22].

As shown by Pérez et al.[30], a VQC can act as a universal function approximator, similar to a classical NN [34]. This implies that, given a sufficient number of parameters, a VQC can approximate any continuous function. This property makes VQCs highly suitable for optimization and machine learning tasks, where function approximation is a central component. Additionally, VQCs offer flexible control over circuit depth and the number of qubits, which makes them particularly adaptable for use on NISQ-devices [5].

3 Background on Reinforcement Learning

In this section we will introduce the basic concepts of RL and its mathematical foundation, the Markov Decision Process (MDP). We will also discuss various RL algorithms, including Q-Learning and Temporal-Difference Learning (TDL). Finally, we will explore DQL and its application in complex environments. RL is a subfield of machine learning which falls into the category of interaction-based learning. The goal of RL is to learn a optimized policy with regard to a reward received by its corresponding problem.

3.1 Markov Decision Process

The mathematical framework undermining RL is the MDP, which provides a formal notation to model the sequential interaction between an agent and its environment over discrete time steps t. An MDP is comprehensively defined by the tuple $\langle S, A, P, R, \gamma \rangle$, capturing the entire dynamics and objective of the problem, which can be seen in Fig. 3.1.

In each time step t, the state of the environment is summarized by the state $S_t \in S$, where S denotes the set of all possible states. Based on this configuration, the agent selects an action $A_t \in A$, where A is the set of all possible actions. This selection is governed by a policy π, defined as the probability P of taking action A_t in state S_t, $\pi(s, a) = P[A_t = a | S_t = s]$.

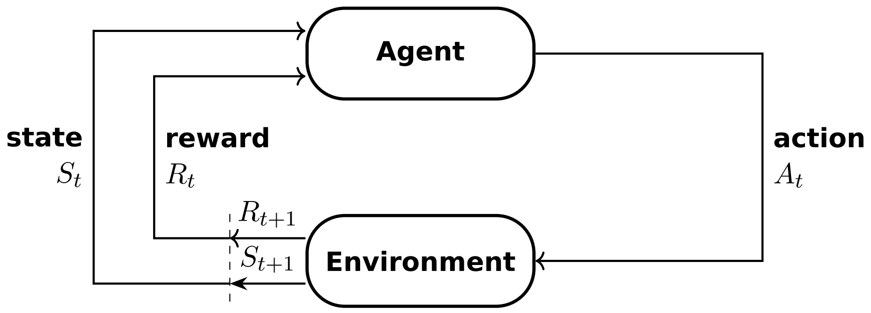


Figure 3.1: Transition of an agent in a MDP interacting with its environment over discrete time steps [35].

Taking action A_t in state S_t causes the environment to transition to a new state $S_{t+1} \in S$. This transition is determined by the state transition probability $P(S_{t+1}|S_t, A_t)$, which specifies the likelihood of reaching state S_{t+1} from state S_t after taking action A_t. This probability fulfills the Markov property [36], meaning the next state depends solely on the current state S_t and action A_t, independent of the past history of the process.

3.1.1 Rewards and Return

The agent's objective is to find a policy that maximizes the expected cumulative reward it receives from the environment, known as the return G_t. This is calculated as the discounted sum of future rewards until a terminal state S_T is reached [35]:

$$G_t = \sum_{t'=t}^{T} \gamma^{t'} R_{t'}. \tag{3.1}$$

Here, the discount factor $\gamma \in [0, 1]$ determines the present value of future rewards, effectively balancing immediate and long-term gains. The period from the initial state S_0 to the terminal state S_T is referred to as an episode.

A key challenge in RL is that the MDP's underlying dynamics $P(S_{t+1}, R_{t+1}|S_t, A_t)$ are typically unknown to the agent. This necessitates that the agent learns an optimal behavior π, typically through trial-and-error interaction with the environment.

3.1.2 Learning Policies

The efficacy of a policy π is assessed using the action-value function $q_\pi(s, a)$:

$$q_\pi(s, a) = \mathbb{E}\left[G_t \mid S_t = s, A_t = a\right], \tag{3.2}$$

where G_t represents the cumulative discounted reward, often referred as the Q-value [35]. This function measures the expected return starting from state $s \in S$, executing action $a \in A$, and subsequently following policy π.

The primary objective in a RL task is to identify a policy that maximizes the accumulated reward over time. We can rank policies by comparing their corresponding action-value function. Hence, policy π is considered superior to π' if

$$q_\pi(s, a) > q_{\pi'}(s, a) \quad \forall (s, a). \tag{3.3}$$

The optimal action-value function, denoted as q^*, represents the maximum expected return achievable from any state-action pair:

$$q^*(s, a) = \max_\pi q_\pi(s, a), \quad \forall s \in S, \forall a \in A. \tag{3.4}$$

This function is maximized by the optimal policy π^*. Once q^* is determined, π^* can be directly derived by selecting the action that yields the highest q^*-value in each state [11], [35]:

$$\pi^*(a, s) = \arg\max_\pi q_\pi(s, a). \tag{3.5}$$

In practice, RL methods aim to approximate or learn this optimal policy.

3.2 Q-Learning

Since the underlying dynamics of the MDP are typically unknown to the agent, RL agents must utilize methods to approximate the optimal policy π^*. Q-Learning [37] is a core RL technique that estimates the optimal action-value function defined in Eq. 3.4. Q-Learning effectively combines principles from Dynamic Programming (DP) and TDL, which we introduce in this section.

3.2.1 Dynamic Programming and the Bellman Optimality Equation

In DP, value functions are employed to guide the search for optimal policies [11]. The Bellman Optimality Equation is central to this process, relating the optimal value of the current state-action pair to the maximum optimal value of the subsequent state:

$$q^*(s,a) = \mathbb{E}\left[R_{t+1} + \gamma \max_{a'} q^*(S_{t+1}, a') \mid S_t = s, A_t = a\right]. \tag{3.6}$$

This equation establishes a self-consistency condition for q^* that is independent of any specific policy. In iterative DP methods, a sequence of Q-value functions $(q_0, q_1, \dots)$ is generated, starting from an arbitrary initialization q_0. The subsequent approximation q_{k+1} is derived from the current estimate q_k with $k \in \mathbb{N}$ using the recursive update rule:

$$q_{k+1}(s,a) = \mathbb{E}\left[R_{t+1} + \gamma \max_{a'} q_k(S_{t+1}, a') \mid S_t = s, A_t = a\right]. \tag{3.7}$$

The policy π_k derived from q_k is guaranteed to converge toward the optimal policy π^* as $k \to \infty$. While DP methods mathematically guarantee convergence to the optimal policy, the necessity of an accurate environment model makes them impractical for most real-world scenarios [35]. Q-Learning overcomes this limitation by operating in a model-free manner, relying instead on collected experience [35].

3.2.2 Temporal-Difference Learning

TDL enables the construction of improved estimates of the action-value function by bootstrapping from previous approximations. Q-Learning, a prominent TDL control method, was originally introduced as a tabular learning algorithm [37]. In this formulation, a Q-value is stored for every encountered state-action pair within a lookup table, randomly initialized.

As the agent explores the environment, it updates the Q-value for each visited state-action pair according to the following update rule [35]:

$$q(S_t, A_t) \leftarrow (1-\alpha)\, q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_{a'} q(S_{t+1}, a')\right], \tag{3.8}$$

where $\alpha \in [0,1]$ denotes the learning rate (or step size). This update moves the old estimate $q(S_t, A_t)$ toward the TDL target, given by the term in brackets.

A crucial property of Q-Learning is that it is an off-policy algorithm, where it uses the maximum estimated Q-value of the next state, $\max_{a'} q(S_{t+1}, a')$, rather than the Q-value corresponding to the action actually taken. This allows Q-Learning to directly approximate the optimal action-value function q^*. In the tabular case, it has been proven that Q-Learning converges to the optimal Q-values $q^*(s,a)$ provided that all state-action pairs are visited infinitely often [38].

3.3 Deep Q-Learning

Classical Q-Learning's reliance on a lookup table to store $q(s,a)$ values are infeasible for environments with large or continuous state spaces [35], therefore in DQL [39] a NN is used to approximate the action-value function, which is discussed in this section.

3.3.1 Deep Q-Learning Architecture and Loss

The fundamental idea of DQL is to learn the optimal Q-values from the Bellman Optimality Equation Eq. 3.6 using a multi-layered NN that, for any given state s, outputs a vector of Q-values $q(s, a; \vec{\theta})$, where $\vec{\theta}$ are the parameters of the network. If the state space has n dimensions and the action space has m actions, the neural network acts as a function mapping $\mathbb{R}^n$ to $\mathbb{R}^m$.

The neural network is trained by minimizing a loss function $L(\vec{\theta})$ [39], typically the mean-squared temporal-difference error, which quantifies the difference between the current and target Q-value estimates:

$$L_i(\vec{\theta}_i) = \mathbb{E}_{(S_t, A_t, R_{t+1}, S_{t+1})}\left[\left(Y_i - q(S_t, A_t; \vec{\theta}_i)\right)^2\right], \tag{3.9}$$

where each training step $i \in \mathbb{N}$ samples transitions $\langle S_t, A_t, R_{t+1}, S_{t+1}\rangle$ from experience. The target value Y_i (the TDL target) is defined [40] as

$$Y_i = R_{t+1} + \gamma \max_{a'} q(S_{t+1}, a'; \vec{\theta}_i), \tag{3.10}$$

which defines a one-step estimate of this optimal value. In this way, the network is trained to minimize the discrepancy between its current Q-value estimate and the target derived from the Bellman equation. The remainder of this section discusses the optimization of the neural network parameters $\vec{\theta}$ to minimize the loss function defined in Eq. 3.9 and addresses the stability challenges associated with training DQN agents.

3.3.2 Gradient Update

The parameters, also referred to as weights, of the deep neural network are optimized using gradient-based techniques. A standard optimization method is Stochastic Gradient Descent [41], but the Adam optimizer [42] is frequently employed due to its adaptive learning rate mechanism and efficient convergence properties.

For each parameter, the Adam optimizer updates the weight based on the learning rate α and a ratio of the bias-corrected moving averages of the gradient (first moment, $\hat{m}$) and the squared gradient (second moment, $\hat{v}$):

$$\vec{\theta_{t+1}} = \vec{\theta_t} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \varepsilon}, \tag{3.11}$$

where $\alpha \in [0, 1]$ denotes the learning rate (step size) and ε is a small constant added for numerical stability. The gradients $g_t = \frac{\partial L(\vec{\theta_t})}{\partial \vec{\theta_t}}$ are obtained via backpropagation, which applies the chain rule to compute derivatives with respect to the loss function L.

However, training q^* with a high-capacity function approximator such as a deep neural network can lead to convergence issues. To mitigate these instabilities, DQL employs two key mechanisms: the use of a target network and experience replay, which are described in the following subsections.

3.3.3 Experience Replay

Experience replay [43] enables the agent to store its past interactions with the environment. Each transition is represented as $e_t = \langle S_t, A_t, R_{t+1}, S_{t+1} \rangle$ and is stored in a dedicated memory structure known as the replay buffer $\mathcal{D}_t = (e_1, ..., e_t)$. This buffer continuously collects experiences over multiple episodes. During training, the network parameters are updated by randomly sampling small batches of transitions $e \sim \mathcal{U}(\mathcal{D})$ from this memory [39] based on Eq. 3.9

The main advantage of experience replay is that it allows the agent to reuse past experiences, which improves data efficiency [39]. Moreover, by sampling transitions randomly, it breaks the strong correlations between consecutive samples that typically occur during sequential data collection. In DQL, the utilizes an $\epsilon - greedy$ behavior policy. This behavior policy introduces stochasticity by selecting a random action with probability ϵ. The gradual decay of ϵ throughout the training process is implemented to ensure the agent initially maintains wide exploration of the environment. As the training progresses and ϵ approaches zero, the exploration diminishes, guaranteeing that the behavior policy ultimately converges towards the optimal target policy.

3.3.4 Target Network

To improve the stability of DQL, a separate target network is introduced to compute the target Q-values Y_i [35]. This network, parameterized by $\vec{\theta}^-$, is updated less frequently than the online network, which is trained continuously.

The target network parameters are periodically synchronized with those of the online network every $C \in \mathbb{N}$ training steps:

$$\vec{\theta}^{-} \leftarrow \vec{\theta} \quad \text{every } C \text{ steps.} \tag{3.12}$$

Using a delayed copy of the parameters $\vec{\theta}^{-}$ decouples the target generation from the most recent updates, reducing feedback loops that can cause instability. This delay significantly mitigates divergence that may occur when a single network is used for both prediction and target estimation [39]. The target value Y_i is, instead of using Eq. 3.10, computed using the target network as

$$Y_i = R_{t+1} + \gamma \max_{a'} q(S_{t+1}, a'; \vec{\theta}_i^{-}). \tag{3.13}$$

This section concludes the discussion of the core concepts of classical DQL, which builds the foundation for its quantum analog. The following section will focus on the modifications and adaptations specific to the quantum version of DQL.

3.4 Variational Deep Q-Learning

VQ-DQL represents a recent class of hybrid quantum-classical RL methods. In this approach, the traditional deep neural network within the DQL framework is replaced by a VQC [10], [14], [28], [29]. As VQCs, outlined in Sec. 2.4, can act as universal function approximators, they are well-suited for machine learning and especially for the DQN algorithm.

The parameters $\vec{\theta}$ of the VQC are optimized using classical optimization techniques, such as the Adam optimizer, in combination with the parameter-shift rule, described in Sec. 2.4.3, for gradient estimation. This section provides an overview of how the VQC handles input embedding and output extraction, which differs from the classical neural network approach.

Quantum Embedding A crucial step in VQ-DQL is quantum embedding, where the classical state information $s \in S$ of the MDP is encoded into a quantum state $|\psi(s)\rangle$. The choice of embedding method significantly affects the performance of the VQC. Suitable encoding strategies were discussed in Sec. 2.4.1 and can therefore be employed. Therefore Sec. 5.1.1 describes the input components of the MDP state in our environment and Sec. 5.1.2 explores the encoding method.

Q-Value Extraction For a given MDP state, the VQC outputs Q-values for all $|A|$ actions simultaneously by measuring the expectation values of Pauli-Z observables on the output qubits. These values lie within $[-1, 1]$ and may require scaling to obtain valid action values. Replacing the classical neural network with a VQC allows leveraging QC in RL. While VQ-DQL models are not proven superior to classical DQL, several studies indicate potential advantages [10], [11], [28], [44] and even sample efficiency [14].

4 Background on Hardware

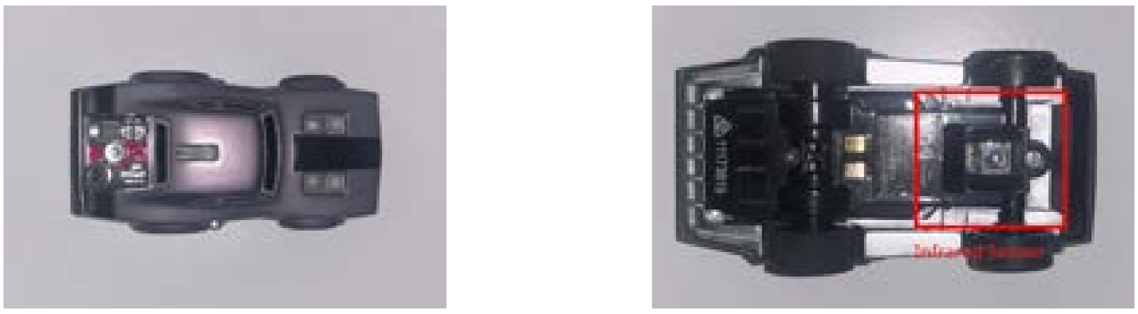
In this section, we introduce the hardware architecture and communication protocols underlying the QC demonstrators. We outline the roles of each hardware component and describe how they interact through different communication protocols. This provides the foundation for understanding how the quantum demonstrators are developed and integrated.

4.1 Reinforcement Learning Demonstrator

To serve as a physical demonstrator, a gamified and intuitively accessible use-case was chosen. This is to ensure that the core concepts of RL and its quantum enhancement can be easily adapted and explained to a major audience. The chosen hardware demonstrator for the RL use-case is detailed in the following subsections.

4.1.1 Anki-Overdrive

The Anki Overdrive demonstrator is a sensor-based racing system that combines physical toy cars with digital control. Each vehicle operates on a modular track and is controlled via a smartphone application, enabling races against either human players or pre-programmed artificial intelligent opponents.



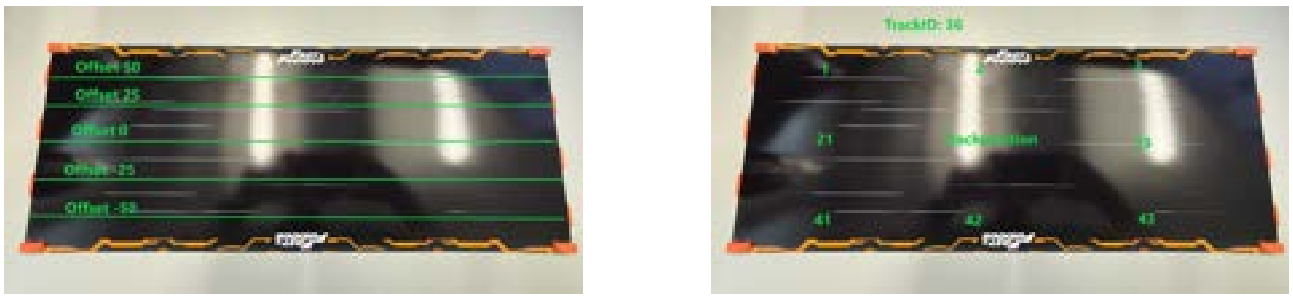
(a) Top view of the used Anki Overdrive car.

(b) Bottom view of the used Anki Overdrive car. On the side, the infrared sensor for track piece detection is visible.

Figure 4.1: Top and bottom view of the Anki Overdrive car used as RL demonstrator. It is 8,5 cm long and 4 cm wide.

The system effectively merges physical hardware with algorithmic decision-making, providing a perfect use-case environment for RL. It consists primarily of two components:

modular track pieces that can be assembled into various configurations, and small autonomous vehicles, depicted in Fig. 4.1a, equipped with embedded electronics for sensing and control. A key feature of the Anki Overdrive platform is its integrated localization system. Each track piece contains a unique infrared code that is detected by a downward-facing sensor installed on the car, depicted in Fig. 4.1b. This detection mechanism allows the car to continuously determine its position on the track by identifying the current track piece ID and its relative offset from the center of the lane, depicted in Fig. 4.2a. The combination of track pieces, which gives us information, and the controlable vehicle, gives the reason to use Anki Overdrive as the demonstrator in comparison to other toy car systems like the Donkey Car Racing Set build on a Raspberry Pi [45]. The lateral offset, represented as a floating-point value, quantifies how far the vehicle deviates from the track's centerline and is essential for navigation, control, and reinforcement learning feedback. Other essential information, depicted in Fig. 4.2b are the



(a) Distinction of the track in Anki Overdrive with discrete offset positions from -50 to +50 with ±25 steps.

(b) Cumulative track position and unique track ID in Anki Overdrive.

Figure 4.2: Anki Overdrive location features which builds the state space for the RL agent.

horizontal offset on each piece, represented as a float-point value, its current speed, between [0,1,2,...,1500], and a boolean value quantifying if the car is driving clockwise or counterclockwise.

4.1.2 Communication Distribution

The communication between the Anki Overdrive vehicles and the smartphone application operates via Bluetooth. Each car is identified by a unique MAC address, which enables direct access through Bluetooth Low Energy (BLE). By using a predefined BLE library [46], it becomes possible to both receive telemetry data from the vehicle and send control commands such as acceleration or lane adjustment in real time. This wireless interface plays a crucial role in the training of our RL agent. In order to obtain precise information, that is the car's lateral offset or transitions between track pieces, we prioritized certain telemetry signals over others. As a result, direct speed measurements had to be omitted, since not all data could be transmitted simultaneously within a single BLE packet. To compensate for the missing speed information, we introduce an indirect speed estimation method. Instead of using the raw velocity readings, the system calculates the time required for the vehicle to transition from one track piece to the next. This transition time proved to be a stable and reliable indicator of the vehicle's effective speed while maintaining correct information of other critical features used for training the RL model.

4.2 Quantum Simulator Hardware

Building and maintaining a real quantum computer is an exceptionally complex task that requires advanced expertise and highly specialized equipment. While various online quantum circuit simulators exist, they remain abstract and difficult to grasp for newcomers to the field. The goal is therefore to create a tangible, real-world demonstrator that allows audiences to physically interact with and intuitively explore the principles of a quantum circuit. To realize this simulator, a range of hardware components are employed, as described in the following section. The entire system is composed of four primary hardware components: the ESP32, an Adafruit LED-matrix for displaying the state distribution, a PCF8575 IO expander for general-purpose input/output (GPIO) extension, and an MFRC522 RFID-module with its tag, which serves as a gate within the circuit.

4.2.1 Hardware Building Blocks

ESP32 At the core of the setup is the ESP32 Development Board (DevKitC V4) [47], which integrates a CP2102 USB-to-UART bridge and is fully compatible with the Arduino development environment, supporting C++ programming. This makes it well-suited for lightweight and flexible embedded implementations. The ESP32 features a dual-core processor, built-in Wi-Fi and Bluetooth connectivity, and a wide range of GPIO pins, as shown in Fig. 4.3.

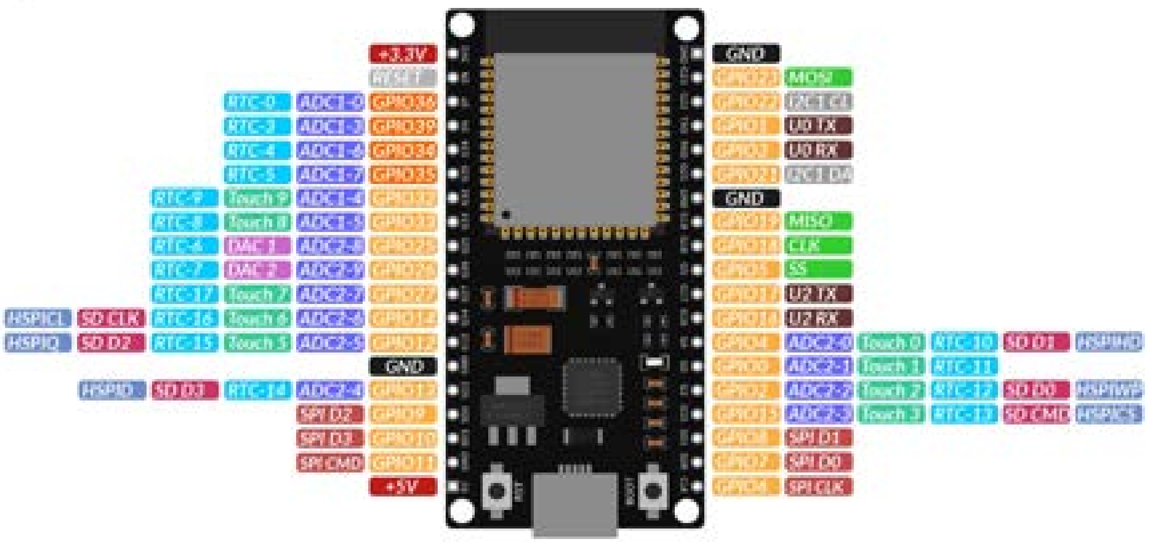

+3.3V
RESET
RTC-0 ADC1-0 GPIO36
RTC-3 ADC1-3 GPIO39
RTC-4 ADC1-6 GPIO34
RTC-5 ADC1-7 GPIO35
RTC-9 Touch 9 ADC1-4 GPIO32
RTC-8 Touch 8 ADC1-5 GPIO33
RTC-6 DAC 1 ADC2-8 GPIO25
RTC-7 DAC 2 ADC2-9 GPIO26
RTC-17 Touch 7 ADC2-7 GPIO27
HSPICL SD CLK RTC-16 Touch 6 ADC2-6 GPIO14
HSPIQ SD D2 RTC-15 Touch 5 ADC2-5 GPIO12
GND
HSPID SD D3 RTC-14 ADC2-4 GPIO13
SPI D2 GPIO9
SPI D3 GPIO10
SPI CMD GPIO11
+5V
GND
GPIO23 MOSI
GPIO22 I2C SCL
GPIO1 U0 TX
GPIO3 U0 RX
GPIO21 I2C SDA
GND
GPIO19 MISO
GPIO18 CLK
GPIO5 SS
GPIO17 U2 TX
GPIO16 U2 RX
GPIO4 ADC2-0 Touch 0 RTC-10 SD D1 HSPIHD
GPIO0 ADC2-1 Touch 1 RTC-11
GPIO2 ADC2-2 Touch 2 RTC-12 SD D0 HSPIWP
GPIO15 ADC2-3 Touch 3 RTC-13 SD CMD HSPICS
GPIO8 SPI D1
GPIO7 SPI D0
GPIO6 SPI CLK

Figure 4.3: ESP32 pin-layout [47].

These characteristics make the ESP32 particularly suitable for embedded sensing applications that require real-time data acquisition, which are important in our use-case considering our other components. In addition, its low cost compared to alternatives such as the Raspberry Pi [48], combined with its processing power and energy efficiency, makes the ESP32 an excellent choice as the central computation and control unit of the system.

IO-Expander The PCF8575 I/O expander [49] is used to increase the number of available GPIO pins on the ESP32, which are otherwise limited. This 16-bit I/O expander communicates with the ESP32, enabling control over additional output lines required to assert the CS lines for multiple Serial Peripheral Interface (SPI) slave devices. Each output pin can be individually configured as input or output, and the device supports fast switching for real-time control signals. Its Inter-Integrated Circuit (I^2C) address is configurable via address pins, allowing multiple expanders on the same bus depicted in Fig. 4.4. The PCF8575 is powered at 3.3V allowing to use the same energy supply and logic levels as the ESP32. This component was chosen because it is inexpensive, easy to use, and highly flexible, making it ideal for expanding the control capabilities of the ESP32. Alternatively, a shift register (i.e 74HC595) could have been used to extend the GPIO pins, but the PCF8575 was preferred, because its interface is used in other components of the setup (i.e LED-matrix) as well.

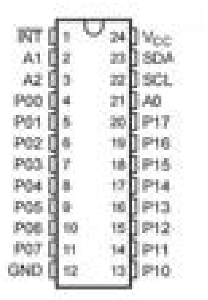
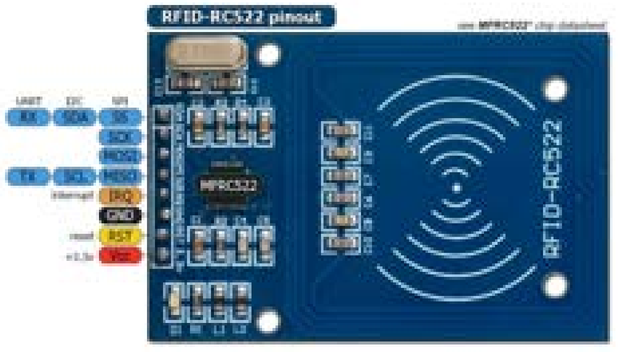
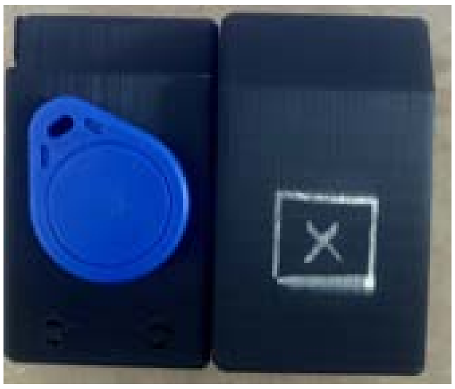


Figure 4.4: PCF8575 pin-layout [49].



(a) MFRC522 RFID-module pin-layout [50].

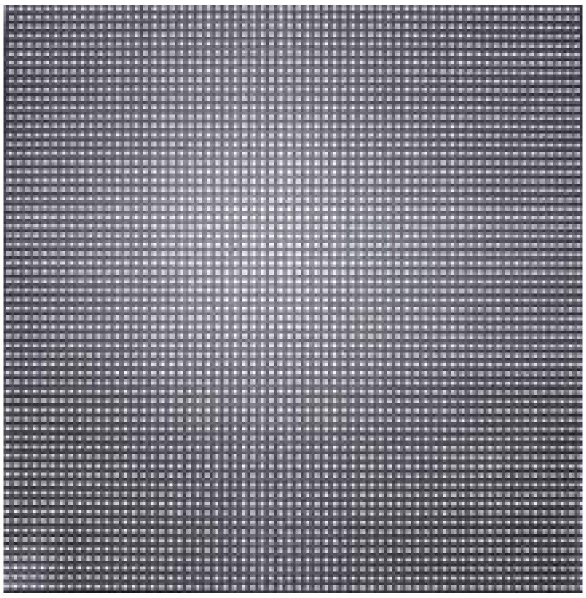


(b) Front and backside of the 3D-printed Gates.

Figure 4.5: RFID-module with its 3D-printed gate to simulate a quantum gate.

RFID-module The MFRC522 RFID-module [50] functions as a gate within the quantum circuit demonstrator. Connected to the ESP32 via SPI, it detects the RFID-modules and identifies the tags placed on it and defines their position in the quantum circuit. Each RFID-module contains a small memory chip where information can be stored and read by the ESP32, enabling the simulation of quantum gates by encoding custom data on the tags. The MFRC522 operates at 3.3V, uses a standard SPI interface with Master Output Slave Input (MOSI), Master Input Slave Output (MISO), Serial Clock (SCLK), Chip Select (CS), and Reset (RST) lines, seen in Fig. 4.5a, and supports high-speed reading for real-time operation. The MFRC522 also allows, as only the chip is necessary to read data, to customly design the chips on 3D-printed setup, depicted in Fig. 4.5b, representing a gate, making it for the audience more approachable and understandable. Its low-cost, availability and simple interface with fast reading times of the information of the chip, makes it the perfect candidate for our setup.

LED-Matrix The Adafruit 64x64 RGB LED-matrix [51] serves as a visual interface for the quantum circuit simulation. Each pixel can be individually addressed, enabling representation of probabilities or gate operations in real time. This LED-matrix, depicted in


(a) Front side of the used 64x64 LED-matrix.

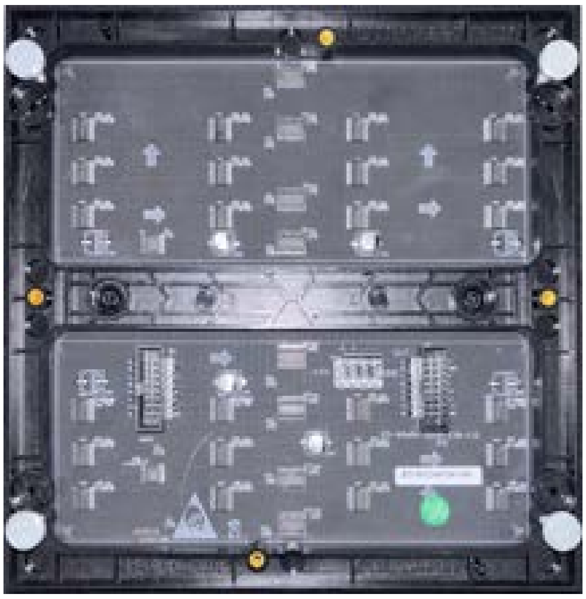

(b) Backside of the LED-matrix with its power supply and pin-layout.

Figure 4.6: Front and backside of the LED-matrix.

Fig. 4.6, was chosen because it is particularly suitable for visualizing the distribution of the quantum circuit with columns, and it is much more cost-effective than other display options (i.e. OLED-screen), making it an valid choice for an interactive demonstrator.

4.2.2 Communication Protocols between Hardwarecomponents

In embedded systems, data exchange between different Hardware components is achieved through communication protocols. In the development setup, the ESP32 manages communication with the other peripheral devices using both Serial Peripheral Interface and Inter-Integrated Circuit protocols.

I^2C Communication The I^2C interface is a widely used protocol that enables multiple devices to communicate over four wires: the data line (SDA) and the clock line (SCL), common ground potential Ground (GND) and a power line Voltage Common Collector (VCC), as summarized in Tab. 4.1. Originally developed by NXP Semiconductors, I^2C is designed for short-distance communication between controllers and peripheral devices [52]. The protocol operates in half-duplex mode, meaning that data can only be transmitted or received at a given time [52], [53]. A single data line is shared for bidirectional communication, with direction arbitration implemented via an open-drain configuration and external pull-up resistors, ensuring that no two devices drive the bus simultaneously [52]. Each slave device is assigned a unique 7-bit address, while the eighth bit of the address byte specifies whether the operation is a read or a write. Following each transmitted byte, the receiving device must send an acknowledgment or a non-acknowledgment signals the end of the communication. Data transfers are synchronized to the clock, with data remaining stable while SCL is high, except during start and stop conditions [52].

Table 4.1: I^2C connection overview between ESP32 and PCF8575 IO-Expander Modules.

I^2C Signal	ESP32 Pin Function
SDA (Serial Data)	Carries data between devices
SCL (Serial Clock)	Synchronizes SDA data transfer between devices
GND (Ground)	Common electrical reference ground for all connected devices
VCC (Power Line)	Supplies operating voltage to the PCF8575 modules

In our setup, I^2C is used for communication between the ESP32 and both the LED-matrix and the IO expanders. Since no data needs to be read back from these devices, I^2C is an ideal choice as commands can be sent efficiently, for example to update the display or set the appropriate control lines on the IO-expanders. The protocol's simplicity and reliability make it particularly suited for these low-speed control tasks, ensuring stable operation without unnecessary complexity.

SPI Communication The SPI protocol is a high-speed, full-duplex communication standard that enables efficient data transfer between a master device and one or more slave devices [54], [55]. In our system, the ESP32 acts as the master, communicating with the RFID-modules via SPI. The connection employs the following signal lines:

Table 4.2: SPI connection overview between ESP32 and MFRC522 RFID-modules.

SPI Signal	ESP32 Pin Function
MISO	Data from slave to ESP32
MOSI	Data from ESP32 to slave
SCK	SPI clock signal
CS	Controlled via I/O Expander
RST	Reset signal
VCC	Power line
GND	Ground line

In SPI systems with multiple slaves, each slave must have its own CS line. The master determines which slave it is communicating with by asserting the corresponding CS line. Slaves that are not currently selected must ignore any data on the MOSI line and refrain from driving the MISO line to prevent data collisions. The SCLK is provided by the master, and slave devices should never manipulate this line [54], [55].

SPI is a full-duplex protocol, meaning that data can be transmitted and received simultaneously. Typically, one edge of the clock signal is used to transmit data from the master to the slave, while the opposite edge is used for the master to receive data from the slave. This allows every byte sent by the master to correspond to a byte received from the slave, enabling data rates of up to 80 Mbps in full-duplex mode [55]. Therefore, SPI is particularly well-suited for the quantum circuit simulator application due to its high data transfer rates and low latency , which are essential for rapid RFID-module and tag detection and response [55]. With clock speeds in the megahertz range, SPI allows the RFID-module to operate in real time without significant communication overhead. Additionally, its simple implementation makes SPI an ideal choice for communication between the ESP32 as the master and the RFID-modules as slave devices.

This concludes the overview of the hardware demonstrator setup, highlighting the key components and communication protocols that enable its operation. The next section will focus on the conception, implementation, and integration of the hardware demonstrators, starting with the VQ-DQL use-case and followed by the quantum circuit simulator.

5 Development of Accessible Quantum Computing Demonstrators

5.1 Implementation of the Quantum Reinforcement Learning Demonstrator

This section details the implementation of the QRL demonstrator. We begin by introducing the Anki Overdrive environment used as the test-bed for our experiments. We then describe the methodology and reproduction details of our VQ-DQL agent, including the VQC architecture, training pipeline, and hyperparameter settings. Finally, we outline the classical NN baseline used for comparison.

5.1.1 The Anki Overdrive Environment

The Anki Overdrive environment serves as an excellent test-bed for reinforcement learning research, as it combines real physical dynamics with continuous sensor feedback. Its continuous state space, together with a discrete action space, makes the task non-trivial and allows for a meaningful evaluation of RL algorithms.

The objective of the environment is to achieve the fastest possible lap time on a given track by passing each track segment in the minimum amount of time. Since the vehicle operates on a physical track with real sensor measurements, the agent must anticipate upcoming situations rather than reacting only to instantaneous observations. The complete state representation, depicted in Tab. 5.1, therefore captures both the current driving context and the local track structure.

Table 5.1: State definition and boundaries of the observation space.

State Variable	**Min**	**Max**	**Description**
`Transitiontime`	0	3	Positive Time since last track transition (s)
`Offset`	-70	70	Lateral offset from track center
`Current Piece`	17	40	ID of the current track piece
`Next Piece`	17	40	ID of the upcoming track piece
`Piecetype`	0	1	Type of the track piece (e.g., straight/curve)
$\texttt{Action}_{\texttt{t-1}}$	0	6	First action component (e.g., accelerate)
$\texttt{Action}_{\texttt{t-2}}$	0	6	Second action component

As discussed in Sec. 4.1.1, we were unable to use the vehicle's velocity as a state variable due to hardware limitations. Instead, we use the time since the last track transition as a proxy for speed. This value simultaneously serves as the reward of the environment due to its high update rate and its strong correlation with lap time. Fast reward feedback is crucial, as the agent must learn an optimal policy within the limited training horizon.

We include the current piece and next piece to encode the identity of the track segments that the car is currently on and about to enter. Different segments correspond to different curvature properties, which strongly influence both the feasible speed range and the optimal lateral offset. Without this information, the agent would struggle to accelerate efficiently on straight segments, perform optimal lane changes before entering curves, and maintain stable control of the vehicle. The piece type variable further abstracts this information by indicating whether a segment is a curve or a straight, enabling the policy to generalize across different track layouts.

Due to sensor noise and delayed physical responses, the immediate effect of an action A_t is not always reflected in the subsequent observation. Therefore, we additionally include the variables previous action A_{t-1} and action history A_{t-2}. These variables help stabilize the control behavior by informing the agent about its recent decisions and allowing it to anticipate delayed effects of its actions.

The action space is discrete and consists of seven possible actions that combine acceleration, deceleration, and lateral control. To preserve a discrete representation, the track width is divided into five offset levels, as illustrated in Fig. 4.2a, to ensure that the car remains within the track boundaries during lane changes and does not crash. The complete action space is summarized in Tab. 5.2.

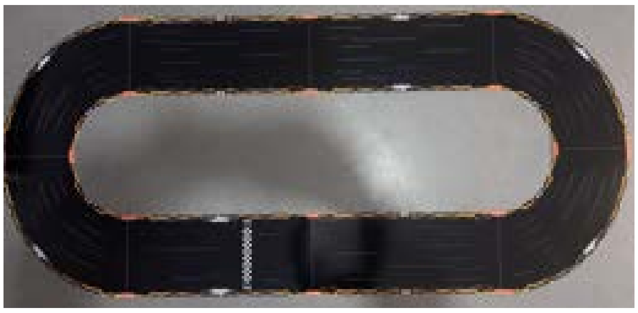
Table 5.2: Discrete action space used in the Anki Overdrive environment. The first two actions control the vehicle's velocity, while the remaining five actions manage lane switching.

Action	**Effect**	**Description**
$\texttt{Action}_0$	Velocity = +100	Accelerate
$\texttt{Action}_1$	Velocity = -100	Decelerate
$\texttt{Action}_2$	Offset = -50	Switch to outer right lane
$\texttt{Action}_3$	Offset = -25	Switch to inner right lane
$\texttt{Action}_4$	Offset = 0	Switch to the middle of the lane
$\texttt{Action}_5$	Offset = +25	Switch to inner left lane
$\texttt{Action}_6$	Offset = +50	Switch to outer left lane

An episode begins with the car positioned at the starting line of the track and immediately transitioning to the next track piece by crossing the finish line depicted in Fig. 5.1a. The agent then receives its initial observation, which depends on the car's starting position and the information of the newly entered track segment.

An episode terminates when the car leaves the track, disconnects, completes a full lap, or reaches a predefined maximum number of training steps. At each step, the agent receives a reward defined as the negative transition time between consecutive track pieces, thereby incentivizing the minimization of lap time. Consequently, the cumulative episode reward is directly correlated with the achieved lap time, with lower cumulative values indicating better performance.

(a) Crossing of the start line signals the beginning of the training pipeline, as this track piece is divided into two sections separated by the finish line.



(b) Track layout used for the training of the agents, which can be divided into 9 pieces.

Figure 5.1: Track layout and starting position of the Anki Overdrive environment used for training the agent.

Several constraints had to be introduced to ensure stable operation of the environment. First, we capped the maximum speed of the vehicle at 800 to prevent the car from losing traction, sliding off the track, or abruptly changing direction at excessively high speeds. Moreover, beyond a certain speed the car moved too quickly for the embedded firmware to capture sensor signals reliably, due to its limited sampling rate. For our experiments, we chose a circular track layout, shown in Fig. 5.1b, as it represents the simplest and most verifiable track configuration. The track is composed of nine pieces, consisting of five straight and four curved segments. This minimal layout was necessary due to the limited battery capacity of the Anki Overdrive vehicle, which restricts each online-learning episode to approximately 1200 steps. More complex or bigger track configurations would require substantially longer training episodes and were therefore not feasible to learn an optimal policy within the available time frame. Furthermore, a low battery level did not affect the vehicle's speed or sensing quality, meaning that the learning process remained stable until the battery was fully depleted.

5.1.2 Methodology

To ensure a stable and reproducible learning process consistent with classical RL, the implementation utilizes a single-file implementation in PyTorch [56]. The VQCs within the VQ-DQL framework are implemented using Pennylane [57], as its integration with PyTorch is supported and provides more efficient parameter updates compared to alternatives like Qiskit [58]. This efficiency is crucial because the system has only a small window to update its VQC due to the sparse transition times of the environment. The algorithm used in the training process is outlined in Alg. 1.

To describe our methodology in detail, let us first define employed convention: By training steps we refer to the number of iteration by gradient descent the model has undergone. By sampling steps we refer to the number of interaction the agent has with the environment. For our VQ-DQL we initialize a Replay Memory Buffer $\mathcal{D}$ to store experience tuples $\langle S, A, R, S_{t+1} \rangle$ up to 7000 transitions (see Alg. 1, line 1), which is sufficient for the limited number of sampling steps gathered during training. The Q-Network is initialized as a VQC with random parameters $\vec{\theta}$ (see Alg. 1, line 6). The architecture of the VQC is discussed in the next section in detail. The Target Network is updated every 32 steps (see Alg. 1, line 3, 18-20). The output undergoes post-processing with a ReLU activation function to ensure non-negative Q-Values estimates and promote faster learning times [59], [60]. The Adam optimizer is used for parameter updates (see Alg. 1, line 17), employing a learning rate of 0.01.

Algorithm 1 Variational Quantum Deep Q-Network

```
 1: Initialize replay memory buffer $\mathcal{D}$
 2: Initialize training start $M_{train}$ = 250
 3: Initialize target network update parameter $C$
 4: Initialize global step counter $M$ = 0
 5: Initialize $\epsilon$ = 1.0
 6: Initialize Quantum Circuit for action-value function with random parameters $\vec{\theta}$
 7: Initialize the Anki environment, get $S_1$ and encode it into a quantum state
 8: for $M$ < 1100 or Done do
 9:     With probability $\epsilon$ select random action $A_t$
10:     Otherwise select $A_t = \arg\max_a q(S_t, a; \vec{\theta})$ from the output of the VQC
11:     Execute action $A_t$ on the car, observe reward $R_t$ and next state $S_{t+1}$
12:     Store transition $(S_t, A_t, R_t, S_{t+1})$ in $\mathcal{D}$
13:     if $M \bmod M_{train}$ = 0 then
14:         Sample random minibatch of transitions from $\mathcal{D}$
15:         $Y_i = \begin{cases} R_t & \text{if episode terminates at } S_{t+1} \\ R_{t+1} + \gamma \max_{a'} q(S_{t+1}, a'; \vec{\theta}) & \text{otherwise} \end{cases}$
16:         Compute loss: $L = \left(Y_i - q(S_t, a'; \vec{\theta})\right)^2$
17:         Update $\vec{\theta}$ using Adam optimizer on $L$
18:         if $M \bmod C = 0$ then
19:             Update the target network $\vec{\theta}^-$ with $\vec{\theta}$
20:         end if
21:     end if
22:     Decay $\epsilon$ linearly
23: end for
```

We start the training loop with 250 sampling steps (see Alg. 1, line 2) by initializing ϵ to 1.0 to ensure full exploration of the environment (see Alg. 1, line 5). At each sampling step, the agent selects an action A_t using an ϵ-greedy policy based on the Q-values estimated by the VQC (see Alg. 1, line 9). With probability ϵ, a random action is selected to encourage exploration. We decay ϵ linearly to 0.00 over the next 200 sampling steps to gradually shift towards exploitation (see Alg. 1, line 22). The selected action (see Alg. 1, line 9-10) is therefore with a probability of $1 - \epsilon$ the action with the highest estimated Q-value:

$$A_t = \arg\max_a q(S_t, a; \vec{\theta}). \tag{5.1}$$

After executing the action A_t in the environment, the agent observes the reward R_t and the next state S_{t+1} (see Alg. 1, line 11). The transition $\langle S_t, A_t, R_t, S_{t+1} \rangle$ is stored in the replay buffer $\mathcal{D}$ (see Alg. 1, line 12). Once enough experience is gathered and the training starts (see Alg. 1, line 13), the agent samples a random mini-batch of transitions from $\mathcal{D}$ to perform a training step (see Alg. 1, line 14). For each transition in the mini-batch, we compute the target Q-value Y_i (see Alg. 1, line 15) using Eq. 3.13:

$$Y_i = \begin{cases} R_t & \text{if episode terminates at } S_{t+1} \\ R_{t+1} + \gamma \max_{a'} q(S_{t+1}, a'; \vec{\theta}) & \text{otherwise} \end{cases} \tag{5.2}$$

The loss (see Alg. 1, line 16) is then computed as the mean squared error using Eq. 3.9:

$$L = \frac{1}{N} \sum_i \left[\left(Y_i - q(S_t, A_t; \vec{\theta}_i) \right)^2 \right], \tag{5.3}$$

where N is the mini-batch size. The Loss is minimized by updating the parameters θ of the VQC. We update the target network parameters $\vec{\theta}^-$ with the current network parameters $\vec{\theta}$ every 32 sampling steps (see Alg. 1, line 18-20) to ensure stable Q-value estimates. This process is repeated for a total of 1100 sampling steps or until the early stopping criterion is met. The additional classical hyperparameter used to tune the algorithm are depicted in Tab. 5.3. As an automated hyperparameter search is infeasible due to hardware constrains, we manually tested different combinations and selected the best performing ones for both agents. If the average lap time over five consecutive evaluation episodes drops to its best lap time, we consider the training successful and terminate early to prevent catastrophic forgetting [61] (see Alg. 1, line 8). This early stopping criterion is crucial given the short training time in the training setup.

Table 5.3: Comparison of the hyperparameters used for training the classical DQN agent and the quantum-enhanced agent in the Anki Overdrive environment. The hyperparameter were optimized through a manual grid search process.

Hyperparameter	Classic DQN	Quantum Agent	Description
num_steps M	1100	1100	#sampling steps
train_after M_{train}	250	250	#sampling steps before first training step
learning_rate	0.001	0.01	optimizer learning rate
buffer_size	8000	7000	size of the replay buffer
gamma γ	0.9999	0.9999	discount factor γ
optimiser	Adam[42]	Adam[42]	lossfunction optimiser
loss	L_2	L_2	lossfunction
target_network_frequency C	32	32	frequency of sampling steps between target network updates
batch_size	128	128	batch size for gradient descent
ϵ_{start}	1.0	1.0	initial exploration rate ϵ
ϵ_{end}	0.00	0.00	final exploration rate ϵ
$\epsilon_{fraction}$	0.15	0.11	fraction of training steps for epsilon decay

Variational Quantum Circuit Architecture

The VQC architecture used for Q-value estimation consists of 7 qubits, representing the observation space. For the input encoding, we initially employed a continuous angle encoding scheme in which all features were prepared by applying a Hadamard gate followed by a σ_y rotation as described in Sec. 2.4. The Hadamard gate provides an unbiased initial state, and the subsequent rotation implements the state preparation. Empirically, scaling the inputs with $\arctan(x_i)^2$ stabilized training and improved convergence toward a sufficient policy. Prior work suggests input or output scaling with trainable weights to mitigate barren plateaus [11], [29], [62]. In our setting, however, these methods did not improve performance and sometimes prevented the agent from converging to a sufficient policy. Each variational layer consists of one parameterized single-qubit rotation around the z-axis σ_z. For l layers and n qubits, this results in $1nl$ trainable parameters. The entanglement pattern follows a circular topology introduced by [31] and is defined as $CNOT[i, (i + l) \ mod \ n]$ where l is the index of the layer and n the number of qubits. Similar to the classical hyperparameters, an exhaustive grid search of the architectures was not feasible. Therefore, the presented architechture was selected as it yielded the best results.

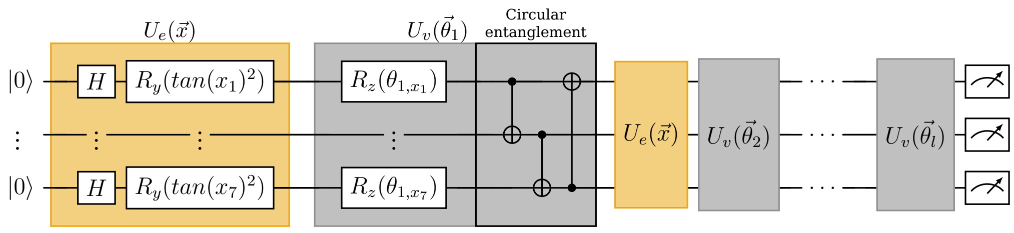


Figure 5.2: VQC architecture used in our experiments using data-reuploading and circular entanglement with 7 qubits and l layers.

To ensure sufficient expressivity, we employ data re-uploading [30], which is described in Sec. 2.4. We tested architectures with 3 to 10 repeated layers, using the 4-layer configuration as the baseline, since it achieved the best performance in the Anki Overdrive environment.

The results are shown in Fig. 5.3. Besides the baseline model, only two additional architectures reached the sufficient policy within the available training time, although both required longer training periods. All architectures exhibited phases where the training became stuck in local minima. Nevertheless, even the architectures that ultimately failed to reach the sufficient policy discovered promising intermediate solutions at some point during training. It is therefore plausible that, given longer training durations, all models would eventually converge. This behaviour is consistent with the findings of Mnih et al. [39], who showed that Q-learning can become unstable when combined with non-linear function approximators such as VQCs.

Although the 4-layer architecture discovered the optimal policy in the shortest time, deeper architectures exhibited more stable learning dynamics, which is in agreement with the literature on the relationship between layer depth and expressivity [63]. Architectures with too few or too many layers likely suffered either from insufficient expressivity or from barren plateaus, which limited their performance within the short training horizon imposed by the hardware. Since all architectures eventually produced reasonable intermediate policies, it is reasonable to assume that extended training, beyond what the limited battery capacity of the Anki Overdrive car allowed, would lead to convergence across all tested configurations.

Demonstration of the Quantum Reinforcement Learning Agent

To highlight the educational value of our demonstrator, we trained the best performing agent using the four-layer VQC architecture. The training was conducted directly on the physical Anki Overdrive environment and filmed to track the entire training progress.

Fig. 5.4 illustrates the learning progression of the quantum agent. In Fig. 5.4a, the agent's initial behavior after five episodes is shown, reflecting early exploration and learning attempts. Fig. 5.4b depicts the agent after the training has converged, demonstrating its ability to find the sufficient policy to find the best route to reach the fastest laptime. To further illustrate the agent's decision-making, the trajectories of the car over the last three episodes are visualized using a pink line, showing how the agent refines its actions over time and gradually improves its performance. This intuitive demonstration

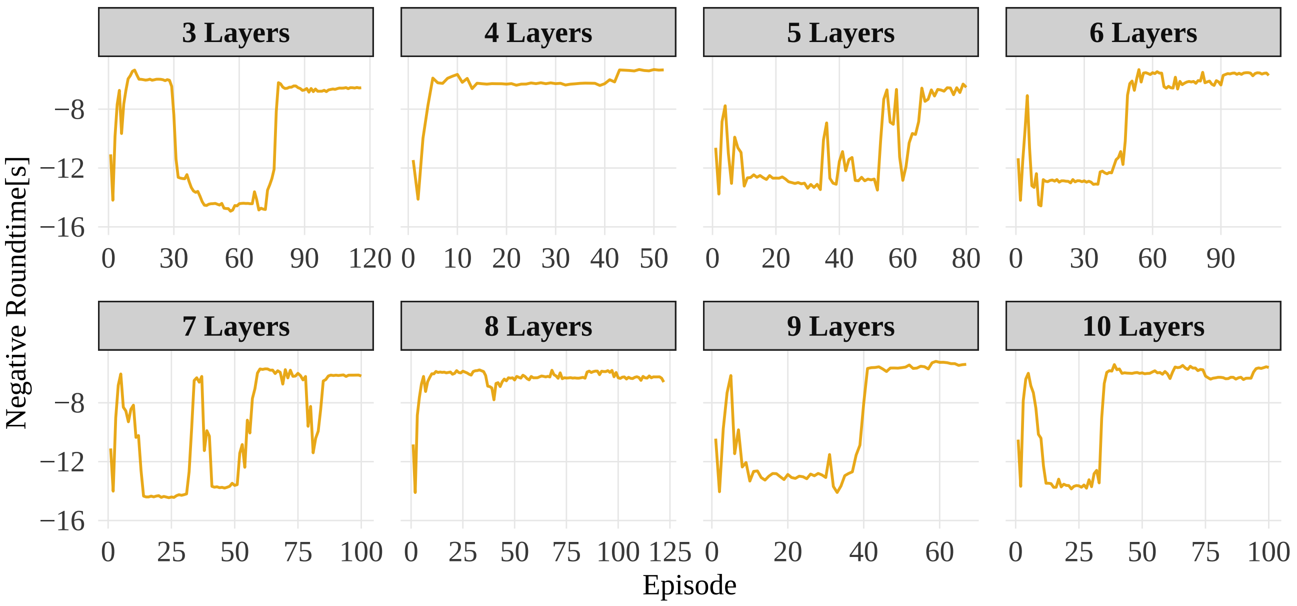
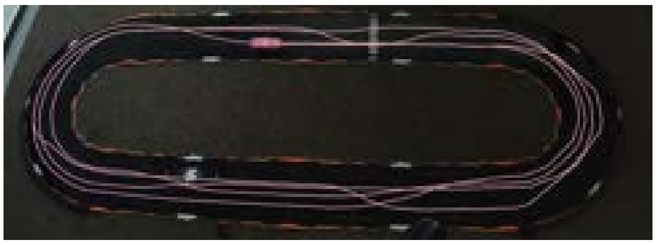
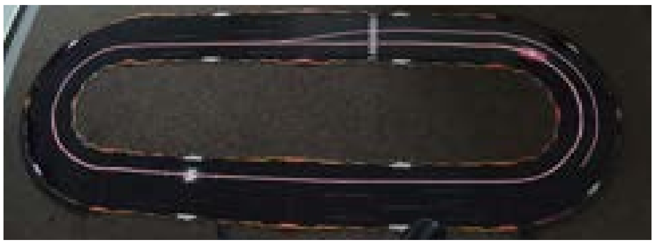


Figure 5.3: Training performance of VQ-DQL agents with different VQC layers depths in the Anki Overdrive environment. The x-axis shows the number of episodes, while the y-axis represents the episode value (negative lap time). The nearer the episode value is to zero, the better the lap time achieved by the agent. Layer depths 4, 9 and 10 reached the training stopping criteria within the available training time, while the other architecture found reasonable, but not optimal, policies. The layer depth 4 architecture converged the fastest among all tested configurations. Still, deeper architectures exhibited more stable learning dynamics, in comparison to shallower ones.



(a) The quantum agent after 5 episodes of training.



(b) The quantum agent after convergence of the training process.

Figure 5.4: Demonstration of the Quantum Reinforcement Learning Agent in the Anki Overdrive environment.Fig. 5.4a shows the agent's behavior after the first five training episodes, while Fig. 5.4b illustrates the agent after convergence. To highlight the action selection process, the trajectories over the last three episodes are visualized with a pink line.

not only highlights the agent's learning dynamics but also makes the concepts of QRL more accessible and engaging to a broader audience.

Comparison to Classical Neural Network

Franz et. al[6] suggests that quantum-enhanced agents can outperform classical baselines, particularly in low-data regimes. To provide a meaningful comparison, we used the same single file framework and for the Q-Network we used a fully connected NN as the function approximator. The architecture is shown in Fig. 5.5 and consists of three hidden layers with 128, 32, and 16 neurons. All layers use ReLU activation to produce Q-values for each action.

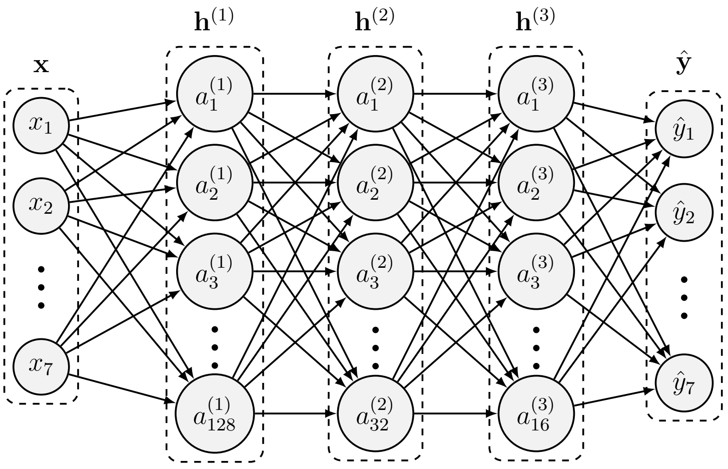


Figure 5.5: Neural Network Architecture where the input layer $\mathbf{x}$ has 7 neurons corresponding to the observation space and the output layer $\hat{\mathbf{y}}$ has 7 neurons corresponding to the action space. We used in our Neural Network three hidden layers $\mathbf{h}^{(i)}$, with $i \in [1, 2, 3]$ denoting the hidden layer and a_n^i denotes the number of neurons in the i-th hidden layer.

The relatively compact architecture was chosen to match the limited complexity of the environment and the resource constraints of the embedded hardware. With only seven input features and a moderate state–action structure, larger networks would introduce computational overhead without improving performance. Smaller networks are also known to generalize more reliably in low-dimensional environments [64], which helps avoid over-parameterization and instability during training. The hyperparameters used for the classical agent correspond to the best-performing configuration listed in Tab. 5.3. In this setting, the classical NN contains a total of 5859 trainable parameters. In contrast, the quantum agent employs 28 trainable parameters in its variational circuit.

Both agents are able to learn a reasonable driving policy within the available training time. The classical agent converges approximately ten episodes earlier, which can be attributed to its substantially higher number of trainable parameters, allowing it to approximate the Q-function more accurately given the limited data. The quantum agent, however, exhibits a notably more stable learning curve with reduced variance across episodes. The comparison of both agents during training is presented in Fig. 5.6. This supports the findings reported in [6], indicating that quantum models may provide smoother and more stable function approximations in reinforcement learning tasks. These results should be interpreted with caution, since we were unable to exhaustively tune hyperparameters for both agents due to hardware constraints. Furthermore, the difference in the number of trainable parameters is substantial. We were not able to identify a classical NN with a similar parameter count comparable to the VQC that could still learn a reasonable policy within the same training time. This indicates that the quantum

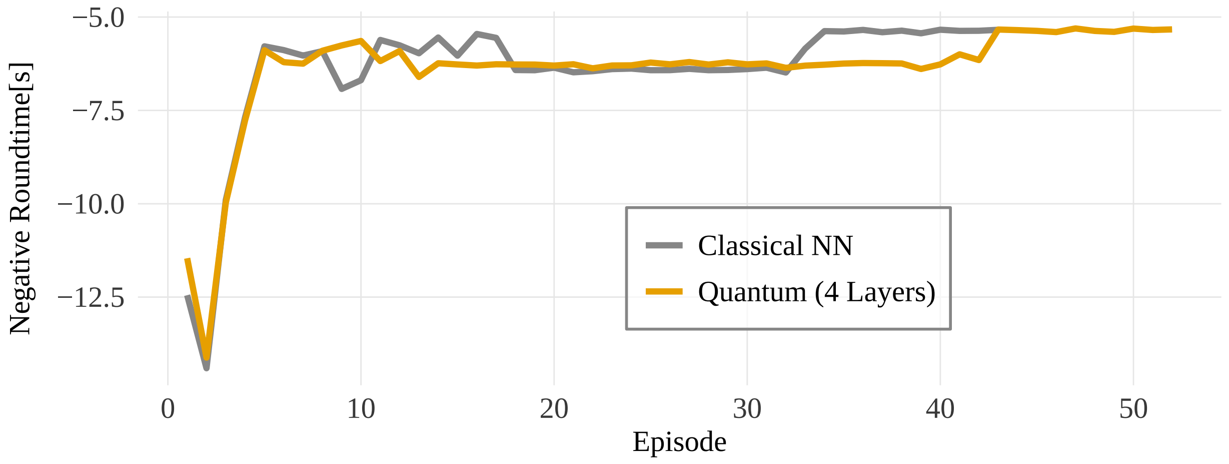


Figure 5.6: Training progress of the classical and quantum agents. The yellow line represents the VQ-DQL agent, while the gray line corresponds to the classical NN agent. The classical agent converges slightly faster, but the quantum agent exhibits a more stable learning curve. The first few episodes are almost identical as both agent are seeded the same way for reproducibility.

model achieves competitive performance with significantly fewer parameters, which is particularly promising for resource-constrained reinforcement learning applications.

5.2 Development of the Quantum Simulator

To address, similar to the QRL demonstrator, the abstract nature of QC for newcomers, the goal is to create a physical demonstrator of a quantum circuit. This section details the design and implementation of a lightweight quantum simulator tailored for embedded hardware platforms. The simulator is built to efficiently execute a small quantum circuit and visualize the resulting quantum state in a performative manner on resource-constrained devices for a broad audience.

5.2.1 Simulation Library Design

The quantum library is designed to simulate a 5-qubit system on embedded hardware. The state of each qubit is represented as a complex two-dimensional vector, and the full 5-qubit system is stored as a 32-dimensional complex vector corresponding to the computational basis states. The initial state of the system is the ground state $|00000\rangle$, where only the first element of the state vector is one and all others are zero. We used the statevector simulation method, as it is particular suitable for simulating pure state, as mixed state simulation is significantly more computationally expensive (see Sec. 2.2) The library implements the matrix representation of the universal gateset defined in Sec. 2.3.2 using $\{H, X, Y, Z, T, CNOT\}$ as our gates.

Single-qubit gates are implemented using their standard 2×2 unitary matrices, while two-qubit gates such as the CNOT use their usual 4×4 form. To apply any of these gates to the full five-qubit system, the corresponding operator is lifted to the entire state space by inserting identity matrices on all qubits that are not affected by the operation. This expansion yields a $2^5 \times 2^5$ matrix for each gate application. Consequently, every layer of the circuit acts as a 32×32 unitary on the global statevector, and the simulation proceeds by performing matrix–vector multiplications. This approach allows the simulation to be performed in real time on embedded systems with limited computational resources. As we don't have to deal with higher qubit counts, it is computational feasible to use this method, as only higher qubit counts of magnitude 20 or above, makes the system more computational intensive.

Measurements of all five qubits are simulated using probabilistic sampling. The statevector is first normalized to ensure total probability equals one, satisfying Eq. 2.29. For each shot, a random number in $[0, 1)$ is generated to select an outcome according to the cumulative probabilities. On the ESP32, this number is produced by the hardware random generator and is therefore pseudorandom. Repeating many shots approximates the ideal quantum probability distribution, as pseudorandom numbers are also used for simulation methods [65]. To enable a non-deterministic measurement outcome, we could further integrate the temperature sensor of the ESP32 as a source of entropy for random number generation to ensure greater randomness in the measurement process.

5.2.2 Hardware Setup and Gate Execution

As mentioned in Sec. 4.2, the complete hardware system is built around an ESP32 microcontroller, which handles all control and computation tasks. To overcome the limited number of GPIO pins on the ESP32, I/O expanders (PCF8575) are employed. The I/O expanders are connected to two GPIO pins, which are depicted in Fig. 4.3, on the ESP32, ensuring the data transfer and receivement of the slaves information. To connect to multiple I/0 expanders WAGO terminal blocks were used for the bus connections, simplifying assembly and reducing the complexity of soldering individual wires. This minimizes the amount of GPIO pins used, as the pin count of the ESP32 was limited and needed good structure. We tested the alternative of soldering all wires directly to each other, but this didn't result in better performance, so we decided to settle with the WAGO terminal blocks for better maintainability.

This configuration allows the system to manage up to 40 RFID-modules simultaneously. Each I/O expander represents a logical qubit and provides addressable channels, each connected to one RFID-module. For the five qubit system, for each qubit an expander is used. Each CS pin of the RFID-module is connected to one of the addresses of the expanders, depcited in Fig. 4.4. The SPI bus is shared among all readers, while the 3.3V power lines are isolated to avoid crosstalk. The CS lines are routed through the expanders, ensuring that only one reader is active at any given moment.

To reduce the number of GPIO pins required on the ESP32, the SPI communication lines MISO, MOSI, SCLK, and RST, seen in Fig. 4.5a, are not wired individually to each RFID-module. Instead, they are implemented as a shared bus system arranged in a tree-like topology. The ESP32 sends a request signal on the MOSI line, requesting information from the currently selected reader. Only the RFID-module whose CS line is active responds by sending its unique identifier back to the ESP32 via the MISO line.

This identifier encodes which quantum gate is present on the tag. All other readers keep their MISO outputs in a high-impedance state, ensuring that the ESP32 receives only the data of the active reader. The shared clock line synchronizes all data transfers, and all readers see the clock signal, but only the selected reader participates in communication. The shared RST line allows the ESP32 to reset all readers after gathering the entire loop, ensuring that tags are detected reliably during continuous operation. All bus lines are common-grounded and routed in parallel. Power distribution and grounding were designed carefully to avoid signal degradation across the bus network. During each scan cycle, the ESP32 iterates over all qubits and their associated gate positions by toggling the CS lines of the respective expanders. Whenever a tag is present, the corresponding RFID-module transmits its unique identifier, indicating the assigned quantum gate. All detected gates are stored in a buffer in the order qubit and gate position. Certain multi-qubit operations, such as CNOT, cannot be executed immediately and are temporarily stored until the complete scan is finished, ensuring correct control–target assignment and consistent logical qubit mapping. After the full scan, the ESP32 constructs the corresponding quantum circuit and simulates it using the custom library, defined in Sec. 5.2.1. The resulting quantum state is sampled 1000 times to obtain a stable probability distribution.

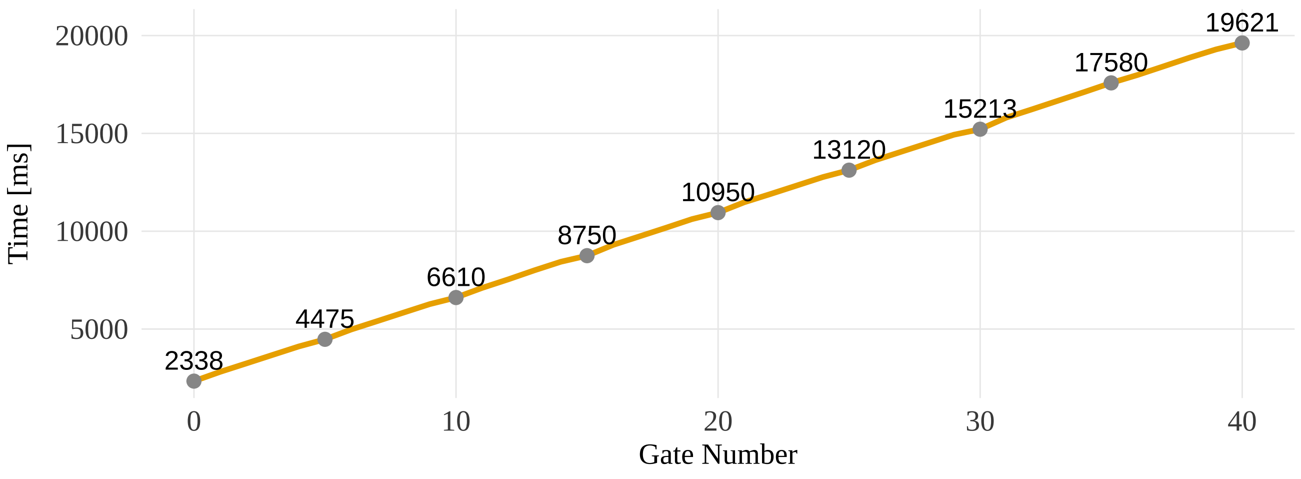


Figure 5.7: Loop time to calculate and visualize the result for our simulator. Placing more gates increases the loop time linearly, as each gate requires a fixed amount of time to read and process. Placing no gates results in a baseline overhead of 2338 ms to check and read all RFID-modules.

To maximize performance, the SPI clock was set to 2 MHz, the highest frequency that still guarantees reliable tag detection. The I^2C bus for the expanders runs at 1.52 MHz; higher frequencies resulted in missed tags and therefore were unsuitable for stable operation. This is caused by reduced signal integrity, higher noise sensitivity, and timing limitations of the I/O expanders at higher frequencies [66].

Placing a gate on the board increases the total loop time until the full quantum state is visualized. The total loop time t_{looptime} is composed of several measurable components consisting of the transmission time t_{transmit}, which is dominated by the physical reading and communication of all RFID-modules, the quantum gate computation time t_{circsim}, representing the time required to apply all placed quantum gates to the simulated state vector, the measurement time $t_{\text{measurement}}$, which describes the time required to compute the probability distribution for 1000 shots, and the drawing time t_{draw}, denoting the visualization time needed to display the final state on the LED-matrix. The total loop time is therefore expressed as:

$$t^y_{\text{looptime}} = t^y_{\text{transmit}} + t^y_{\text{circsim}} + t^y_{\text{measurement}} + t_{\text{draw}}, \tag{5.4}$$

where t_x describes the time of its respective component in the entire loop for $x \in$ [looptime, transmit, circsim, measurment, draw] and the exponent y the gates placed in the loop for $y \in [0, 1, 2, ..., 40]$. When no gates were placed, the transmission time was measured to be $t^0_{\text{transmit}} = 2337.80$ ms. Using the full configuration of 40 gates, $t_{\text{transmit,each}}$, which is the approximate time for the information of each RFID-module to be read transmitted to the ESP32, can be estimated as:

$$t_{\text{transmit,each}} \approx \frac{t^{40}_{\text{transmit}} - t^0_{\text{transmit}}}{40} = 426.55 \text{ ms/gate}. \tag{5.5}$$

Thus, t_{transmit} can be written as

$$\begin{aligned} t^y_{\text{transmit}} &\approx t^0_{\text{transmit}} + t_{\text{transmit,each}} \cdot N_{\text{gate}} \\ &= 2337.80 \text{ ms} + 426.55 \text{ ms} \cdot N_{\text{gate}}, \end{aligned} \tag{5.6}$$

where N_{gate} is the number of placed gates on the board. The drawing time remained constant as $t_{\text{draw}} = 0.78$ ms, which is to be expected as the visualization process does not depened on the number of gates placed and stays for each loop the same.

Analogously, the same approach can be used for $t^y_{\text{measurement}}$, but we need to consider that if no gates are placed, the computation process is skipped, resulting in $t^0_{\text{measurement}} = 0$ ms. Therefore, we derive for each measurement $t_{\text{measurement,each}} \approx 0.407$ ms per gate, so that the total measurement time is

$$\begin{aligned} t^y_{\text{measurement}} &\approx t^0_{\text{measurement}} + t_{\text{measurement,each}} \cdot N_{\text{gate}} \\ &= 0 + t_{\text{measurement,each}} \cdot N_{\text{gate}} \\ &= 0.407 \text{ ms} \cdot N_{\text{gate}} \end{aligned} \tag{5.7}$$

Similarly, the state-vector simulation time per gate can be formulated as $t_{\text{gate,each}} \approx 0.113$ ms/gate, and the total gate computation time is:

$$\begin{aligned} t^y_{\text{circsim}} &\approx t^0_{\text{circsim}} + t_{\text{gate,each}} \cdot N_{\text{gate}} \\ &= 0 + 0.113 \text{ ms} \cdot N_{\text{gate}} \\ &= 0.113 \text{ ms} \cdot N_{\text{gate}}. \end{aligned} \tag{5.8}$$

Combining all components, Eq. 5.4 can be written as

$$\begin{aligned} t^y_{\text{looptime}} &\approx t^y_{\text{transmit}} + t^y_{\text{circsim}} + t^y_{\text{measurement}} + t_{\text{draw}} \\ &= 2337.80 \text{ ms} + 426.55 \text{ ms} \cdot N_{\text{gate}} + 0.113 \text{ ms} \cdot N_{\text{gate}} + 0.407 \text{ ms} \cdot N_{\text{gate}} + 0.78 \text{ ms} \\ &= 2338.58 \text{ ms} + 427.07 \text{ ms} \cdot N_{\text{gate}}. \end{aligned} \tag{5.9}$$

Fig. 5.7 shows the measured loop time as a function of the number of placed gates. The clearly linear trend confirms the previously derived relationship: as more gates are added to the circuit, the total loop time increases proportionally. The baseline offset is primarily caused by the constant time required for gathering the informations of the RFID-modules and handling the communication with the I^2C expanders. As discussed earlier, the SPI clock frequency could not be increased further, since higher communication speeds led to unreliable tag detection. This limitation results in longer loop times for circuits with many gates. However, this does not pose a significant issue, as the system primarily serves educational purposes.

To position each RFID-module at its intended location, a CAD sketch of the complete layout was first created and used to mill the outer contour of the system into a wooden board, as wood is not conductive material, which meet the requirements of the demonstrator. All RFID-modules were then mounted according to this template, and the entire bus system was carefully wired on the back side of the board.

5.2.3 Visualization of Quantum States

To provide an intuitive understanding of the quantum state after circuit execution, the 5-qubit state vector is visualized on a 64×64 LED-matrix display connected to the ESP32 via the I^2C bus. After calculating the entire distribution each of the 32 possible basis states is represented by a vertical bar whose height corresponds to the probability amplitude, scaled to 64 pixels. The visualisation updates after each measurement loop, allowing users to observe the evolution of the quantum state as different gates are applied through the RFID-tags. The LED-matrix, which requires 16 GPIO pins to be connected, met the entire GPIO resources given by the ESP32.

To improve visual clarity, each bar changes its color above 50% probability, helping users easily identify dominant states. This real-time visualisation thus bridges the gap between quantum computation and human perception, providing both a functional and educational tool for exploring quantum state dynamics on embedded hardware.

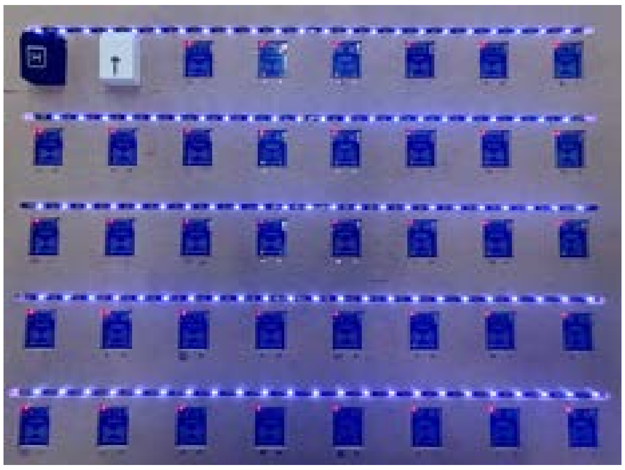
5.2.4 Edge-Cases on the Simulator

To correctly handle two-qubit gates such as the CNOT, several edge cases had to be addressed to prevent incorrect or ambiguous operations.

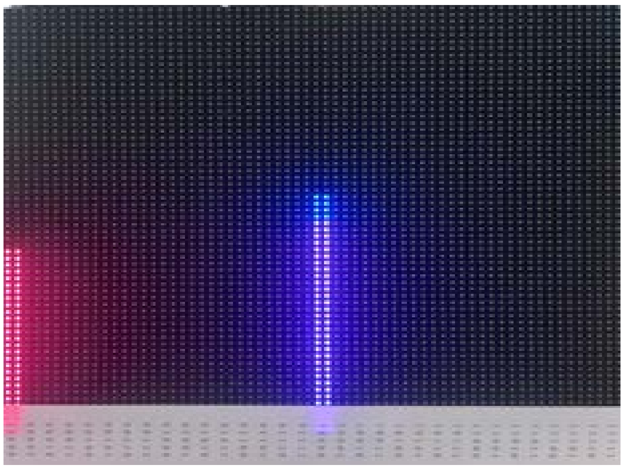
The first challenge is to reliably match each CNOT control with its corresponding target. To achieve this, every RFID-tag encoding a CNOT gate contains an additional attribute, which is a unique integer that identifies the pair. For users and testers, this pairing is also visually indicated through color-coded CNOT pairs on the hardware, seen in Fig. 5.9a.

A second edge case occurs when only one part of the CNOT pair is detected during a scan, that is when the user forgets to place the second half of the pair, or when the reading loop has already passed the corresponding gate position. In such a scenario, the system does not apply the CNOT gate. Instead, it temporarily skips its execution and waits until both the control and target RFID-tags have been detected in a complete scan cycle. This behaviour is illustrated in Fig. 5.8.

The final edge case concerns invalid placements where both the control and the target of the CNOT gate are placed in the same row (i.e., the same qubit line). Many software-based quantum simulators classify this as an error and terminate the execution. In contrast, our system instead resolves this conflict by collapsing the circuit into the ground state, resulting in a probability distribution of 100% in $|00000\rangle$.

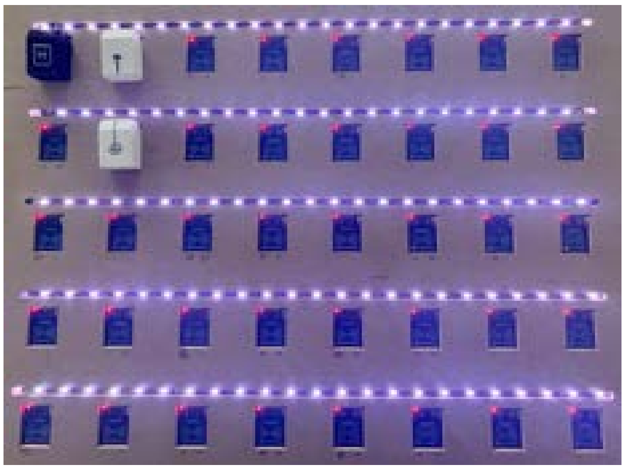


(a) Entanglement edge-case, where the second gate was not placed. In this scenario only the hadamard gate was executed till the second part of the cnot is placed.

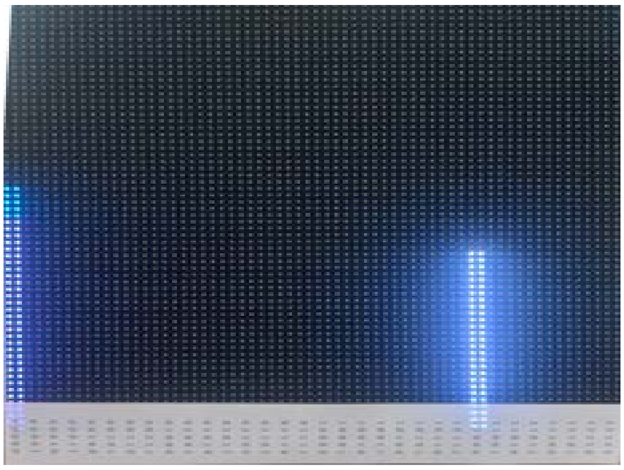


(b) Visualization of the edge-case.

Figure 5.8: Comparison between the edgecase circuit (left) and its resulting measurement distribution (right).



(a) Correct placement of the CNOT gate.

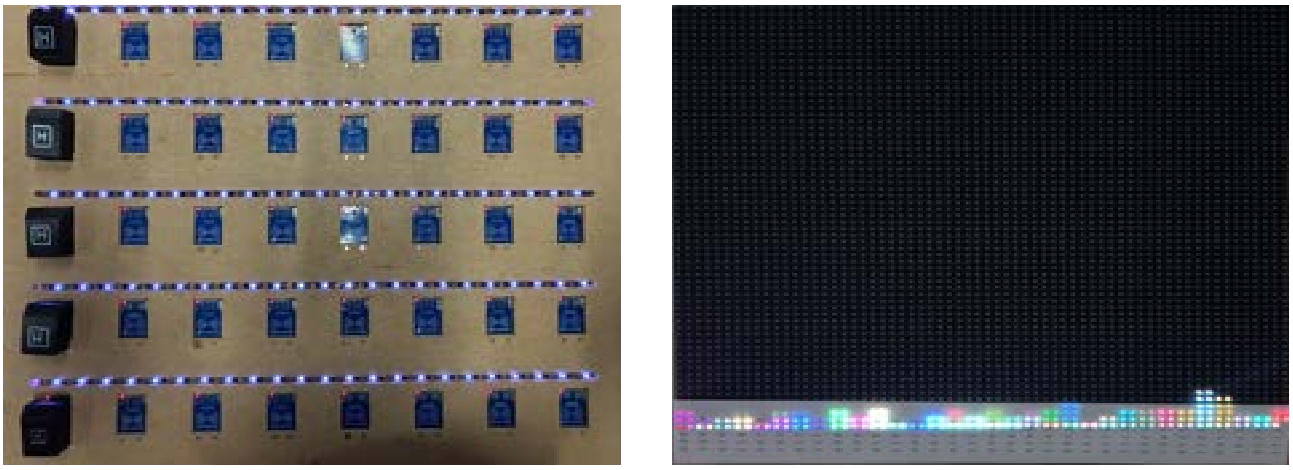


(b) Probability distribution of the correct cnot placement state.

Figure 5.9: Comparison between the correct edgecase circuit (left) and its resulting measurement distribution (right).

5.2.5 Hands-On Tests on the Simulator

To show the functionality of the quantum simulator, several test circuits were implemented and executed on the hardware prototype.



(a) Hadamard circuit with 5 qubits. **(b)** Probability distribution of the hadamard state.

Figure 5.10: Comparison between the Hadamard circuit (left) and its resulting measurement distribution (right).

To test the correct implementation of single-qubit gates, a simple circuit was designed which implemented on each qubit a Hadamard gate. After applying the Hadamard gate to each qubit initialized in the $|0\rangle$ state, the expected output state is a uniform superposition of all basis states:

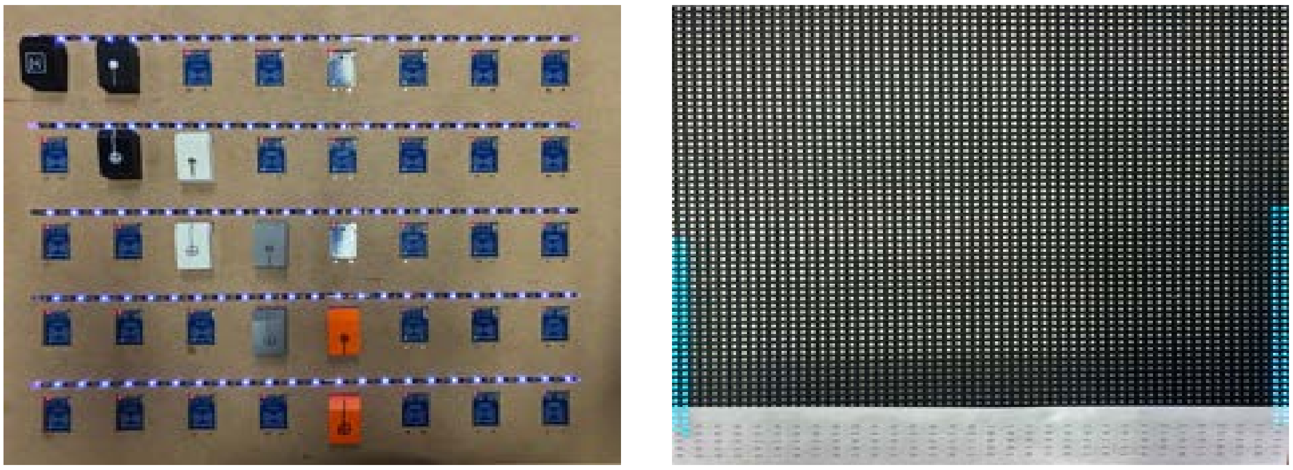
$$|\psi\rangle = \frac{1}{\sqrt{2^n}} \sum_{i=0}^{2^n-1} |i\rangle \tag{5.10}$$

where n is the number of qubits. For a 5-qubit system, this results in a superposition of all 32 basis states with equal probability amplitudes of $\frac{1}{\sqrt{32}} \approx 0.1768$. After executing the circuit on the hardware prototype and performing 1000 measurement shots, the resulting probability distribution, seen in Fig. 5.10b, closely matched the expected uniform distribution, confirming the correct implementation of single qubit gates.

Additionally, we tested the implementation of entangling gates by creating a Bell state, explained in Sec. 2.3, between five qubits. This circuit is also known as a GHZ circuit, which generates a maximally entangled state across all qubits [67]. The circuit consisted of applying a Hadamard gate to the first qubit followed by a series of CNOT gates entangling all qubits. The expected output state for five qubits isolated in a Bell state is:

$$|\psi\rangle = \frac{1}{\sqrt{2}}(|00000\rangle + |11111\rangle) \tag{5.11}$$

After executing this circuit on the hardware prototype, Fig. 5.11 showed approximately 50% probability for both $|00000\rangle$ and $|11111\rangle$ states, confirming the correct implementation of the CNOT gate and the ability of the simulator to create entangled states and



(a) GHZ circuit with 5 qubits.

(b) Probability distribution of the GHZ state.

Figure 5.11: Comparison between the GHZ circuit (left) and its resulting measurement distribution (right).

therefore enabling use it with the entire power of all states. These hands-on tests demonstrate the functionality and correctness of the quantum simulator implemented on the embedded hardware prototype. The successful execution of both single qubit and multi qubit circuits validated the design and implementation choices made for the simulator library and hardware integration.

6 Discussion and Outlook

In the previous chapters, we systematically studied the performance of VQ-DQL on an Anki Overdrive car demonstrator and present a custom microcontroller-based embedded quantum simulator. Our findings revealed the technical feasibility of deploying quantum-inspired concepts on resource-constrained hardware provided crucial insights into their performance characteristics and limitations in real-world and embedded environments.

The empirical study demonstrated that the VQ-DQL agent successfully learned to control the physical Anki Overdrive vehicle under realistic constraints. Although the quantum agent achieved a performance comparable to the classical DQN baseline, no clear improvements could be identified, due to hardware constrained training duration (see Fig. 5.6). Interestingly, while the classical agent converged slightly faster due to its substantially higher number of trainable parameters, the quantum agent exhibited a notably more stable learning curve with reduced variance across episodes. This supports the finding that quantum models may provide smoother and more stable function approximations in reinforcement learning tasks. This can be seen in the results of Sec. 5.1.2.

However, the optimization of the quantum agent was strongly affected by the barren plateau problem, leading to vanishing gradients and reduced learning efficiency. This behavior is consistent with known challenges in variational quantum algorithms. Since the VQCs in our experiments were initialized systematically and were neither very wide nor deep, the observed instabilities are likely due to the high sensitivity of VQ-DQL to classical hyperparameters [14]. Furthermore, our evaluation of different VQC layer depths showed that the deeper data re-uploading strategy did not always better convergence. Still, while the 4-layer architecture converged the fastest, deeper architectures showed more stable learning dynamics (see Fig. 5.3). This suggests that future improvements depend on enhanced circuit architectures or alternative optimization techniques that balance expressivity with trainability. Further research could be done by gathering training data and performing offline training on more powerful hardware to overcome the limitations of the embedded platform. Nonetheless, the successful operation of the quantum agent on embedded hardware demonstrates that quantum-enhanced learning methods can be integrated into small autonomous systems and presenting QRL to a broader audience.

The second demonstrator, the portable microcontroller-based QC platform built on an ESP32, further highlights the possibilities for physical demonstrators for educational purposes. Using a custom state-vector simulator and RFID-encoded quantum gates, the system allows users to construct and evaluate simple quantum circuits in a tangible and intuitive manner. Functional tests confirmed the correct execution of fundamental operations, including the creation of fully entangled Bell states (see Eq. 2.10), validating the underlying design (see Fig. 5.11a, Fig. 5.11b). The primary limitation of this prototype concerns its restricted qubit count and gate set, which limit the complexity of circuits that can be explored. The required loop time for calculating and visualizing the quantum state increases linearly with the number of gates, resulting in long loop times for higher gate counts, which are depicted in Fig. 5.7. This suggests that future demonstrators need faster protocols, better datastructures or more powerful microcontroller to enhance scalability. Beyond their technical contributions, both demonstrators offer substantial educational value by translating abstract quantum concepts into concrete, interactive experiences accessible even for non-experts. Further extensions could be done by integrating rotational gates into the demonstrator, hence allowing to construct parameterized circuits for variational algorithms by manually adjusting the rotational parameters.

Overall, this work shows that embedded quantum demonstrators, despite their current limitations in the NISQ-Era, already provide meaningful opportunities for experimentation, learning, and prototyping. Continued research into improved optimization strategies, scalable hardware designs, and hybrid algorithmic approaches will be essential for unlocking their full potential in both scientific and educational contexts.

7 Conclusion

This thesis demonstrated that quantum-inspired algorithms and QC concepts can be integrated into compact embedded systems. The RL experiments showed that a VQ-DQL agent can operate under strict hardware constraints and achieve performance comparable to a classical DQN, although no clear improvements could be identified. These results underline both the potential and current limitations of quantum-enhanced learning on NISQ hardware.

The second contribution of this work was a lightweight QC demonstrator based on an ESP32 microcontroller. It enabled the execution of basic quantum circuits, including superposition and entanglement, and highlighted how low-cost platforms can support accessible, hands-on quantum education despite scalability limits.

Together, both demonstrators show that embedded hardware can serve as an effective entry point into QC and QML. They provide a foundation for future research, particularly in improving optimization methods and developing educational tools that make quantum concepts more tangible.

Acronyms

BLE Bluetooth Low Energy. 22

CS Chip Select. 25, 26, 38, 39

DP Dynamic Programming. 17

DQL Deep Q-Learning. 1, 15, 18, 19, 20

DQN Deep Q-Network. I, II, 18, 20, 47

GND Ground. 26

GPIO general-purpose input/output. 23, 24, 38, 41

I^{2}C Inter-Integrated Circuit. 24, 26, 39, 41

MDP Markov Decision Process. 15, 16, 17, 20, 50

MISO Master Input Slave Output. 25, 27, 38, 39

MOSI Master Output Slave Input. 25, 27, 38

NISQ Noisy Intermediate-Scale Quantum. I, II, 1, 11, 13, 46, 47

NN Neural Network. 11, 13, 18, 29, 36, 37, 51

QC Quantum Computing. I, II, 1, 2, 3, 5, 6, 7, 8, 9, 20, 21, 37, 45, 47

QML Quantum Machine Learning. I, II, 1, 47

QRL Quantum Reinforcement Learning. 1, 7, 9, 10, 11, 29, 35, 45

RFID Radio Frequency Identification. I, II, 23, 24, 25, 26, 27, 38, 39, 40, 41, 42, 45, 50, 51, 52

RL Reinforcement Learning. 1, 2, 15, 16, 17, 20, 21, 22, 23, 29, 31, 47, 50

RST Reset. 25, 38, 39

SCL clock line. 26

SCLK Serial Clock. 25, 27, 38

SDA data line. 26

SPI Serial Peripheral Interface. 24, 26, 27, 38, 39, 41

TDL Temporal-Difference Learning. 15, 17, 18

VCC Voltage Common Collector. 26

VQA Variational Quantum Algorithm. 3, 9, 10, 12

VQC Variational Quantum Circuit. 8, 9, 10, 11, 12, 13, 20, 29, 31, 32, 33, 34, 36, 45, 50

VQ-DQL Variational Quantum Deep Q-Learning. I, II, 1, 20, 27, 29, 31, 35, 37, 45, 47, 50, 51

List of Figures

List of Tables

Bibliography

[1] L. K. Grover, “A fast quantum mechanical algorithm for database search,” in *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, ser. STOC ’96, Philadelphia, Pennsylvania, USA: Association for Computing Machinery, 1996, pp. 212–219, ISBN: 0897917855. DOI: 10.1145/237814.237866. [Online]. Available: https://doi.org/10.1145/237814.237866.

[2] P. W. Shor, “Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer,” *SIAM Review*, vol. 41, no. 2, pp. 303–332, 1999. DOI: 10.1137/S0036144598347011. [Online]. Available: https://doi.org/10.1137/S0036144598347011.

[3] J. Preskill, “Quantum Computing in the NISQ era and beyond,” *Quantum*, vol. 2, p. 79, Aug. 2018, ISSN: 2521-327X. DOI: 10.22331/q-2018-08-06-79. [Online]. Available: https://doi.org/10.22331/q-2018-08-06-79.

[4] K. Mitarai, M. Negoro, M. Kitagawa, *et al.*, “Quantum circuit learning,” *Phys. Rev. A*, vol. 98, p. 032 309, 3 Sep. 2018. DOI: 10.1103/PhysRevA.98.032309. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevA.98.032309.

[5] K. Bharti, A. Cervera-Lierta, T. H. Kyaw, *et al.*, “Noisy intermediate-scale quantum algorithms,” *Rev. Mod. Phys.*, vol. 94, p. 015 004, 1 Feb. 2022. DOI: 10.1103/RevModPhys.94.015004. [Online]. Available: https://link.aps.org/doi/10.1103/RevModPhys.94.015004.

[6] M. Franz, L. Wolf, M. Periyasamy, *et al.*, “Uncovering instabilities in variational-quantum deep q-networks,” *Journal of the Franklin Institute*, vol. 360, no. 17, pp. 13 822–13 844, Nov. 2023, ISSN: 0016-0032. DOI: 10.1016/j.jfranklin.2022.08.021. [Online]. Available: http://dx.doi.org/10.1016/j.jfranklin.2022.08.021.

[7] Y. Liu, S. Arunachalam, and K. Temme, “A rigorous and robust quantum speed-up in supervised machine learning,” *Nature Physics*, vol. 17, no. 9, pp. 1013–1017, Jul. 2021, ISSN: 1745-2481. DOI: 10.1038/s41567-021-01287-z. [Online]. Available: http://dx.doi.org/10.1038/s41567-021-01287-z.

[8] D. Silver, J. Schrittwieser, K. Simonyan, *et al.*, “Mastering the game of go without human knowledge,” *nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[9] A. Mjeda and H. Murray, *Quantum computing education for computer science students: Bridging the gap with layered learning and intuitive analogies*, 2024. arXiv: 2405.09265 [cs.ET]. [Online]. Available: https://arxiv.org/abs/2405.09265.

[10] S. Y.-C. Chen, C.-H. H. Yang, J. Qi, *et al.*, “Variational quantum circuits for deep reinforcement learning,” *IEEE Access*, vol. 8, pp. 141 007–141 024, 2020. DOI: 10.1109/ACCESS.2020.3010470.

[11] A. Skolik, S. Jerbi, and V. Dunjko, "Quantum agents in the Gym: A variational quantum algorithm for deep Q-learning," *Quantum*, vol. 6, p. 720, May 2022, ISSN: 2521-327X. DOI: `10.22331/q-2022-05-24-720`. [Online]. Available: `https://doi.org/10.22331/q-2022-05-24-720`.

[12] M. A. Nielsen and I. L. Chuang, *Quantum computation and quantum information*. Cambridge university press, 2010.

[13] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information - 10th Anniversary Edition*. Cambridge: Cambridge University Press, 2010, ISBN: 978-1-107-00217-3.

[14] M. Franz, L. Wolf, M. Periyasamy, *et al.*, "Uncovering instabilities in variational-quantum deep q-networks," *Journal of the Franklin Institute*, 2022, ISSN: 0016-0032. DOI: `10.1016/j.jfranklin.2022.08.021`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S0016003222005725`.

[15] J. S. Bell, "On the einstein podolsky rosen paradox," *Physics Physique Fizika*, vol. 1, no. 3, p. 195, 1964.

[16] A. Einstein, B. Podolsky, and N. Rosen, "Can quantum-mechanical description of physical reality be considered complete?" *Physical review*, vol. 47, no. 10, p. 777, 1935.

[17] D. E. Deutsch, "Quantum computational networks | Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences," *Proc. Roy. Soc. Lond. A*, vol. Volume 425, no. Issue 1868, pp. 73–90, 1989. DOI: `10.1098/rspa.1989.0099`.

[18] E. G. Rieffel and W. H. Polak, *Quantum computing: A gentle introduction*. MIT press, 2011.

[19] E. G. Rieffel and W. H. Polak, *Quantum Computing - A Gentle Introduction*. Cambridge: MIT Press, 2014, ISBN: 978-0-262-52667-8.

[20] S. Bravyi and A. Kitaev, "Universal quantum computation with ideal clifford gates and noisy ancillas," *Physical Review A*, vol. 71, no. 2, Feb. 2005, ISSN: 1094-1622. DOI: `10.1103/physreva.71.022316`. [Online]. Available: `http://dx.doi.org/10.1103/PhysRevA.71.022316`.

[21] G. Biswas, "On classical simulation of quantum circuits composed of clifford gates," *Quanta*, vol. 13, pp. 20–27, Jul. 2024, ISSN: 1314-7374. DOI: `10.12743/quanta.v13i1.265`. [Online]. Available: `http://dx.doi.org/10.12743/quanta.v13i1.265`.

[22] M. Cerezo, A. Arrasmith, R. Babbush, *et al.*, "Variational quantum algorithms," *Nature Reviews Physics*, vol. 3, no. 9, pp. 625–644, 2021.

[23] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, 2017. arXiv: `1412.6980 [cs.LG]`. [Online]. Available: `https://arxiv.org/abs/1412.6980`.

[24] M. Weigold, J. Barzen, F. Leymann, *et al.*, "Encoding patterns for quantum algorithms," *IET Quantum Communication*, vol. 2, no. 4, pp. 141–152, 2021.

[25] M. Weigold, J. Barzen, F. Leymann, *et al.*, "Data encoding patterns for quantum computing," in *Proceedings of the 27th Conference on Pattern Languages of Programs*, ser. PLoP '20, Virtual Event: The Hillside Group, 2022, ISBN: 9781941652169.

[26] M. Mottonen and J. J. Vartiainen, *Decompositions of general quantum gates*, 2005. arXiv: quant-ph/0504100 [quant-ph].

[27] A. Skolik, S. Mangini, T. Bäck, *et al.*, *Robustness of quantum reinforcement learning under hardware errors*, 2022. arXiv: 2212.09431 [quant-ph].

[28] O. Lockwood and M. Si, "Reinforcement learning with quantum variational circuit," in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 16, 2020, pp. 245–251.

[29] A. Skolik, S. Mangini, T. Bäck, *et al.*, "Robustness of quantum reinforcement learning under hardware errors," *EPJ Quantum Technology*, vol. 10, no. 1, pp. 1–43, 2023.

[30] A. Pérez-Salinas, A. Cervera-Lierta, E. Gil-Fuster, *et al.*, "Data re-uploading for a universal quantum classifier," *Quantum*, vol. 4, p. 226, Feb. 2020, ISSN: 2521-327X. DOI: 10.22331/q-2020-02-06-226. [Online]. Available: https://doi.org/10.22331/q-2020-02-06-226.

[31] M. Schuld, R. Sweke, and J. J. Meyer, "Effect of data encoding on the expressive power of variational quantum-machine-learning models," *Phys. Rev. A*, vol. 103, p. 032 430, 3 Mar. 2021. DOI: 10.1103/PhysRevA.103.032430. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevA.103.032430.

[32] J. R. McClean, J. Romero, R. Babbush, *et al.*, "The theory of variational hybrid quantum-classical algorithms," *New Journal of Physics*, vol. 18, no. 2, p. 023 023, Feb. 2016. DOI: 10.1088/1367-2630/18/2/023023. [Online]. Available: https://doi.org/10.1088/1367-2630/18/2/023023.

[33] M. Schuld, V. Bergholm, C. Gogolin, *et al.*, "Evaluating analytic gradients on quantum hardware," *Phys. Rev. A*, vol. 99, p. 032 331, 3 Mar. 2019. DOI: 10.1103/PhysRevA.99.032331. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevA.99.032331.

[34] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989, ISSN: 0893-6080. DOI: https://doi.org/10.1016/0893-6080(89)90020-8. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0893608089900208.

[35] A. G. Barto, "Reinforcement learning: An introduction. by richard's sutton," *SIAM Rev*, vol. 6, no. 2, p. 423, 2021.

[36] A. Markov and N. Nagorny, *The Theory of Algorithms* -. Dortrecht Heidelberg London New York: Springer Netherlands, 1988, ISBN: 978-9-027-72773-2.

[37] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, pp. 279–292, 1992.

[38] F. S. Melo, "Convergence of q-learning: A simple proof," *Institute Of Systems and Robotics, Tech. Rep*, pp. 1–4, 2001.

[39] V. Mnih, K. Kavukcuoglu, D. Silver, *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[40] H. van Hasselt, A. Guez, and D. Silver, *Deep reinforcement learning with double q-learning*, 2015. arXiv: 1509.06461 [cs.LG].

[41] B. Polyak, *Introduction to Optimization*. Optimization Software, Publications Division, 1987, ISBN: 978-0-911-57514-9.

[42] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014, Published as a conference paper at ICLR 2015. [Online]. Available: https://arxiv.org/abs/1412.6980.

[43] L.-J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Machine learning*, vol. 8, pp. 293–321, 1992. DOI: 10.1007/BF00992699. [Online]. Available: https://doi.org/10.1007/BF00992699.

[44] S. Wiedemann, D. Hein, S. Udluft, *et al.*, *Quantum policy iteration via amplitude estimation and grover search – towards quantum advantage for reinforcement learning*, 2023. arXiv: 2206.04741 [quant-ph]. [Online]. Available: https://arxiv.org/abs/2206.04741.

[45] W. Roscoe and the Donkeycar Community, *Donkeycar: Open source self driving library for python*, https://github.com/autorope/donkeycar, Accessed: 2025-12-12, 2016.

[46] I. Harvey, *Bluepy: Python interface to bluetooth le on linux*, https://github.com/IanHarvey/bluepy, Last time accessed: 2025-11-13, 2025.

[47] AZDelivery. "Esp32 development board datasheet." Accessed: 2025-04-30. (2025), [Online]. Available: https://cdn.shopify.com/s/files/1/1509/1638/files/AZ281_A_18-10_EN_B08BTWJGFX_30680600-27af-4387-b729-6cb1aee15e5d.pdf.

[48] W. Gay, *Raspberry Pi Hardware Reference*. Jan. 2014, ISBN: 978-1-4842-0800-7. DOI: 10.1007/978-1-4842-0799-4.

[49] NXP Semiconductors. "Pcf8575 remote 16-bit i/o expander for i2c bus: Data sheet." Accessed: 2025-11-13. (1999), [Online]. Available: https://www.nxp.com/docs/en/data-sheet/PCF8575.pdf.

[50] NXP Semiconductors. "Mfrc522 standard performance mifare® and ntag® frontend: Data sheet." Accessed: 2025-11-13. (2014), [Online]. Available: https://www.alldatasheet.com/datasheet-pdf/pdf/227839/NXP/MFRC522.html.

[51] Adafruit Industries. "64×64 rgb led matrix – 2.5 mm pitch." Accessed: 2025-11-13. (2025), [Online]. Available: https://www.adafruit.com/product/3649.

[52] *Um10204 i2c-bus specification and user manual*, Revision 7.0, NXP Semiconductors, 2021. [Online]. Available: https://www.nxp.com/docs/en/user-guide/UM10204.pdf.

[53] Y. Yang and S. Zhou, "I2c data transfer program design and communication protocol improvement," *Clausius Scientific Press*, 2018. [Online]. Available: http://www.clausiuspress.com/front/pdf/201808/161396155230.pdf.

[54] *Spi block guide v03.06*, Motorola, 2003. [Online]. Available: https://www.nxp.com/docs/en/reference-manual/SPI_Block_Guide.pdf.

[55] N. Semiconductors. "Mfrc522 standard performance mifare® and ntag® frontend: Data sheet." Accessed: 2025-11-13. (2014), [Online]. Available: https://www.alldatasheet.com/datasheet-pdf/pdf/227839/NXP/MFRC522.html.

[56] S. Huang, R. F. J. Dossa, C. Ye, *et al.*, “Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms,” *Journal of Machine Learning Research*, vol. 23, no. 274, pp. 1–18, 2022. [Online]. Available: `http://jmlr.org/papers/v23/21-1342.html`.

[57] V. Bergholm, J. Izaac, M. Schuld, *et al.*, *Pennylane: Automatic differentiation of hybrid quantum-classical computations*, 2022. arXiv: `1811.04968 [quant-ph]`. [Online]. Available: `https://arxiv.org/abs/1811.04968`.

[58] Qiskit contributors, *Qiskit: An open-source framework for quantum computing*, 2023. DOI: `10.5281/zenodo.2573505`.

[59] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in Neural Information Processing Systems*, F. Pereira, C. Burges, L. Bottou, *et al.*, Eds., vol. 25, Curran Associates, Inc., 2012. [Online]. Available: `https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf`.

[60] V. Nair and G. E. Hinton, “Rectified linear units improve restricted boltzmann machines,” in *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ser. ICML’10, Haifa, Israel: Omnipress, 2010, pp. 807–814, ISBN: 9781605589077.

[61] A. Cahill, “Catastrophic forgetting in reinforcement-learning environments,” Ph.D. dissertation, University of Otago, 2011.

[62] J. R. McClean, S. Boixo, V. N. Smelyanskiy, *et al.*, “Barren plateaus in quantum neural network training landscapes,” *Nature Communications*, vol. 9, no. 1, p. 4812, 2018, ISSN: 2041-1723. DOI: `10.1038/s41467-018-07090-4`. [Online]. Available: `https://www.nature.com/articles/s41467-018-07090-4` (visited on 05/23/2022).

[63] A. Sequeira, L. P. Santos, and L. S. Barbosa, *Policy gradients using variational quantum circuits*, 2023. arXiv: `2203.10591 [quant-ph]`. [Online]. Available: `https://arxiv.org/abs/2203.10591`.

[64] J. Robine, T. Uelwer, and S. Harmeling, “Smaller world models for reinforcement learning,” *Neural Processing Letters*, vol. 55, no. 8, pp. 11 397–11 427, 2023.

[65] J. E. Gentle, *Random Number Generation and Monte Carlo Methods*, 2nd ed. New York: Springer, 2003.

[66] A. Albalooshi, A.-H. M. Jallad, and P. R. Marpu, “Fault analysis and mitigation techniques of the i2c bus for nanosatellite missions,” *IEEE Access*, vol. 11, pp. 34 709–34 717, 2023.

[67] D. M. Greenberger, M. A. Horne, and A. Zeilinger, *Going beyond bell’s theorem*, 2007. arXiv: `0712.0921 [quant-ph]`. [Online]. Available: `https://arxiv.org/abs/0712.0921`.

Erklärung

1. Mir ist bekannt, dass dieses Exemplar der Bachelor Thesis als Prüfungsleistung in das Eigentum der Ostbayerischen Technischen Hochschule Regensburg übergeht.
2. Ich erkläre hiermit, dass ich diese Bachelor Thesis selbstständig verfasst, noch nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

Ort, Datum und Unterschrift

Presented by:	Benjamin Zec
Student ID:	3316636
Study Programme:	Allgemeine Informatik
Supervisor:	Prof. Dr. Wolfgang Mauerer
Secondary Supervisor:	Prof. Dr. Maike Stern