energies

MDPI

Article

On the Use of Quantum Reinforcement Learning in Energy-Efficiency Scenarios

Eva Andrés, Manuel Pegalajar Cuéllar * and Gabriel Navarro

Department of Computer Science and Artificial Intelligence, ETSI Informática y de Telecomunicación, Universidad de Granada, C/. Pdta Daniel Saucedo Aranda sn, 18014 Granada, Spain
* Correspondence: manupc@decsai.ugr.es

Abstract: In the last few years, deep reinforcement learning has been proposed as a method to perform online learning in energy-efficiency scenarios such as HVAC control, electric car energy management, or building energy management, just to mention a few. On the other hand, quantum machine learning was born during the last decade to extend classic machine learning to a quantum level. In this work, we propose to study the benefits and limitations of quantum reinforcement learning to solve energy-efficiency scenarios. As a testbed, we use existing energy-efficiency-based reinforcement learning simulators and compare classic algorithms with the quantum proposal. Results in HVAC control, electric vehicle fuel consumption, and profit optimization of electrical charging stations applications suggest that quantum neural networks are able to solve problems in reinforcement learning scenarios with better accuracy than their classical counterpart, obtaining a better cumulative reward with fewer parameters to be learned.

Keywords: quantum neural networks; variational quantum circuits; quantum reinforcement learning; energy efficiency

Citation: Andrés, E.; Cuéllar, M.P.; Navarro, G. On the Use of Quantum Reinforcement Learning in Energy-Efficiency Scenarios. *Energies* **2022**, *15*, 6034. https://doi.org/10.3390/en15166034

Academic Editor: Jaroslaw Krzywanski

Received: 27 July 2022
Accepted: 17 August 2022
Published: 19 August 2022

1. Introduction

Reinforcement learning (RL) [1] is a type of machine learning inspired by behavioral psychology. It provides the fundamental basis to describe how intelligent agents learn autonomously to take actions in unknown environments in order to maximize the notion of cumulative reward. Deep reinforcement learning (DRL) [1] integrates RL algorithms together with neural networks, and this combination has attracted great attention in the scientific literature and has been growing at an astounding speed in the last decade, due to its success in solving massively complex problems such as games (chess, Go [2], and Atari games [3]), robotics [4], or autonomous driving [5], just to mention a few. In the last few years, it has also attracted the attention of researchers in energy-efficiency methods such as [6,7], due to its ability to perform online learning and the capacity of adaptation to new changes in an environment. Neural network models used in DRL range from the multilayer perceptron or radial basis function networks, to more complex recurrent models such as Elman, Jordan, LSTM, or GRU networks, as the most sound examples.

With regards to the state of the art of RL for energy efficiency and energy management, there is a wide range of approaches, such as the efficient allocation/deallocation of the resources in response to the workload variations [8,9] or task scheduling [10,11], providing predictive and efficient routing schemes [12], or minimizing resources power usage [13]. Raman et al. [14] developed the Zap Q-learning algorithm, aimed at climate control of a commercial building. Apart from the temperature comfort, the proposal attempted to minimize energy consumption while maintaining comfort regarding humidity. As another example, Wei et al. [15] used deep Q-learning to control building automation control systems. The DQN algorithm was compared with a rule-based system and a Q-learning algorithm. Their contribution proved to be able to maintain temperature comfort in the

desired range while reducing the energy consumption. Lissa et al. [16], using the same neural network model, developed a methodology to find the bounds of indoor temperature comfort, and their proposal was able to save energy more efficiently than a rule-based system, while comfort was not affected. Finally, Wang et al. [17] proposed a reinforcement learning controller based on actor/critic models to building HVAC control, using recurrent neural networks (LSTM). Experiments were performed on a simulated environment containing a digital twin of an office building, and their results were able to save energy while maintaining the user comfort. A complete survey about RL applied for energy efficiency in buildings can be found in [18]. Despite the success of DRL that has been reported in the literature in different energy-efficiency and management scenarios, the main limitation of RL methods lies on the fact that, to be able to learn a correct behavior, the agent must interact with the environment, making good actions but also making failures until the optimal operation is learned. For this reason, researchers rely on the development of simulation environments that implement digital twins of real devices, which in some cases use historical data also. Simulation tools such as Energy Plus [19] or Modelica [20] play a fundamental role in the training and testing of models before deploying them in real energy systems. RL environments are built on top of this simulation software, for example, Sinergym [21], Gym-EPlus [22], or EnerGym [23] to model energy consumption and control in buildings, the simulation of energy management in electrical vehicles (EVs) [24,25], or the simulation of EV charging stations [26]. Different environments are described and analyzed in [21].

Contemporaneously with the advances in DRL, quantum computing (QC) [27] and quantum machine learning (QML) [28,29] in particular, have evolved rapidly as research areas thanks to the advances in quantum hardware and the creation of new quantum hardware simulators such as Google's Cirq, IBM's Qiskit, Xanadu's Pennylane, etc. The concept of quantum computing was originally presented in 1982 by Richard Feynman [30] following an observation of the exponential complexity involved in modeling the behavior of a quantum system with the existing knowledge of classical computing. It is based on quantum mechanics properties such as superposition and entanglement, which also enable quantum parallelism. The promise of QC in general, and QML in particular, is a huge speedup to obtain solutions to complex problems that could not be solved in reasonable time using classic computers. For example, early theoretical works demonstrated the massive potential of quantum algorithms solving complex problems more efficiently than their classical counterparts, such as Grover's search [31], or integer factorization using Shor's algorithm [32].

On the other hand, quantum machine learning is the intersection of machine learning and quantum computing, and it attempts to create machine learning models able to run in quantum computers with the final goal of achieving a significant speedup in their execution, or solving more complex problems with fewer resources than their classical counterpart. Quantum reinforcement learning (QRL) is a very recent research field, and the lack of worldwide access to a real quantum hardware able to run the learning process forces researchers to use quantum computer simulators running in classical computers. We believe that, for that reason, the existing QRL research works in the literature, such as [33–36], focus their experimentation on well-tested simple environments, mainly from the OpenAI Gym RL platform [37]. Despite these limitations, the existing QRL approaches have provided relevant insights into how RL can be adopted by a quantum system, such as the use of Grover iterations [33] to provide a quantum implementation of the classic Q-learning algorithm, or variational quantum circuits (VQC) [38] as quantum agents in deep Q-learning or policy gradient methods [34–36]. The VQC is a class of hybrid classical–quantum algorithm based on a quantum circuit with learnable parameters optimized using classical algorithms so that, in the case of QRL, VQCs are used as function approximators for the agents' policy. The trainable parameters would be optimized using gradient or gradient-free methods.

In this manuscript, we apply QRL to solve problems in the field of energy efficiency and energy management using quantum artificial neural networks. We hypothesize that QRL technologies could solve the problems using a more compact model than their classical counterparts thanks to inherent nonlinear data management and quantum parallelization. To the best of our knowledge, this article is the first attempt to address energy-efficiency problems with quantum technologies. For that reason, we study different types of use cases where classical RL has been successfully applied before: the Sinergym [21] project, which is a simulator backed up by EnergyPlus and contains reference problems of energy efficiency and HVAC control in buildings and facilities; the Prius [24] simulator of electrical vehicles, aimed at reducing the fuel and battery usage of electrical vehicles with the Toyota Prius car as testbed; and a simulator of electrical vehicle charging stations [26] whose objective is to maximize the station's profit in a real scenario, considering the workload and the price of energy. Thus, the main contributions of the article can be summarized as follows: (1) the proposal of QRL methodology for energy-efficiency and management scenarios; (2) the design of hybrid quantum/classical agents to solve QRL energy-related environments; and (3) a wide experimental study with an analysis of pros and cons of contemporary QRL technologies in the field of energy efficiency and management.

The remainder of the manuscript is structured as follows: Section 2 provides a background in reinforcement learning and quantum computing. After that, Section 3 explains the approach to build hybrid quantum/classical QRL agents. Section 4 describes the experimentation and the analysis of results, and finally Section 5 provides conclusions.

2. Background

For article self-completeness and notation description, this section summarizes the main concepts required to describe the proposal of the manuscript in Section 3. Firstly, Section 2.1 describes the fundamentals of artificial neural networks. Then, Section 2.2 introduces the field of reinforcement learning and its integration with (deep) artificial neural networks. Finally, Section 2.3 provides an introduction to quantum computing and quantum variational circuits methods.

2.1. Artificial Neural Networks

Artificial neural networks [39] are computational models inspired by connectionist models of human and animal brains. A *feedforward* neural network is formalized as a weighted, non-cyclic, directed graph where nodes represent computation units called *neurons*, and edges are *synapses*. The most known neural network model is the *multilayer perceptron* (MLP), where neurons are stacked in layers, and its graphical representation is shown in Figure 1.

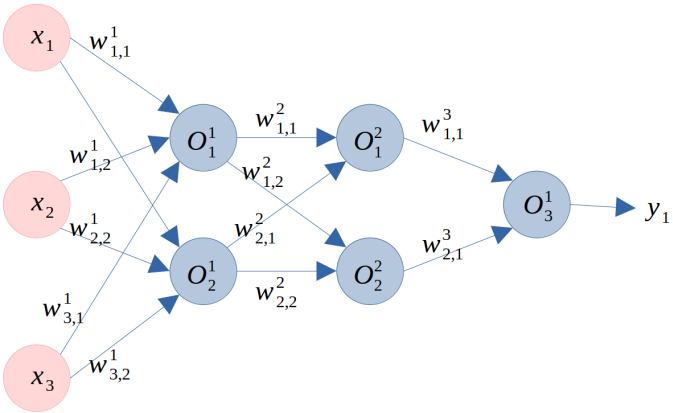


Figure 1. Example of a multilayer perceptron with 3 inputs, 2 layers with 2 neurons each, and 1 output.

The dynamic of an MLP is as follows: Assuming an input data pattern $O^0 = X$ composed of N_0 values, $X = (x_1, x_2, \ldots, x_{N_0})$, for which an output response $Y = (y_1, y_2, \ldots)$

is expected, each neuron i at layer l calculates its output O_i^l as a function f of the weighted sum of its inputs (Equation (1)), where $w_{ji} \in \mathbb{R}$ represents the synaptic strength (weight) of the connection between output of neuron j at layer $l-1$ to neuron i at layer l; $b_i^l \in \mathbb{R}$ is a neuron intercept also called *bias*, and f is the neuron activation function. The network output Y corresponds to the returned values of neurons located at the last layer. Usual activation functions are sigmoid, hyperbolic tangent, rectified linear units (ReLUs), etc. In this work, we use MLP as baseline models to compare quantum neural networks behavior in the experimentation. The ReLU activation function (Equation (2)) is used as activation function.

$$O_i^l = f\left(\sum_{j=1}^{N_{l-1}} w_{ji}^l O_j^{l-1} + b_i^l\right) \tag{1}$$

$$f(x) = \begin{cases} 0 & x < 0 \\ x & x \geq 0 \end{cases} \tag{2}$$

Training a neural network encompasses finding the correct values for weights w_{ij}^l and biases b_i^l so that the network output $\tilde{Y}$ matches a desired output Y. This problem is usually formulated as a minimization problem over a defined cost function, also called *loss*, for instance, in Equation (3), where $Y(p)$ is the desired output for a given input pattern $X(p)$, and $\tilde{Y}(p)$ is the output provided by the network for pattern $X(p)$. Different algorithms such as backpropagation, gradient descent, ADAM, etc., are used to optimize both weights and biases. For more information about neural networks, we believe that the reference [39] is a good, detailed book for both novicesand experts in the field.

$$[w_{ij}^{l\,*}, b_i^{l*}] = argmin_{\{w_{ij}^l, b_i\}} \left\{ \sum_p (\tilde{Y}(p) - Y(p)) \right\} \tag{3}$$

2.2. Reinforcement Learning

The main model of RL encompasses two main components: an environment (whose internal dynamics are unknown) and an intelligent agent that interacts with the environment over time. At any time instant, the environment is at an unknown state, but the agent can *observe* the environment to obtain some information about such state, e.g., using sensors. For the sake of simplicity, in this manuscript we name as s_t the environment observation perceived by the agent at time t. Then, the agent selects an action a_t from an available action set considering the observed state s_t, and performs the action over the environment. After that, the environment evolves from state s_t to a state s_{t+1} as a consequence of the agent–environment interaction, and returns s_{t+1} and a reward r_{t+1} to the agent. This reward is a scalar value that informs the agent about how good/bad the selection was of action a_t conditioned to state s_t. This process is repeated in a loop indefinitely (Figure 2).

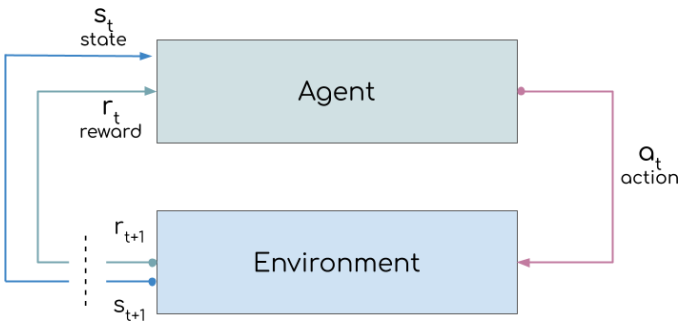


Figure 2. General reinforcement learning diagram.

The process of sequential agent–environment interactions (also known as trajectories or episodes) involving state transitions, actions, and rewards in the RL problem can be formulated as a Markov decision process (MDP). An MDP is characterized by the tuple

$\langle S, A, P, r \rangle$, where S is the set of states, A is the set of actions, P is the probability of state transition $P(s'|s,a) = P[s_{t+1} = s'|s_t = s, a_t = a]$; i.e., the probability of transitioning from the state s_t at time t to state s_{t+1} at time $t+1$ using action a_t selected at time t, and $r(s_t, a_t, s_{t+1})$ is the reward function for executing action a_t at state s_t and then performing a transition from s_t to s_{t+1}.

The goal of an agent in RL is to maximize the total cumulative reward, or *return*, of a sequence of agent–environment interactions τ starting at time t_0 defined in Equation (4), where γ is a hyperparameter to establish how important recent or older rewards are for the learning, known as *discount factor*.

$$R(\tau) = \sum_{t=t_0}^{\infty} r_t \gamma^{t-t_0} \tag{4}$$

In order to maximize $R(\tau)$, it is necessary that the agent learns the best action a to be performed at a given state s, i.e., the *policy* $\pi(a|s)$, which is the probability to select action a in state s. Two fundamental concepts arise from this problem formulation: the value of a state–action pair $Q(s,a)$ in Equation (5) (the expected return obtained in the trajectory starting from state s and action a), and the value of a state $V(s)$ in Equation (6) (the expected return obtained in the trajectory starting from state s). A third relevant concept is the advantage of the state–actionpair $Adv(s,a)$ (Equation (7)), which returns the advantage of choosing action a in state s with respect to the other actions available in the action set for the same state s.

$$Q(s,a) = \mathbb{E}_{\tau\sim\pi}(R(\tau)|s_t = s, a_t = a) = \textstyle\sum_{s'} p(s'|s,a)(r(s,a,s') + \gamma \sum_{a'} \pi(a'|s')Q(s',a')) \tag{5}$$

$$V(s) = \mathbb{E}_{\tau\sim\pi}(R(\tau)|s_t = s) = \sum_i R(\tau_i)\pi(a_i|s) \tag{6}$$

$$Adv(s,a) = Q(s,a) - V(s) \tag{7}$$

Deep reinforcement learning attempts to use a (deep) artificial neural network to learn the optimal policy $\pi(a|s)$, having the state s as input and, as output, a selected action a (deterministic policy), or the probability distribution of selection of action a in the state s (stochastic policy). In the literature, two main families of algorithms have been highlighted in the last few years: deep Q-networks (DQN) [40] and policy gradient [41]. The former family attempts to train an artificial neural network to approximate the function $Q(s,a)$, while the latter attempts to approximate $\pi(a|s)$ directly. DQN training is inspired by the classic Q-learning method, and attempts to minimize the loss function in Equation (8), where $\hat{Q}(s,a)$ is the a-th output value of the neural network for the input s. A deep Q-network contains as many input neurons as the dimension of the state, and as many output neurons as actions available in the action set.

On the other hand, actor–critic policy gradient models require at least two types of neural networks for training: one type of network (actor) has a similar structure to a DQN, but the a-th network output is designed to return $\pi(a|s)$. The second type of network attempts to return an approximation of $V(s)$, so that it contains the same number of inputs as the actor, and a single output value.

$$MSE = \sum_t \left(r(s_t, a_t, s_{t+1}) + \gamma max_{a_{t+1}}\left[\hat{Q}(s_{t+1}, a_{t+1})\right] - \hat{Q}(s_t, a_t)\right)^2 \tag{8}$$

Different strategies have been devised to improve DQN training and policy gradient methods and, for further reading, we refer the reader to references [40,41]. In this work, we use the Advantage Actor–Critic (A2C) training algorithm [41], whose designed loss function is described in Equation (9).

$$LogLoss = -\sum_t Adv(s_t, a_t) log\pi(a_t|s_t) \tag{9}$$

2.3. Quantum Computing and Variational Quantum Circuits

The most basic information unit in classic computing is the bit, and the underlying mathematical model to work with bits is $\mathbb{Z}_2$, the field with two elements. Unlike the classic model, the fundamental information unit in quantum computing is the qubit, and the underlying mathematical model is the complex plane $\mathbb{C}^2$, where the standard orthonormal basis vectors are usually denoted by $|0\rangle$ and $|1\rangle$. Hence, the value (state) of a qubit $|\psi\rangle$ is a linear combination of these vectors:

$$|\psi\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle ; \alpha_0, \alpha_1 \in \mathbb{C} \tag{10}$$

with the constraint that $|\alpha_0|^2 + |\alpha_1|^2 = 1$, since Born's rule tells us that if this qubit is measured, we will obtain $|0\rangle$ with probability $|\alpha_0|^2$ and $|1\rangle$ with probability $|\alpha_1|^2$.

When a new bit is included into a classical system, the underlying state space increases by one dimension, since the resultant space is the Cartesian product of the older one by $\mathbb{Z}_2$. Thus, in n-bit systems, the state space is $\times^n \mathbb{Z}_2 = \mathbb{Z}_2^n$. On the other hand, when a new qubit is added to a quantum system, the underlying space doubles its dimension, since the resultant space is the tensor product of the older one by $\mathbb{C}^2$, yielding an exponential growth of the dimension. In an n-qubit system, the underlying model is $\mathbb{C}^{2^n} = \bigotimes^n \mathbb{C}^2$, i.e., the tensor product of $\mathbb{C}^2$, n times. For instance, the $\mathbb{C}$-vector space $\mathbb{C}^{2^2}$ has a basis formed by the vectors $|0\rangle \otimes |0\rangle$, $|0\rangle \otimes |1\rangle$, $|1\rangle \otimes |0\rangle$, and $|1\rangle \otimes |1\rangle$, which, for simplicity, are denoted by $|00\rangle$, $|01\rangle$, $|10\rangle$, and $|11\rangle$, respectively.

Classical computers manipulate bits using logic gates such as AND, OR, NOT, NAND, XOR, etc. Likewise, quantum computers manipulate qubits using quantum gates. A quantum gate is modeled as a unitary matrix that multiplies the system qubit state, and all quantum operations must be reversible (except measurement). Some examples of quantum gates are the Pauli-X (σ_x), Pauli-Y (σ_y), and Pauli-Z (σ_z) single qubit rotations with a phase of π, or their generalization rotation gates $R_x(\theta)$, $R_y(\theta)$, and $R_z(\theta)$ that rotate a qubit in the specified axis by the given angle θ. Another relevant quantum gate for the proposal described in this work is the CNOT gate over two qubits. The unitary matrices of the gates $R_x(\theta), R_y(\theta), and R_z(\theta), CNOT$ are shown in Equations (11), (12), (13), and (14), respectively.

$$R_x(\theta) = \begin{pmatrix} cos(\frac{\theta}{2}) & -isin(\frac{\theta}{2}) \\ -isin(\frac{\theta}{2}) & cos(\frac{\theta}{2}) \end{pmatrix} \tag{11}$$

$$R_y(\theta) = \begin{pmatrix} cos(\frac{\theta}{2}) & -sin(\frac{\theta}{2}) \\ sin(\frac{\theta}{2}) & cos(\frac{\theta}{2}) \end{pmatrix} \tag{12}$$

$$R_z(\theta) = \begin{pmatrix} e^{-i\theta/2} & 0 \\ 0 & e^{i\theta/2} \end{pmatrix} \tag{13}$$

$$CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{14}$$

A *quantum program*, implemented as a *quantum circuit*, is a sequence of quantum gates applied over one or more qubits. An example of a graphical representation of a quantum circuit with two qubits is shown in Figure 3: First, two single-qubit Hadamard (H) quantum gates are applied separately over qubits q_0 and q_1. Then, a two-qubit SWAP quantum gate is applied on both qubits, followed by a CNOT gate over q_1 controlled by q_0. Finally, both qubits are measured to obtain the result, which can be any quantum state in $\{|00\rangle, |01\rangle, |10\rangle, |11\rangle\}$.

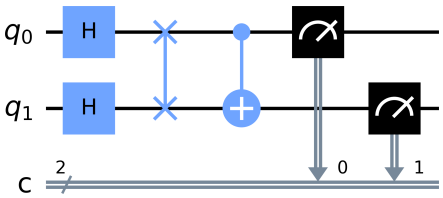


Figure 3. Example of a quantum circuit with two qubits.

A variational quantum circuit is a quantum circuit containing parameterized gates, such as the aforementioned R_x, R_y, R_z gates with parameter θ. Since VQCs are universal approximators and are able to approximate any continuous function similar to a classical neural network [34,42], they are also known in the quantum machine learning community as quantum neural networks. An example of a VQC is shown in Figure 4, where three rotations R_x, R_y, R_z gates are applied over each qubit (12 total parameters), and then different non-parameterized gates transfer the qubit information. If the reader is interested in extending the fundamentals of QC and QML, we suggest the references [27,28].

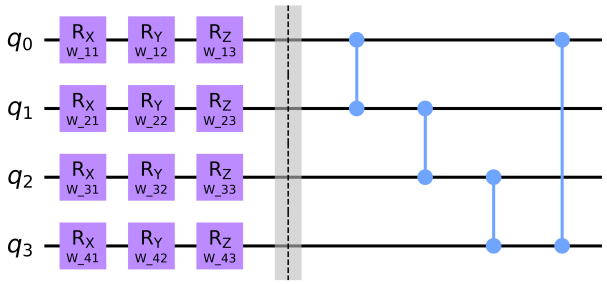


Figure 4. Example of a variational quantum circuit with 4 qubits.

The output of a quantum system is obtained using the *measurement* operator, which makes a qubit collapse to a basis vector $|0\rangle$ or $|1\rangle$. In quantum computing hardware it is required to measure outputs with regards to an *observable*, i.e., quantum properties that can be observed. It is also common to measure expectations of these observables [27], and this last technique is the one we use in our experimentation. The expectation of an observable whose unitary matrix is U, over a quantum state $|\phi\rangle$, is denoted as $\langle\phi| U |\phi\rangle$. In our case, we use the Z observable whose unitary matrix σ_z is the Pauli-Z shown in Equation (15). As an example, if the quantum state is $|\phi\rangle = |0\rangle$, then the expectation of the observable returns value 1, and if the quantum state is $|\phi\rangle = |1\rangle$, then the expectation of the observable returns value -1. Any other intermediate value in $[-1, 1]$ is linked to the probabilities α_0 and α_1 described previously.

$$\sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \tag{15}$$

3. Methodology

We assume a QRL scenario of a hybrid quantum–classical agent with a set of parameters $\theta = (\theta_1, ..., \theta_k)$ that learns by interacting with a classical (non-quantum) environment, as shown in Figure 5. In the beginning of an episode, the quantum–classical hybrid agent receives a state information from the environment, determines its action from its policy $\pi_\theta(a|s)$, and perceives a reward r and the next environment state after applying the action to the environment. The policy π_θ is given by a VQC running on a quantum processing unit (QPU). The optimization of this policy is carried out by updating the parameters θ, which is performed by a classical learning algorithm in a central processing unit (CPU) with the objective of reducing the cost function [43]. In our work, the cost function is described in Equation (9) for A2C (actor network) and Equation (16) for the critic network, where

$V^\tau(s)$ is the true value function of state s in a given trajectory τ, and $\hat{V}^\tau(s)$ is the value network output.

$$MSELoss = \sum_{\tau} \sum_{s \in \tau} \left(\hat{V}^\tau(s) - V^\tau(s) \right)^2 \tag{16}$$

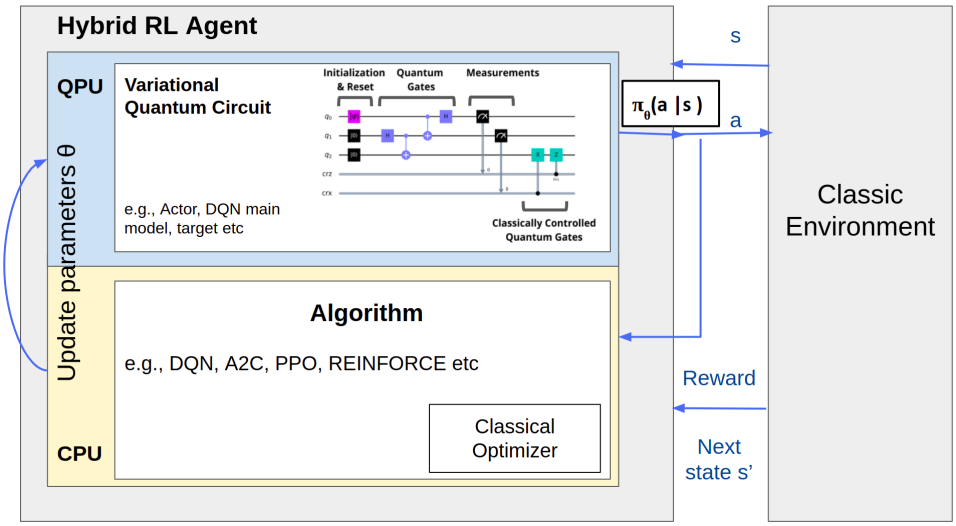


Figure 5. QRL schema. Example of hybrid agent and classical/quantum interaction to optimize the trainable parameters of the policy using a classical learning algorithm.

The general hybrid methodology to train a quantum agent in QRL is illustrated in Figure 6, and encompasses the following steps:

1. **Preprocessing (CPU)**: This step has the environment state s as input, and performs data preprocessing to prepare classical data before encoding the environment state into a quantum state. This preprocessing could be a simple operation of translation and/or change of scale (Figure 7), or more complex data processing such as a change of space (projection, injection) with a classical (non-quantum) neural network layer (Figure 8).
2. **Quantum Embedding (QPU)**: The objective of the encoding layer, also named quantum embedding, is to map the classic data to the qubits' Hilbert space to be manipulated afterward. This layer is composed of rotation and entanglements gates, and the tensor product and the entanglement generated in those encoding circuits capture the existing nonlinearities of the data. The information of the environment state s, after the previous preprocessing step, must be encoded into a quantum state $|s\rangle$ by means of a quantum circuit with the appropriate sequence of quantum gates. In this work, we use the tensor product encoding technique [44] using $R_x(\theta)$ rotation gates, so that each feature of state s is matched with a qubit. The value of the feature is the rotation angle of the R_x gate. Thus, we must ensure that all features in s are in the range $[-\pi/2, \pi/2]$, which is the domain of the parameter of $R_x(\theta)$.
3. **Variational Layer (QPU)**: The quantum state $|s\rangle$ is the input of the VQC in charge of modeling $\pi_\theta(a|s)$ as a quantum state $|\phi\rangle$. Usually, VQCs are composed of entanglement and rotation gates parameterized with learnable parameters θ. A classic optimization algorithm will optimize these parameters, aiming at minimizing a given cost function. The choice of the encoding strategy, as well as the construction of the VQC, are crucial to obtain optimal results. In this work we use the same VQC in all experiments, and it is inspired by previous works [35,43].
4. **Measurement Process (QPU/CPU)**: In this final step, the quantum state provided by the VQC is measured and decoded to obtain the desired output. The critical point in the quantum measurement is to find an optimal way to associate outputs of the observations with target classes. The selection of the observables used to read out

information from the quantum model is crucial to achieve a good performance. In this work, we calculate as output the expectation of the σ_z observable, i.e., the operation $\langle\phi|\,\sigma_z\,|\phi\rangle$, for each qubit. The value of this observable is in the range $[-1, +1]$, where the bounds $+1$ means that ket $|0\rangle$ is always returned, and -1 means that ket $|1\rangle$ is always returned as output.

5. **Postprocessing (CPU)**. This step gathers the outputs returned by the QPU, i.e., the policy $\pi_\theta(a|s)$, and performs post-processing operations over the outputs, if necessary.
6. **Learning (CPU)**. This last step computes the loss function of the VQC to optimize its parameters θ. In general QRL approaches, and also in this work, this is performed by a traditional optimization algorithm such as Adam, RMSProp, etc.

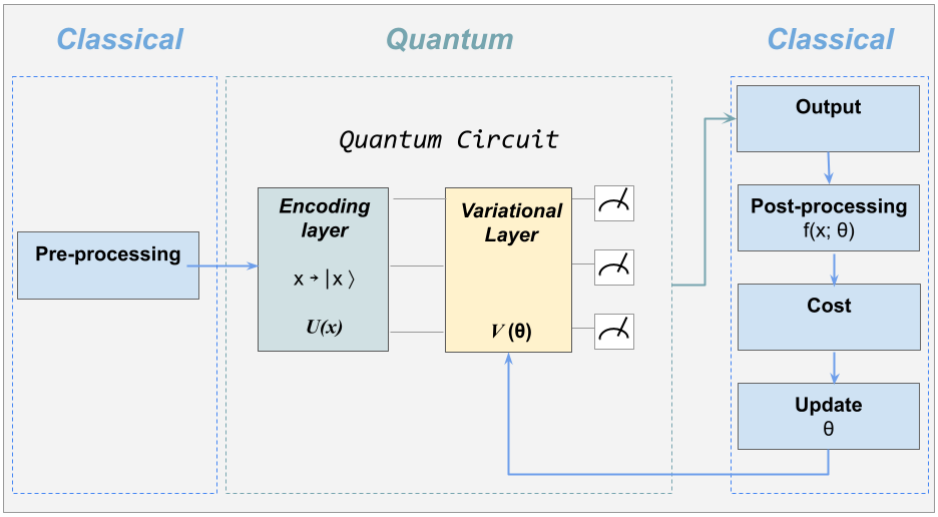


Figure 6. General hybrid quantum–classical agent structure.

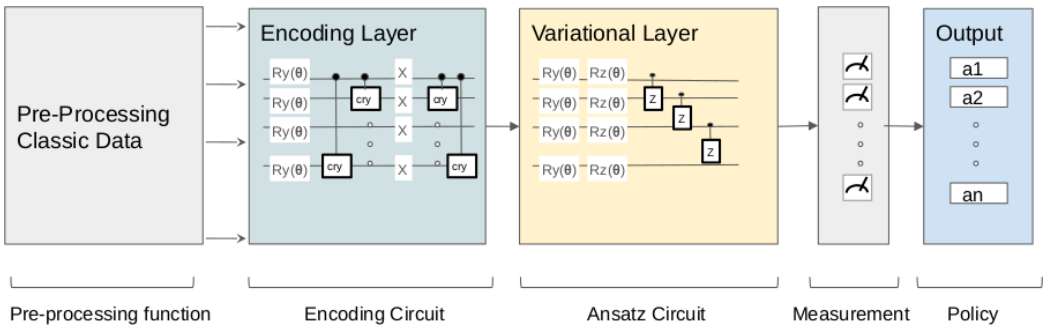


Figure 7. Preprocessing with input data manipulation. The output of preprocessing is fed to the encoding circuit and then to the variational layers.

As happens in classical neural networks, VQCs also suffer from the gradient vanishing phenomenon, also known as barren plateau [38,45]. In general, a VQC is able to solve a problem using fewer parameters than its classical counterpart, although there is an inflection point at which the excessive number of qubits, entanglements, or layers used in the construction of the circuit makes this phenomenon more prone to happen. For the case study of Section 4, we selected a circuit made up of R_y, R_z rotation gates, followed by entanglement gates similar to circuits used in other previous QRL algorithms that provided promising results in simple RL problems [35,43]. Figure 9 shows an example of the VQC structure used in this work for a quantum program containing 7 qubits. A multilayer VQC is generated in our experiments by means of creating a sequence of VQCs with this structure. Thus, we may see that the number of learnable parameters in a VQC increases linearly, which is an advantage when compared with classical (non-quantum) neural networks.

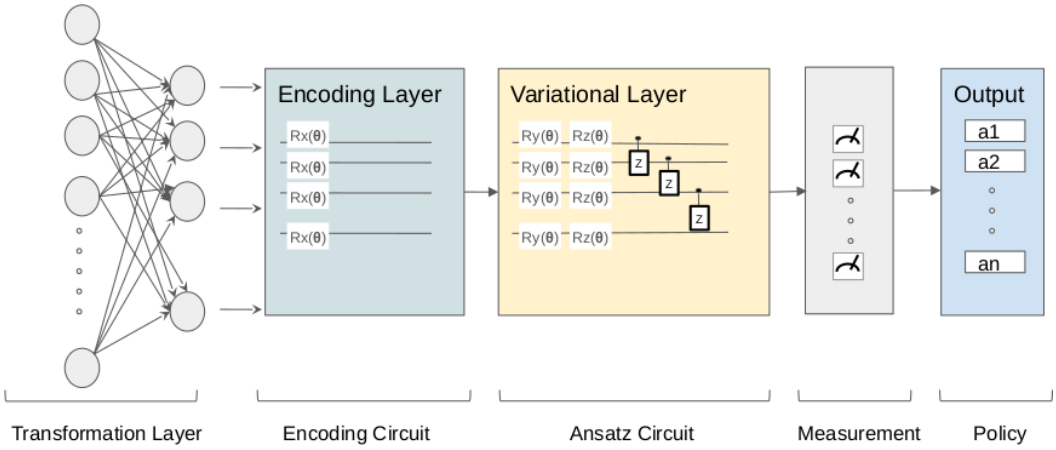


Figure 8. Preprocessing with a hybrid classical/quantum neural network. The output of preprocessing is fed to the encoding circuit and then to the variational layers.

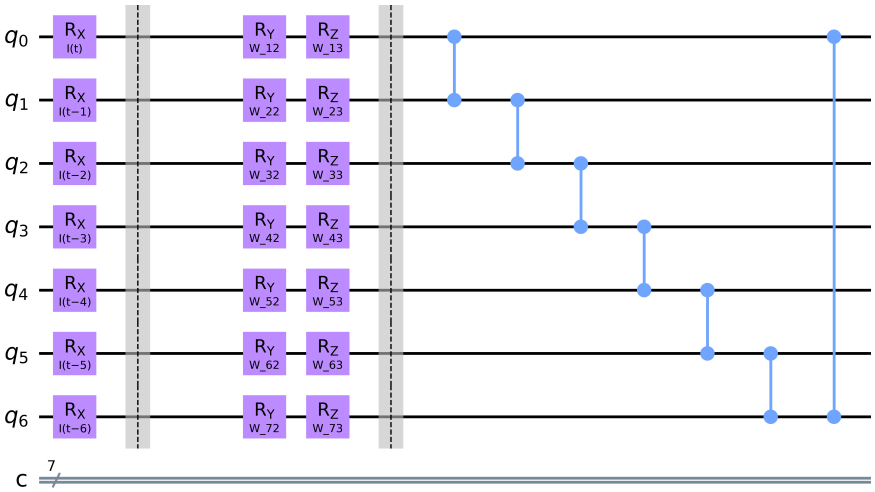


Figure 9. Example of a 7-qubit VQC with the structure used in the experiments. Left: encoding layer; middle: variational gates; right: entanglement CNOT gates organized in a ring.

4. Experiments

In this section, we test quantum reinforcement learning algorithms in different scenarios related to energy efficiency, and compare the results with their corresponding classical counterparts. The goal is to verify if QRL and quantum technologies are appropriate in the field of energy efficiency. To that end, three different use cases with different constraints and nature are used. Section 4.1 addresses a use case of classic control of HVAC energy-saving in buildings; Section 4.2 solves an environment regarding energy management strategies in hybrid electric vehicles; and, finally, Section 4.3 addresses a mixed environment regarding optimal control of electric vehicle charging stations.

To avoid biases and to favor baseline comparison and analysis of results among different types of problems, the ground metric used in the analysis is the cumulative reward (return) of episodes, which eases comparison between different types of problems and environments. The QRL models are compared with state-of-the-art baseline approaches, which are classic actor/critic multilayer perceptron networks whose structure can be easily adapted for the three use cases. Both classic RL and QRL models are trained using the same A2C algorithm, to avoid training biases between quantum and classical models.

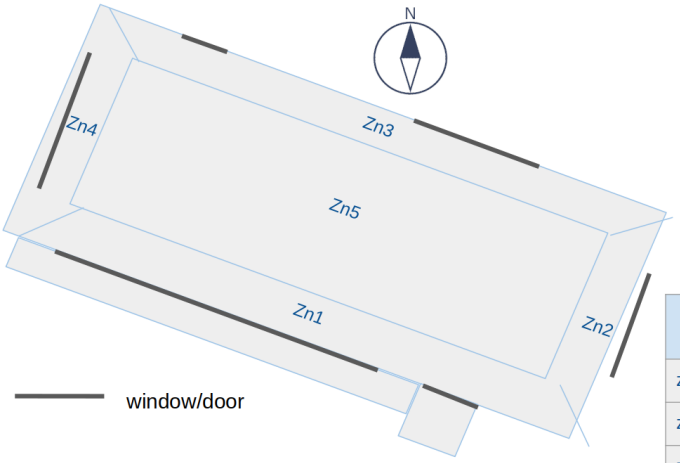
With the previous decisions, we are able to compare QRL with classic RL independently of the use case for their peculiarities, therefore simplifying the global analysis of results among different types of problems.

4.1. Use Case 1: HVAC Control

4.1.1. Problem Statement

This use case is powered by the *Sinergym* [21] simulator. The target building in this study is the environment *Eplus-demo-v1* and focuses on the scenario *5ZoneAutoDx* [21], Pittsburgh, PA, USA. It is a single-floor rectangular building 100 ft × 50 ft with five zones (four exterior, one interior) and all zones were regularly occupied by office workers. The overall building height is 10 feet. There are windows on all four facades; the south and north facades have glass doors. The building is oriented 30 degrees east of north (see Figure 10). The walls are woodshingle over plywood, R11 insulation, and Gypboard. The roof is a gravel built-up roof with R-3 mineral board insulation and plywood sheathing. The windows are of various single- and double-pane construction with 3 mm and 6 mm glass and either 6 mm or 13 mm argon or air gap. The window-to-wall ratio is approximately 0.29. The south wall and door have overhangs. Finally, the floor area is 463.6 m^2 (5000 ft^2). A state s in the reinforcement learning environment represents a sequence of historical observations (e.g., outdoor air temperature and room temperatures) from the buildings. The goal of this environment is to maximize an aggregation of KPIs (key performance indicators) regarding energy usage and user comfort. The selected environment has the following features:

- **State space:** The state space contains 20 features; 16 are described in Table 1 and 4 are reserved for the environment.
- **Action space:** The action space contains a set of 10 discrete actions described in Table 2. The bounds for heating setpoint temperature is [15, 22.5] and for cooling setpoint temperature is [21, 30].
- **Reward function:** Reward is always negative. This means that perfect reward would be 0 (perfect power consumption and perfect temperature comfort); there are two temperature comfort ranges (winter and summer) and a weight of energy and comfort. Therefore, the reward is calculated as the sum of reward of energy and comfort multiplied by their respective relevance weights, given by W_{energy}, as a hyperparameter.



	Area (m²)	Hight (m)
zn1	100	2.4
zn2	43	2.4
zn3	100	2.4
zn4	43	2.4
zn5	186	2.4

Figure 10. Zone building plan.

$$reward = W_{energy} * reward_{energy} + (1.0 - W_{energy}) * reward_{comfort} \quad (17)$$

Table 1. Observation variables.

Variable Name	Units
Site Outdoor Air Drybulb Temperature	°C
Site Outdoor Air Relative Humidity	%
Site Wind Speed	m/s
Site Wind Direction	° N
Site Diffuse Solar Radiation Rate per Area	W/m^2
Site Direct Solar Radiation Rate per Area	W/m^2
Zone Thermostat Heating Setpoint Temperature	°C
Zone Thermostat Cooling Setpoint Temperature	°C
Zone Air Temperature	°C
Zone Thermal Comfort Mean Radiant Temperature	°C
Zone Air Relative Humidity	%
Zone Thermal Comfort Clothing Value	Icl (clo)
Zone Thermal Comfort Fanger Model PPD	%
Zone People Occupant Count	count
People Air Temperature	°C
Facility Total HVAC Electricity Demand Rate	W

Table 2. Action variables.

Variable Name	Heating Setpoint	Cooling Setpoint
0	15	30
1	16	29
2	17	28
3	18	27
4	19	26
5	20	25
6	21	24
7	22	23
8	22	22
9	21	21

In our experimentation we design a classic multilayer perceptron (MLP) neural network agent with 20 inputs (environment state) and 10 outputs and train the agent in the *Eplus-demo-v1* environment with the A2C algorithm [41]. After that, we also train a quantum agent designed with the methodology described in Section 3 and trained with the same A2C algorithm. The settings for the environment and A2C algorithm remain the same for the classic baseline and quantum agents, to make a fair performance comparison.

For the actions' control, the proposed framework provides continuous and discrete action space, and the RL agent controllers were trained and evaluated using a discrete action space environment.

4.1.2. Experimental Settings

Two types of experiments were performed: The first one models the agent's policy with a classic feedforward multilayer perceptron (MLP) neural network; the second one implements the agent's policy as a hybrid architecture with one linear layer and one variational quantum circuit. In these experiments we used the advanced actor critic RL method to train the agent. To that end, we ran four environments in parallel to generate batches of five interactions between the agent and each environment, using a discount factor $\gamma = 0.98$ to calculate the total return of each episode. The stopping criterion was set so that the algorithm stops after 100 episodes were finished. Additionally, the agent performance was tested in the last episode with a deterministic policy that selected the action with maximum probability to check how the agent learned to interact with the environment. Finally, we performed 10 different runs with different initial random seeds to validate the experimentation.

The algorithm A2C requires two networks to perform the learning: a policy network (actor) which inputs a state s and outputs $\pi(a|s)$, i.e., the probability of selection of action a in state s, and a value network (critic) which inputs a state s and outputs an estimation of the value of the state $V(s)$. The structure of the policy and value networks for the classical and the quantum agents are described in Table 3. Different stages of information processing can be distinguished in Table 3: firstly, the quantum agents define the use of 10 qubits (one for each possible action selection). Then, the information of a state of the environment (20 values) is provided to each network (in the case of the quantum VQCs, a scale to the range $[-\pi/2, \pi/2]$ is required prior to the quantum embedding process). After that, the intermediate network layers of the MLP agent contain 50 units that compute a linear combination of the previous inputs followed by an ReLU activation function. In the case of the quantum agent, the encoded information of the input layer is fed to five sequential quantum variational layers, as explained in Section 3. The last step in the policy MLP network calculates the probability of selection of each action using a linear layer containing 10 outputs using a softmax activation function, and the policy VQC agent measures the expectation of the σ_z operator over all qubits separately, followed by a softmax activation function to transform the measured data into a probability distribution.

In the case of the policy MLP (actor), the number of parameters is $20 \times 150 + 150 \times 10$ (weights) + 150 + 10 (bias) = 4660; and the number of parameters of the value network (critic) is $20 \times 150 + 150 \times 1$ (weights) + 150 + 1 (bias) = 3301. If we focus on the policy VQC, each qubit of every quantum variational layer contains two trainable parameters: one for qubit rotation in the y axis, and one for rotation in the z axis. The value network requires two additional parameters to perform the output scale and translation. Then, the total number of parameters is 300 for the policy network: 20×10 = 200 (preprocessing layer) + 5 (number of layers) $\times$ 2 (learning parameters used in Rx and Ry) $\times$ 10 (number of qubits). For the value VQC network, the number of parameters is $20 \times 10 + 5 \times 2 \times 10 + 2 \times 10 = 320$. We remark that the number of total parameters required by the quantum proposal is much lower than the number of parameters required to train the classical agent so that, in theory, the complexity of the quantum agent is lower than the complexity of its classical counterpart.

Table 3. Structure of policy and value classical/quantum networks used in use case 1.

	Policy MLP	**Value MLP**	**Policy VQC**	**Value VQC**
No. of qubits	-	-	10	10
Inputs	20	20	20. Scaled to $[-\pi/2, \pi/2]$	20. Scaled to $[-\pi/2, \pi/2]$
Preprocessing	-	-	Linear (10 units)	Linear (10 units)
	-	-	Tanh	Tanh
Quantum embedding	-	-	Tensor product encoding	Tensor product encoding
Intermediate Layers	Linear (150 units)	Linear (150 units)	Variational Layer (10 qubits)	Variational Layer (10 qubits)
	ReLU	ReLU	-	-
Output layer	Linear (10 units)	Linear (1 unit)	Expectation of $\sigma_z(q_i)$ observable for each qubit q_i	Product of expectation of $\sigma_z(q_i)$ observable for all qubits q_i
Post-processing	softmax activation	-	softmax activation	Scale and translation
Training algorithm	RMSProp ($\lambda = 0.001$)	RMSProp ($\lambda = 0.03$)	RMSProp ($\lambda = 0.01$)	RMSProp ($\lambda = 0.03$)

4.1.3. Results

The performance results are shown in Table 4 summarizing the average, best, and worst total accumulated reward obtained by the MLP and the VQC, as well as the computational time required to finish each experiment (in seconds) and the validation using a deterministic policy that selects the action with maximum probability. The results consider the total accumulated reward of the agent in an episode, i.e., the sum of the reward function described in Equation (17) in all steps of an episode. We refer to the average accuracy as the average total accumulated reward in the 10 different runs performed.

The first thing we notice in the results shown in Table 4 is that the quantum agent achieves the best result in average, in the best and in the worst total reward value, although

the computation time is higher for VQC because of the use of simulators instead of real quantum computers. The boxplots of the total reward distribution for both VQC and MLP are shown in Figure 11. We remark on the high standard deviation of the classical agents during the whole process of learning, and this fact suggests that the MLP can achieve suitable results, but it depends highly on the initial parameter values (weights and biases). It also makes the classical MLP less robust than the quantum VQC.

To conclude with the analysis of results, Figure 12 plots the learning curves of both quantum VQC and classical MLP agents during the learning process from episode 1 to episode 100. We can observe the better learning in the case of VQC and its higher standard deviation of total reward among episodes of different runs in the first episodes of learning than in the last learning episodes, when it has converged, as should be expected. However, the standard deviation of the total reward in the classical agent remains higher during the whole process of learning, as mentioned above. In addition, the classic MLP becomes trapped in a local optima which does not reach the optimal total reward; therefore, it is not able to learn the optimal policy.

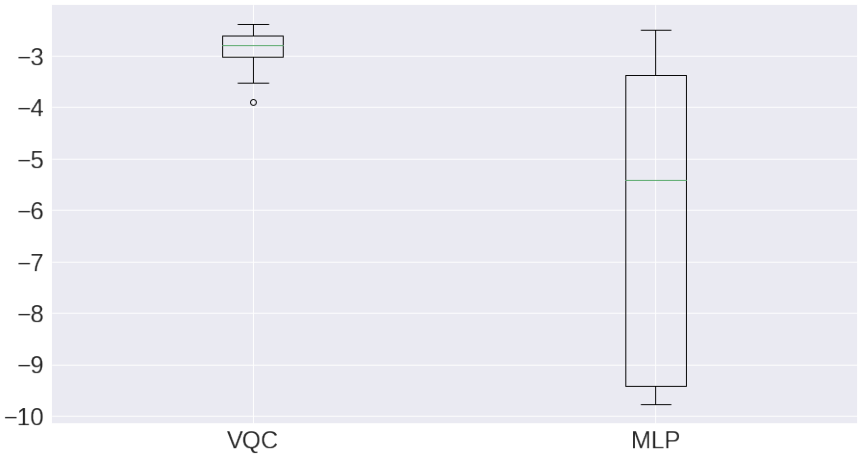


Figure 11. Boxplots of average total reward obtained from VQC and MLP in use case 1.

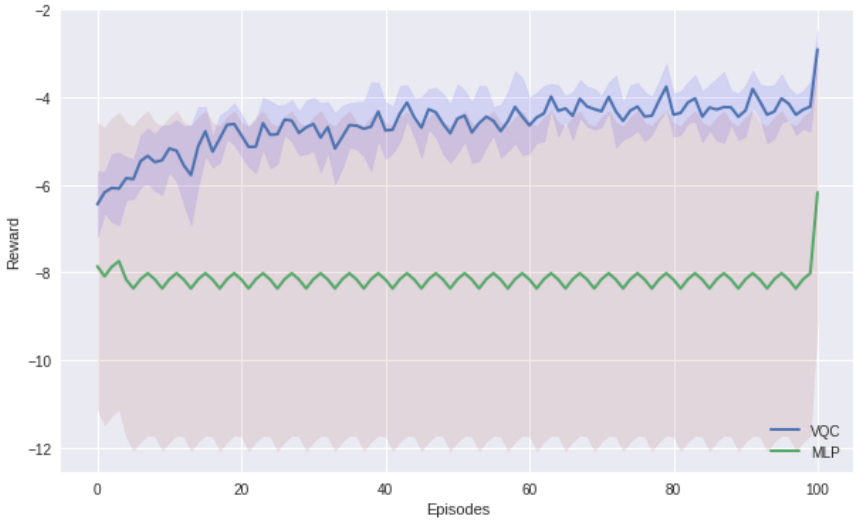


Figure 12. Learning curve obtained from VQC and MLP in use case 1.

Table 4. Results obtained by MLP and VQC agents in the use case 1 environment. Column 1: MLP agent (classic) and VQC agent (quantum); Column 2: Average total reward after training; Column 3: Best total reward after training; Column 4: Worst total reward after training; Column 5: Computational time in seconds; Column 6: Total reward (test with deterministic policy).

Model	Avg. Total Reward	Best Total Reward	Worst Total Reward	Time (s)	Test Reward
VQC (quantum)	−2.91	−2.38	−3.90	1365.46	0.0
MLP (Classic)	−6.17	−2.50	−9.77	737.93	0.0

4.2. Use Case 2: Energy Management in Electrical Vehicles

4.2.1. Problem Statement

The environment tested in this use case was first described in [24], and the authors used deep reinforcement learning (DRL) to improve the energy management strategy in hybrid electrical vehicles. More specifically, they developed algorithms based on the deep deterministic policy gradient method to learn optimal energy management policies for the Toyota Prius car. As described in [24], the vehicle is equipped with a gasoline engine, a traction motor, a generator, and a small Ni-MH battery. The authors gathered real data measured experimentally, and they used these data to create a simplified simulation reinforcement learning environment with the following features:

- **State space:** The state space contains three features *(SoC, velocity, and acceleration)*. The *state of charge (SoC)* refers to the battery state, while the other features *velocity, acceleration* refer to the current state of the vehicle.
- **Action space:** The action space contains a single continuous action, *engine power*, which is a real value that describes the power that the engine must supply. The authors also provided a discretized simulation of the action space, containing 14 different actions that update the required engine power by increasing/decreasing the power by +0 kW, ±1 kW, ±2 kW, ±4 kW, ±6 kW, ±8 kW, and ±10 kW (13 actions), plus an additional action that resets the engine power to value 0 kW. In our experiments, we use the discretized action space.
- **Reward function:** The work [24] attempts to make multi-objective optimization of both the fuel and the battery required by the vehicle. Equation (18) describes the aggregated reward function which considers both objectives, where α, β are parameters to set the relevance of fuel/battery consumption in the aggregated objective, *fuel(t)* is the fuel consumption at time t, SoC_{ref} is the reference value for the battery state of charge, and *SoC(t)* is the battery state of charge at time t. The goal in the environment is to maximize the accumulated value of Equation (18) among the episode, so that the required energy consumption of the vehicle is minimized.

$$r = -\left(\alpha fuel(t) + \beta(SoC_{ref} - SoC(t))^2\right) \tag{18}$$

The environment dynamics are as follows: at the beginning of the episode, the agent receives the car current state as the tuple *(SoC, velocity, and acceleration)*. Then, the agent decides which engine power should be provided according to this state, and sends the selected action to the simulator. Finally, the simulator updates the internal state of the car according to the experimental data and the internal simplified Prius modeling, and returns the next state and the current reward to the agent. This is repeated in a loop until the episode ends, which occurs when the real dataset ends.

The authors of the work [24] proposed a neural network agent to solve both continuous and discrete action set environments. In this manuscript, our interest is not to propose a new approach to improve their findings, but to use the work carried out as a ground test scenario to check whether quantum reinforcement learning can be used as a tool to solve problems related to energy efficiency. To do so, in our experimentation we design a classic multilayer perceptron (MLP) neural network agent with 3 inputs (environment state) and 14 outputs (available discrete actions to be selected), and train the agent in the

Prius environment with the Advantage Actor–Critic (A2C) algorithm. This agent is used as baseline. After that, we also train a quantum agent designed with the methodology described in Section 3 and trained with the same A2C algorithm. The settings for the *Prius* environment and A2C algorithm remain the same for the classic baseline and quantum agents, to make a fair performance comparison.

4.2.2. Experimental Settings

As in use case 1, we also tested the quantum agent implemented as VQCs against the classical MLP, using the same conditions and learning algorithm. As in use case 1, four environments were running in parallel to generate batches of 10 interactions between the agent and each environment, using a discount factor $\gamma = 0.99$ to calculate the total return of each episode. The stopping criterion was set so that the algorithm stops after 100 episodes were finished. In addition, the agent performance was tested every five episodes with a deterministic policy that selected the action with maximum probability to check how the agents were learning to interact with the environment. Finally, we performed 10 different runs with different initial random seeds to validate the experimentation.

The experimental settings for the classical and quantum actor/critic networks are shown in Table 5. In the case of the policy MLP, as the linear combination layers are dense, the number of network parameters is $3 \times 30 + 30 \times 14$ (weights) + 30 + 14 (bias) = 554, and the value MLP has $3 \times 30 + 30 \times 1$ (weights) + 30 + 1 (bias) = 151 parameters. If we focus on the policy VQC, each qubit of every quantum variational layer contains two trainable parameters for R_y, R_z gates. The total number of parameters is 3×14 (preprocessing layer) + $5 \times 2 \times 14$ = 182 parameters. The value VQC network requires three qubits only (one for each input), plus two more parameters to perform the output scale and translation. In this case, the value quantum network has $5 \times 3 \times 2 + 2 = 32$ trainable parameters. It is easy to verify that the number of total parameters required by the quantum proposal is much lower than the number of parameters required to train the classical agent, as happens in use case 1.

Table 5. Structure of policy and value classical/quantum networks used in use case 3.

	Policy MLP	**Value MLP**	**Policy VQC**	**Value VQC**
No. of qubits	-	-	14	3
Inputs	3	3	3. Scaled to $[-\pi/2, \pi/2]$	3. Scaled to $[-\pi/2, \pi/2]$
Preprocessing	-	-	Linear (14 units)	Linear (3 units)
	-	-	Tanh	-
Quantum embedding	-	-	Tensor product encoding	Tensor product encoding
Intermediate Layers	Linear (30 units)	Linear (30 units)	Variational Layer ($\times 5$)	Variational Layer ($\times 5$)
	ReLU	ReLU	-	-
Output layer	Linear (14 units)	Linear (1 unit)	Expectation of $\sigma_z(q_i)$ observable for each qubit q_i	Product of expectation of $\sigma_z(q_i)$ observable for all qubits q_i
Post-processing	softmax activation	-	softmax activation	Scale and translation
Training algorithm	RMSProp ($\lambda = 0.001$)	RMSProp ($\lambda = 0.01$)	RMSProp ($\lambda = 0.01$)	RMSProp ($\lambda = 0.01$)

Finally, we highlight that the previous experimental settings were obtained after a long pre-experimentation in order to find the best configuration for each model.

4.2.3. Results

We describe the performance results considering the total accumulated reward of the agent in an episode, i.e., the sum of the reward function described in Equation (18) in all steps of an episode. As in use case 1, we call average accuracy the average total accumulated reward in the 10 different executions performed. Table 6 summarizes the average, best, and worst total accumulated reward obtained by the MLP and the VQC, the computational time required to finish each experiment (in seconds), and the validation using a deterministic policy that selects the action with maximum probability. The boxplots of the total reward distribution for both VQC and MLP are shown in Figure 13.

Table 6. Results obtained by MLP and VQC agents in the use case 2 environment. Column 1: MLP agent (classic) and VQC agent (quantum); Column 2: Average total reward after training; Column 3: Best total reward after training; Column 4: Worst total reward after training; Column 5: Computational time in seconds; Column 6: Total reward (test with deterministic policy).

Model	Avg. Total Reward	Best Total Reward	Worst Total Reward	Time (s.)	Test Reward
VQC (quantum)	−2.58	−0.99	−4.22	5675.24	0.0
MLP (Classic)	−4.28	−1.79	−8.69	337.04	0.0

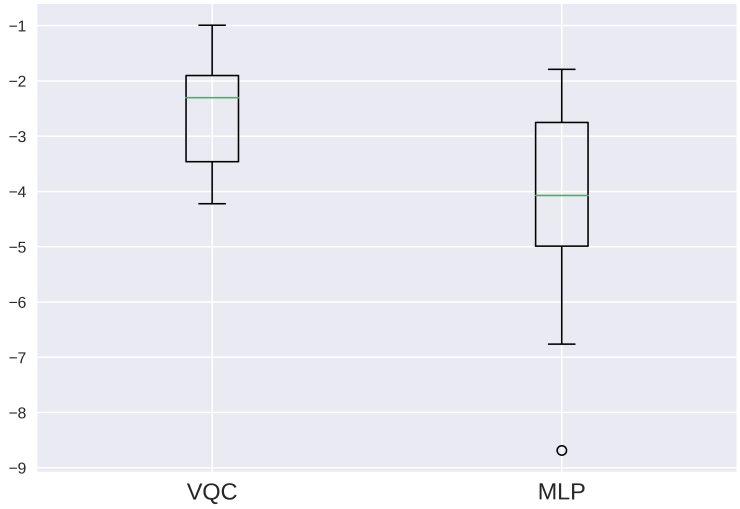


Figure 13. Boxplots of average total reward obtained from VQC and MLP in use case 2.

According to Table 6 and Figure 13, the quantum VQC agent accomplished the task of energy management better than its classic MLP counterpart. In fact, all the average, best, and worst total rewards with a stochastic policy performed better than MLP, and are near the optimum value of 0.0. On the other hand, the clear limitation of the quantum proposal lies in the computational time required to learn the policy, which is near to 1 h 30′ per execution against the approximately 5′40″ required by the classical agent. We validated these results using a paired t-test with 95% confidence level, obtaining a p-value equal to 0.005, which suggests clear statistical differences between the average total reward of VQC and MLP. In any case, neither VQC nor MLP stochastic policies are optimal due to the inherent probabilistic nature of such policies. However, if we transform the stochastic policies to deterministic ones by mean of selecting the action with the highest probability at each episode step, the test rewards of both algorithms perform the same with no statistical differences and are able to learn the optimal policy.

Finally, we analyze the learning curves performed by both agents from episode 1 to episode 100 in Figure 14. As is usual in any deep reinforcement learning problem, the standard deviation of the total reward is higher during the execution of the first episodes in both agents, and it is lower as the agent learns the policy. In Figure 14, we also observe that the MLP is able to obtain a nearly optimal policy in fewer episodes than VQC, and that the quantum model is able to overcome the classic agent just in the last episodes. This is consistent with VQC training problems such as the barren plateau [38,45], which makes the gradient have small values, therefore requiring more iterations to converge. To conclude with the analysis of results, quantum reinforcement learning might compensate a tradeoff between accuracy, model complexity (measured as learnable parameters), and learning time, being able to overcome classical ANNs in terms of long-term learning.

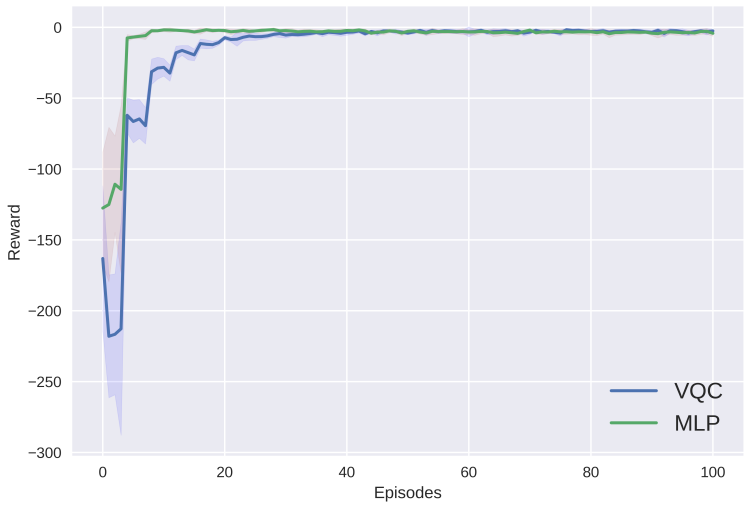


Figure 14. Learning curve obtained from VQC and MLP in use case 2.

4.3. Use Case 3: Profit Optimization of Charging Stations

4.3.1. Problem Statement

This use case differs from the previous experiments since it is not directly targeted at solving energy-efficiency problems. In contrast, it assumes a scenario of scheduling control and real-time pricing in electrical vehicle (EV) charging stations. The reinforcement learning environment used in this experimentation was proposed in [26], and considers the operation of an EV charging station over a discretized period of T time instants. The objective is to develop an optimal policy that maximizes the profit of the charging station, knowing that there are customers that arrive at a certain time slot, then request service, and the scheduler must send them to the service zone. The price of the service also depends on the electricity price at each time slot. The authors of the article [26] formalized the problem as a Markov decision process and created a simulated reinforcement learning environment to provide the scheduler with online learning. The designed environment has the following features:

- **State space:** The original state space in [26] contains eight features, where four of them are directly acquired from a real experimentation over a working charging station, and the remaining four are aggregated features calculated as a function of the raw experimental measurements, reward design, and action design, to obtain the best performance. In this manuscript, we use the first four raw features as input, namely, the set of EVs which are using the charging station at the current time instant, the expected residual charging demand, the parking time of EVs waiting for service, and the online electricity price.
- **Action space:** At each time slot t, the agent must make two decisions: (1) the charging price for the EV that arrived at time t; and (2) the charging rate of each EV in the service zone. Thus, this environment differs substantially from classic reinforcement learning environments, since there are two different action spaces which must be sampled at each time step by the agent to select the two required actions. Both action spaces are discrete: the charging rate can be set to a value in {0 kWh, 20 kWh, 40 kWh, 60 kWh, 80 kWh}, and the charging price is a value in {1, 2, 3, 4, 5, 6}, which indicates a multiplicative factor over the electricity price to offer the service to the customer arriving at the current time slot.
- **Reward function:** The environment sets the charging station profit as the sole criterion to be optimized. Thus, the goal is to maximize the difference between the payment it collects from customers and the charge it has to pay to the utility company to consume the required electricity. This reward is shown in Equation (19), where T_t is the set of

EV arrivals at time t, p_t is the charging price selected at time t, $D_i(p_t)$ is the demand of the i-th EV in response to the price p_t, e_t is the charging rate of the station (service zone) at time t, and c_t is the electricity price the station has to pay at time t.

$$r = \sum_{i \in T_t} p_t D_i(p_t) - c_t e_t \tag{19}$$

The authors of [26] proposed a single artificial neural network that outputs an action from both action sets simultaneously, in response to a current input state. This strategy has the advantage that the internal feature space learned by the network from the input data is shared by the decision layers of both action spaces. However, in this work, we design the agent containing two different networks, one targeted at the selection of an action from each action space. This decision is justified because of the high computational power required to simulate a quantum neural network with the characteristics of [26]. In our case, we have two separate networks with a reduced size which can be run in a quantum program simulation software in affordable time. In fact, by adopting this strategy, we reduce the quantum search space from $\mathbb{C}^{2^{11}}$ to two search spaces in $\mathbb{C}^{2^5}$ and $\mathbb{C}^{2^6}$, respectively, which reduces the simulation computational complexity substantially.

4.3.2. Experimental Settings

As in use case 2, we trained two agents (classical MLP and quantum VQC) under the same reinforcement learning algorithm with the same settings. We also used the A2C algorithm to learn stochastic policies. The A2C configuration used was the same as in use case 2, since in a preliminary battery of experiments it was found that these settings could provide optimal policies. With regards to the policy/value classical/quantum networks, we recall that the environment has two different action sets, and we used two different policy networks (one for each action set). Table 7 shows each network structure, similarly to how it was explained in the previous use case 2.

In the case of the policy MLP for the first action set, the number of network parameters is $4 \times 100 + 100 \times 100 + 100 \times 5$ (weights) $+ 2 \times 100 + 5$ (bias)= 11,105; the number of parameters for the policy MLP in the second action set is $4 \times 100 + 100 \times 100 + 100 \times 6$ (weights) $+ 2 \times 100 + 6$ (bias) = 11,206; and the value MLP has $4 \times 100 + 100 \times 100 + 100 \times 1$ (weights) $+ 2 \times 100 + 1$ (bias) = 10,701 parameters. If we consider the policy VQC network for the first action set, it contains $5 \times 2 \times 5 = 50$ parameters; the policy VQC network for the second action set contains $6 \times 2 \times 6 = 72$ parameters, and the value VQC network has $4 \times 4 \times 4 = 64$ parameters. As happens in the previous use cases, the number of total parameters to be trained by the quantum proposal is on orders of magnitude smaller than the number of parameters of the classic networks.

Table 7. Structure of policy and value classical/quantum networks used in use case 2.

	Policy MLP 1	**Policy MLP 2**	**Value MLP**	**Policy VQC 1**	**Policy VQC 2**	**Value VQC**
No. of qubits	-	-	-	5	6	4
Inputs	4	4	4	4. Scaled to $[-\pi/2, \pi/2]$	4. Scaled to $[-\pi/2, \pi/2]$	4. Scaled to $[-\pi/2, \pi/2]$
Preprocessing	-	-	-	Linear (5 units)	Linear (6 units)	Linear (4 units)
Quantum embedding	-	-	-	Tensor product encoding	Tensor product encoding	Tensor product encoding
Intermediate Layers	Linear (100 units)	Linear (100 units)	Linear (100 units)	Variational Layer (×5)	Variational Layer (×6)	Variational Layer (×4)
	ReLU	ReLU	ReLU	-	-	-
	Linear (100 units)	Linear (100 units)	Linear (100 units)	-	-	-
	ReLU	ReLU	ReLU	-	-	-
Output layer	Linear (5 units)	Linear (6 units)	Linear (1 unit)	Expectation of $\sigma_z(q_i)$ observable for each qubit q_i	Expectation of $\sigma_z(q_i)$ observable for each qubit q_i	Product of expectation of $\sigma_z(q_i)$ observable for all qubits q_i
Post-processing	softmax activation	softmax activation	-	softmax activation	softmax activation	Scale and translation
Training algorithm	RMSProp ($\lambda = 0.001$)	RMSProp ($\lambda = 0.001$)	RMSProp ($\lambda = 0.01$)	RMSProp ($\lambda = 0.01$)	RMSProp ($\lambda = 0.01$)	RMSProp ($\lambda = 0.01$)

The experimental settings described in Table 7 were obtained after a preliminary experimentation was performed to find the best configuration for each model.

4.3.3. Results

The performances of both VQC and MLP agents are described in Table 8, as the total accumulated reward of episodes following the reward function described in Equation (19). As in the previous use cases, Table 8 contains the average, best, and worst total accumulated reward obtained by both quantum and classic agents after the learning process, together with the computational time required to finish each experiment (in seconds) and the total reward over a test episode using a deterministic policy that selects the agent's action with maximum probability. Boxplots in Figure 15 help to support the analysis of the results of the table.

Table 8. Results obtained by MLP and VQC agents in the use case 3 environment. Column 1: MLP agent (classic) and VQC agent (quantum); Column 2: Average total reward after training; Column 3: Best total reward after training; Column 4: Worst total reward after training; Column 5: Computational time in seconds; Column 6: Total reward (test with deterministic policy).

Model	Avg. Total Reward	Best Total Reward	Worst Total Reward	Time (s)	Test Reward
VQC (quantum)	4832.00	4832.00	4832.00	865.47	4832.00
MLP (Classic)	4312.00	4752.00	3846.11	168.83	4704.78

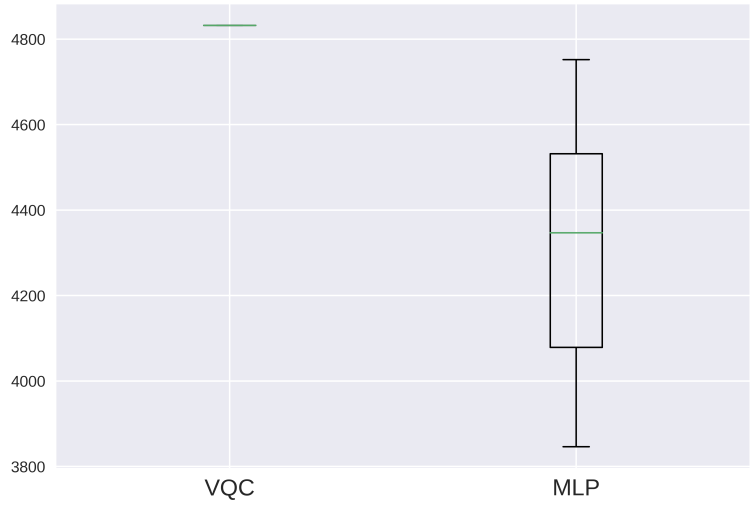


Figure 15. Boxplots of average total reward obtained from VQC and MLP in use case 3.

The first thing we notice in the results shown in Table 8 is that the quantum agent always converges to the same solution, which in turn is the optimal policy, since the average, best, and worst results print the same value of total reward. Additionally, the execution of the VQC in a test episode using the deterministic policy returns the same value of total reward, which supports this statement.

On the other hand, the classical MLP agent is not able to learn the optimal policy, with a total reward of 4752.00 in the best execution and an average total reward of 4312.00 over all executions. Figure 13 shows the average total reward data distributions of both agents at the end of learning, and the previous statement can be verified graphically. In this use case, the quantum VQC agent is also able to overcome its classical counterpart. We validated this analysis using a paired t-test with 95% confidence level, obtaining a p-value under 10^{-6}, which indicates that there are significant differences between the results of executions for VQC and MLP. However, as happens in all use cases, the current limitation of the quantum agent relates to the high computational time required to simulate the quantum

agent in a classic computer. In this use case, it is almost five times higher (approximately 14′30″) than training the MLP (approximately 3′).

To conclude the analysis of results, Figure 16 plots the learning curves of both quantum VQC and classical MLP agents during the learning process from episode 1 to episode 100. It is worth mentioning two facts from this figure: firstly, the classic MLP shows a faster convergence than the quantum VQC agent, achieving an average reward over 4000.00 in only a few episodes of learning. However, it becomes trapped in a local optima which does not reach the optimal total reward. As happened in the previous use cases, the quantum agent learning is slower, but it is able to achieve better solutions than the classical agent. Secondly, the standard deviation of total reward among episodes of different runs is higher in the first episodes of learning of the quantum agent than in the last learning episodes, when it has converged, as should be expected. However, the standard deviation of the total reward in the classical agent remains higher during the whole process of learning. This fact suggests that the model is highly sensitive to initial conditions (weights and bias network initialization values), which makes the classical MLP less robust than the quantum VQC.

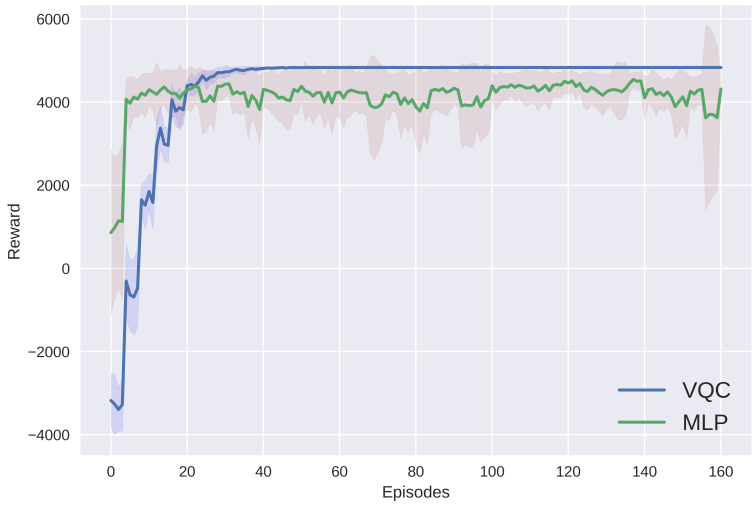


Figure 16. Learning curve obtained from VQC and MLP in use case 3.

4.4. Discussion

We proposed a hybrid classical/quantum model with a classical layer for preprocessing purposes, allowing a reduction of the dimension of the input features by a variational quantum circuit. After the variational layer, a measurement process was performed by the selection of the observables to obtain the expected values and, finally, post-processing using a softmax activation function was performed to transform the measured data into a probability distribution. This structure was applied to three different use cases regarding energy efficiency and management, and it was compared with state-of-the-art classical (non-quantum) RL methods.

A global insight that arises from the study of the results in the three use cases is that the quantum reinforcement learning approach proves the feasibility of implementing decision-making processes with less complexity than their classical counterpart, obtaining better performance than the classical MLP in all use cases studied in this work. On the other hand, they require more time to converge than the classical neural networks, and their computing time is higher than the classical counterpart because of the use of quantum simulators instead of real quantum computers, even when the model complexity is lower than using MLP. In addition, another limitation is that the proposal could not be tested in real contemporary NISQ quantum hardware due to the limitations in accessing these types of computers for large experiments nowadays, and their inherent noise in measurement.

In addition, the quantum agents showed more stability during the training with a lower standard deviation of the total reward, suggesting that the MLP model is more

sensitive to initial conditions (weights, bias, and discount factor), which makes the classical MLP less robust than the quantum VQC (especially in use cases 1 and 3).

5. Conclusions and Future Work

In this work, we carried out a study about the suitability of using quantum decision-making models for deep reinforcement learning in different energy-efficiency and management scenarios. The results suggest that the best performance can be obtained by the quantum agent, although the learning process is slower, requiring more interactions with the environment than classical MLP neural network agents, because of the barren plateau phenomenon, which refers to gradients that vanish exponentially, similarly to classical neural networks, but are becoming even more present in VQC.

On the other hand, the complexity of the quantum agents, measured as the number of free model parameters to be optimized, is orders of magnitude smaller than the number of parameters of the classic networks. This suggests that quantum technologies can be suitable to reduce the size and complexity of decision-making processes, although at the cost of a higher computational time to operate in quantum simulation software.

With regards to the quantum agent, we illustrated the importance of encoding and selecting the observables of a quantum model that can represent the observation spaces and the expected outputs. This task becomes even more important when working with environments with a large number of features. Therefore, as a future work, we would have to investigate how to encode environments with a large number of features and how to handle these observables to obtain the expected output more efficiently using a reduced number of qubits. Another challenge in training quantum agents is avoiding the barren plateaus, which become even more present when increasing the number of qubits, layers, and entanglement gates. Therefore, building expressive but simple circuits plays a fundamental role in quantum reinforcement learning and should be a key aspect to be studied in the future.

It is also worth mentioning the slow learning speed observed, especially in more complex problems with a high number of dimensions in the state space or the action space. In these cases, quantum agents may provide better results than their classical counterpart, although the simulation on classical computers may become a limitation in terms of resources required for computation and the large size of quantum operations, expressed as complex matrix operations of size $2^n \times 2^n$.

In summary, this work is indicative of the potential impact of the relationship between quantum computing and reinforcement learning in energy-efficiency and management scenarios, demonstrating that quantum agents can solve complex environments even better than their classical counterparts.

Author Contributions: Conceptualization, M.P.C. and G.N.; methodology, E.A. and M.P.C.; software, E.A. and M.P.C.; validation, E.A. and M.P.C.; formal analysis, E.A., M.P.C. and G.N.; investigation, E.A., M.P.C. and G.N.; resources, E.A. and M.P.C.; writing, E.A. and M.P.C.; visualization, E.A.; supervision, M.P.C. and G.N.; project administration, M.P.C.; funding acquisition, M.P.C. All authors have read and agreed to the published version of the manuscript.

Funding: This project was supported by the project QUANERGY (Ref. TED2021-129360B-I00), Ecological and Digital Transition R&D projects call 2022, Government of Spain.

Data Availability Statement: The use cases tested in the experimentation were proposed in [21] (simulator publicly available at https://github.com/ugr-sail/sinergym, accessed on 27 July 2022), [24] (the authors made the environment publicly available at https://github.com/lryz0612/DRL-Energy-Management, accessed on 27 July 2022), and [26] (publicly available at https://github.com/wsyCUHK/Reinforcement-learning-for-Real-time-Pricing-and-Scheduling-Control-in-EV-Charging-Stations, accessed on 27 July 2022).

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

A2C	Advantage Actor–Critic
CPU	Central Processing Unit
DQN	Deep Q-network
DRL	Deep reinforcement learning
EV	Electrical vehicle
HVAC	Heating, ventilation, air-conditioning
KPI	Key performance indicator
LSTM	Long/short-term memory
MDP	Markov decision process
MLP	Multilayer perceptron
NISQ	Noisy, intermediate-scale quantum era
QC	Quantum computing
QML	Quantum machine learning
QPU	Quantum Processing Unit
QRL	Quantum reinforcement learning
RL	Reinforcement learning
VQC	Variational quantum circuit

References

1. Dong, H.; Ding, Z.; Zhang, S. *Deep Reinforcement Learning: Fundamentals, Research, and Applications*; Springer: Singapore, 2020. Available online: http://www.deepreinforcementlearningbook.org (accessed on 27 July 2022).
2. Silver, D.; Hubert, T.; Schrittwieser, J.; Antonoglou, I.; Lai, M.; Guez, A.; Lanctot, M.; Sifre, L.; Kumaran, D.; Graepel, T.; et al. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science* **2018**, *362*, 1140–1144. [CrossRef] [PubMed]
3. Badia, A.P.; Piot, B.; Kapturowski, S.; Sprechmann, P.; Vitvitskyi, A.; Guo, D.; Blundell, C. Agent57: Outperforming the Atari Human Benchmark. *arXiv* **2020**, arXiv:2003.13350.
4. Levine, S.; Finn, C.; Darrell, T.; Abbeel, P. End-to-End Training of Deep Visuomotor Policies. *arXiv* **2016**, arXiv:1504.00702.
5. Sallab, A.E.; Abdou, M.; Perot, E.; Yogamani, S.K. Deep reinforcement learning framework for Autonomous Driving. *arXiv* **2017**, arXiv:1704.02532.
6. Wang, Z.; Hong, T. Reinforcement learning for building controls: The opportunities and challenges. *Appl. Energy* **2020**, *269*, 115036. [CrossRef]
7. Mason, K.; Grijalva, S. A review of reinforcement learning for autonomous building energy management. *Comput. Electr. Eng.* **2019**, *78*, 300–312. [CrossRef]
8. Liao, Z.; Peng, J.; Chen, Y.; Zhang, J.; Wang, J. A Fast Q-learning Based Data Storage Optimization for Low Latency in Data Center Networks. *IEEE Access* **2020**, *8*, 90630–90639. [CrossRef]
9. Dutreilh, X.; Kirgizov, S.; Melekhova, O.; Malenfant, J.; Rivierre, N.; Truck, I. Using reinforcement learning for Autonomic Resource Allocation in Clouds: Towards a Fully Automated Workflow. In Proceedings of the ICAS 2011: The Seventh International Conference on Autonomic and Autonomous Systems, Venice, Italy, 22–27 May 2011.
10. Swarup, S.; Shakshuki, E.M.; Yasar, A. Task Scheduling in Cloud Using Deep reinforcement learning. *Procedia Comput. Sci.* **2021**, *184*, 42–51. [CrossRef]
11. Wan, J.; Zhou, J.; Gui, X. Intelligent Rack-Level Cooling Management in Data Centers with Active Ventilation Tiles: A Deep reinforcement learning Approach. *IEEE Intell. Syst.* **2021**, *36*, 42–52. [CrossRef]
12. Fu, Q.; Sun, E.; Meng, K.; Li, M.; Zhang, Y. Deep Q-learning for Routing Schemes in SDN-Based Data Center Networks. *IEEE Access* **2020**, *8*, 103491–103499. [CrossRef]
13. Habibi Khalaj, A.; Halgamuge, S.K. A Review on efficient thermal management of air- and liquid-cooled data centers: From chip to the cooling system. *Appl. Energy* **2017**, *205*, 1165–1188. [CrossRef]
14. Raman, N.S.; Devraj, A.M.; Barooah, P.; Meyn, S.P. reinforcement learning for Control of Building HVAC Systems. In Proceedings of the 2020 American Control Conference (ACC), Denver, CO, USA, 1–3 July 2020; pp. 2326–2332. [CrossRef]
15. Wei, T.; Wang, Y.; Zhu, Q. Deep reinforcement learning for building HVAC control. In Proceedings of the 2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC), Austin, TX, USA, 18–22 June 2017; pp. 1–6. [CrossRef]
16. Lissa, P.; Deane, C.; Schukat, M.; Seri, F.; Keane, M.; Barrett, E. Deep reinforcement learning for home energy management system control. *Energy AI* **2021**, *3*, 100043. [CrossRef]
17. Wang, Y.; Velswamy, K.; Huang, B. A Long-Short Term Memory Recurrent Neural Network Based Reinforcement learning Controller for Office Heating Ventilation and Air Conditioning Systems. *Processes* **2017**, *5*, 46. [CrossRef]
18. Fu, Q.; Han, Z.; Chen, J.; Lu, Y.; Wu, H.; Wang, Y. Applications of reinforcement learning for building energy efficiency control: A review. *J. Build. Eng.* **2022**, *50*, 104165. [CrossRef]

19. Crawley, D.; Pedersen, C.; Lawrie, L.; Winkelmann, F. EnergyPlus: Energy Simulation Program. *Ashrae J.* **2000**, *42*, 49–56.
20. Mattsson, S.E.; Elmqvist, H. Modelica—An International Effort to Design the Next Generation Modeling Language. *IFAC Proc. Vol.* **1997**, *30*, 151–155. [CrossRef]
21. Jiménez-Raboso, J.; Campoy-Nieves, A.; Manjavacas-Lucas, A.; Gómez-Romero, J.; Molina-Solana, M. Sinergym: A Building Simulation and Control Framework for Training reinforcement learning Agents. In Proceedings of the 8th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation, Coimbra, Portugal, 17–18 November 2021; Association for Computing Machinery: New York, NY, USA, 2021; pp. 319–323. [CrossRef]
22. Zhang, Z.; Lam, K.P. Practical Implementation and Evaluation of Deep Reinforcement learning Control for a Radiant Heating System. In Proceedings of the 5th Conference on Systems for Built Environments, BuildSys'18, Shenzen, China, 7–8 November 2018; Association for Computing Machinery: New York, NY, USA, 2018; pp. 148–157. [CrossRef]
23. Scharnhorst, P.; Schubnel, B.; Fernández Bandera, C.; Salom, J.; Taddeo, P.; Boegli, M.; Gorecki, T.; Stauffer, Y.; Peppas, A.; Politi, C. Energym: A Building Model Library for Controller Benchmarking. *Appl. Sci.* **2021**, *11*, 3518. [CrossRef]
24. Lian, R.; Peng, J.; Yuankai, W.; Tan, H.; Zhang, H. Rule-interposing deep reinforcement learning based energy management strategy for power-split hybrid electric vehicle. *Energy* **2020**, *197*, 117297. [CrossRef]
25. Peng, J.; Fan, Y.; Yin, G.; Jiang, R. Collaborative Optimization of Energy Management Strategy and Adaptive Cruise Control Based on Deep reinforcement learning. *IEEE Trans. Transp. Electrif.* **2022**. [CrossRef]
26. Wang, S.; Bi, S.; Zhang, Y.A. reinforcement learning for Real-Time Pricing and Scheduling Control in EV Charging Stations. *IEEE Trans. Ind. Inform.* **2021**, *17*, 849–859. [CrossRef]
27. Sutor, R. *Dancing with Qubits*; Pack: Birmingham, UK, 2019.
28. Wittek, P. *Quantum Machine Learning: What Quantum Computing Means to Data Mining*; Elsevier: Amsterdam, The Netherlands, 2014.
29. Ciliberto, C.; Herbster, M.; Ialongo, A.; Pontil, M.; Rocchetto, A.; Severini, S.; Wossnig, L. quantum machine learning: A classical perspective. *Proc. R. Soc. A Math. Phys. Eng. Sci.* **2018**, *474*, 20170551. [CrossRef] [PubMed]
30. Feynman, R.P. Simulating physics with computers. *Int. J. Theor. Phys.* **1982**, *21*, 467–488. [CrossRef]
31. Grover, L.K. quantum Mechanics Helps in Searching for a Needle in a Haystack. *Phys. Rev. Lett.* **1997**, *79*, 325–328. [CrossRef]
32. Shor, P.W. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a quantum Computer. *SIAM Rev.* **1999**, *41*, 303–332. [CrossRef]
33. Dong, D.; Chen, C.; Li, H.; Tarn, T.J. quantum reinforcement learning. *IEEE Trans. Syst. Man Cybern. Part B (Cybern.)* **2008**, *38*, 1207–1220. [CrossRef] [PubMed]
34. Macaluso, A.; Clissa, L.; Lodi, S.; Sartori, C. A Variational Algorithm for quantum neural networks. In Proceedings of the Computational Science—ICCS 2020, Amsterdam, The Netherlands, 3–5 June 2020; Springer International Publishing: Cham, Switzerland, 2020; pp. 591–604.
35. Skolik, A.; Jerbi, S.; Dunjko, V. quantum agents in the Gym: A variational quantum algorithm for deep Q-learning. *Quantum* **2022**, *6*, 720. [CrossRef]
36. Jerbi, S.; Gyurik, C.; Marshall, S.C.; Briegel, H.J.; Dunjko, V. Parametrized quantum policies for reinforcement learning. *arXiv* **2021**, arXiv:2103.05577.
37. Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; Zaremba, W. OpenAI Gym. *arXiv* **2016**, arXiv:1606.01540.
38. Kwak, Y.; Yun, W.J.; Jung, S.; Kim, J. quantum neural networks: Concepts, Applications, and Challenges. In Proceedings of the 2021 Twelfth International Conference on Ubiquitous and Future Networks (ICUFN), Jeju Island, Korea, 17–20 August 2021; pp. 413–416. [CrossRef]
39. Goodfellow, I.J.; Bengio, Y.; Courville, A. *Deep Learning*; MIT Press: Cambridge, MA, USA, 2016. Available online: http://www.deeplearningbook.org (accessed on 27 July 2022).
40. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A.A.; Veness, J.; Bellemare, M.G.; Graves, A.; Riedmiller, M.; Fidjeland, A.K.; Ostrovski, G.; et al. Human-level control through deep reinforcement learning. *Nature* **2015**, *518*, 529–533. [CrossRef]
41. Shao, K.; Zhao, D.; Zhu, Y.; Zhang, Q. Visual Navigation with Actor-Critic Deep reinforcement learning. In Proceedings of the 2018 International Joint Conference on Neural Networks (IJCNN), Rio de Janeiro, Brazil, 8–13 July 2018; pp. 1–6. [CrossRef]
42. Zhao, C.; Gao, X.S. QDNN: Deep neural networks with quantum layers. *Quantum Mach. Intell.* **2021**, *3*, 15. [CrossRef]
43. Kwak, Y.; Yun, W.J.; Jung, S.; Kim, J.K.; Kim, J. Introduction to quantum reinforcement learning: Theory and PennyLane-based Implementation. In Proceedings of the 2021 International Conference on Information and Communication Technology Convergence (ICTC), Jeju Island, Korea, 20–22 October 2021; pp. 416–420. [CrossRef]
44. Weigold, M.; Barzen, J.; Leymann, F.; Salm, M. Expanding Data Encoding Patterns For quantum Algorithms. In Proceedings of the 2021 IEEE 18th International Conference on Software Architecture Companion (ICSA-C), Stuttgart, Germany, 22–26 March 2021; pp. 95–101. [CrossRef]
45. McClean, J.R.; Boixo, S.; Smelyanskiy, V.N.; Babbush, R.; Neven, H. Barren plateaus in quantum neural network training landscapes. *Nat. Commun.* **2018**, *9*, 4812. [CrossRef] [PubMed]