Quantum Machine Intelligence (2025) 7:19
https://doi.org/10.1007/s42484-024-00226-4

RESEARCH ARTICLE

Nav-Q: quantum deep reinforcement learning for collision-free navigation of self-driving cars

Akash Sinha[1,2] · **Antonio Macaluso**[2] · **Matthias Klusch**[2]

Received: 23 December 2023 / Accepted: 28 November 2024

Abstract
The task of collision-free navigation (CFN) of self-driving cars is an NP-hard problem usually tackled using deep reinforcement learning (DRL). While DRL methods have proven to be effective, their implementation requires substantial computing resources and extended training periods to develop a robust agent. On the other hand, quantum reinforcement learning has recently demonstrated faster convergence and improved stability in simple, non-real-world environments. In this work, we propose Nav-Q, the first quantum-supported DRL algorithm for CFN of self-driving cars, that leverages quantum computation to improve training performance without the requirement for onboard quantum hardware. Nav-Q is based on the actor-critic approach, where the critic is implemented using a hybrid quantum-classical algorithm suitable for near-term quantum devices. We assess the performance of Nav-Q using the CARLA driving simulator, a de facto standard benchmark for evaluating state-of-the-art DRL methods. Our empirical evaluations suggest that Nav-Q shows potential to outperform its classical counterpart in terms of training stability, with reduced sensitivity to different weight initializations and slightly higher average cumulative reward during training. Furthermore, we assess Nav-Q in relation to effective dimension, unveiling that the incorporation of a quantum component results in a model with greater descriptive power compared to classical baselines. Finally, we evaluate the performance of Nav-Q using noisy quantum simulation, observing that the quantum noise deteriorates the training performances but enhances the exploratory tendencies of the agent during training.

Keywords Quantum reinforcement learning · Collision-free navigation · Autonomous driving · Quantum artificial intelligence

1 Introduction

The pursuit of autonomous navigation for self-driving cars has ushered in a new era of transportation, promising safer and more efficient journeys. Central to this endeavor is the challenge of collision-free navigation (CFN), a complex optimization problem that lies at the heart of ensuring the safety and reliability of autonomous vehicles. Over the years, significant strides have been made in tackling this problem, with researchers and engineers harnessing the power of computational methods to chart paths that avoid obstacles and minimize risks [1–3].

In the domain of autonomous navigation, a prevalent approach involves conceptualizing the problem as partially observable Markov decision processes (POMDPs) [4, 5] which poses a significant computational challenge when dealing with real-world scenarios. The current state-of-the-art solutions rely on Deep Reinforcement Learning (DRL) [1–3], which can be integrated with POMDP planning [4, 5]. While these techniques have demonstrated promising results, they demand substantial computing resources and training time to develop a robust reinforcement learning (RL) agent.

In parallel, the field of quantum reinforcement learning (QRL) has recently emerged as an innovative paradigm that harnesses the computational capabilities of quantum computing in conjunction with the principles of reinforcement learning (RL). Early experiments [6–9] have showcased the potential of QRL methods to improve training performance when applied to simplified, non-real-world environments, often exemplified by the benchmark domains provided by OpenAI Gym [10]. This improvement is usually assessed by analyzing the *Return vs Episodes* curve, which allows for

✉ Antonio Macaluso
antonio.macaluso@dfki.de

[1] Department of Mathematics and Computer Science, Saarland University, Saarbrucken, Saarland, Germany

[2] Agents and Simulated Reality Department, German Research Center for Artificial Intelligence (DFKI), Saarbrucken, Saarland, Germany

Published online: 11 February 2025

evaluation of how fast and how well a given agent is capable of learning a good policy as long as the training epochs run, and in particular, this is done by analyzing the AUC of the curve. This approach allows comparing quantum and classical architectures experimentally even when running quantum components using simulations, which is intrinsically hard for classical computation and hampers to consider the effective training times in terms of hours/days. However, a critical gap remains to be bridged as none of these methods have been tested in a challenging, real-world environment, such as the CFN of self-driving cars.

Furthermore, many of the suggested methodologies presuppose replacing traditional neural networks, which provide action directives, with parametrized quantum circuits (PQCs). In the context of CFN, this implies the requirement of a quantum computing device installed onboard the vehicle for experimentation—a requirement that is impractical to fulfill.

This paper delves into the feasibility and potential advantages of quantum-supported DRL within the domain of CFN of self-driving cars. Specifically, we introduce a novel quantum-supported architecture that leverages quantum computation during training, resulting in state-of-the-art performance during testing. Precisely, the contributions are the following:

1. Introducing Nav-Q, the first quantum-supported deep reinforcement learning (DRL) approach that combines quantum and classical neural networks to tackle the CFN problem of self-driving cars. Nav-Q employs an actor-critic algorithm, with the critic implemented through a hybrid quantum-classical approach, aiming to enhance the performance during training without requiring an onboard quantum device for testing.
2. Implement a modification of the data reuploading technique [11] named Qubit Independent Data Encoding and Processing (QIDEP). This variant incorporates the concept of a sublayer, allowing for versatile encoding of high-dimensional classical data and providing flexibility in the number of qubits used.
3. Train and evaluate Nav-Q on the CARLA-CTS1.0 benchmark [5], a widely recognized standard for testing DRL models in the context of self-driving cars. Nav-Q's performance is compared with its natural classical counterpart, NavA2C (Sect. 3.3), which exclusively utilizes classical computational resources. The results suggest that Nav-Q, executed using noiseless simulations, shows improved stability across different architectural initializations and a slightly higher average return. However, these conclusions are specific to the experiments performed and may not necessarily generalize to other settings. Additionally, Nav-Q attains state-of-the-art performance in the driving policy while substantially reducing the number of parameters. It also showcases a heightened potential for learnability when evaluating the effective dimension as a metric to assess the model's ability to learn complex patterns from data.
4. Train a reduced version of Nav-Q on a simulated noisy quantum system with the parameter-shift rule to estimate the expected performance when experiments are performed on near-term quantum hardware.

The rest of the paper is structured as follows. A discussion of the related works, quantum and classical, is reported in Sect. 2. This is followed by the formalization of the CFN problem as a POMDP and a detailed description of an advanced classical solution named NavA2C, which serves as the classical baseline, in Sect. 3. Our quantum-supported solution Nav-Q is described in Sect. 4 and its performance comparatively evaluated in Sect. 5. Section 6 concludes with a summary of achievements and future work.

2 Related works

In this section, we provide a summary of the existing literature relevant to two key aspects of this paper's contributions. The first part delves into current state-of-the-art approaches for addressing the CFN problem, primarily centered around DRL algorithms. Secondly, we explore the QRL landscape, which focuses on training parameterized quantum circuits to function as agents within classical environments.

2.1 Deep reinforcement learning for collision-free navigation

The state-of-the-art approach for addressing the CFN problem of self-driving cars typically involves formulating it as a POMDP and solving it through DRL. Different approaches are possible depending on the specific problem the DRL algorithm is tasked with, ranging from simple lane following to complex tasks like end-to-end driving in urban scenarios [1–3]. These methods exhibit variations in the DRL algorithm used, system architecture, reward design, tasks, input, and actions performed by the agent.

Recently, different actor-critic-based architectures for DRL have been proposed. In particular, A3C approaches [12, 13] execute on multiple parallel instances of the environment to use the parallel actor-learners for stabilizing effect on training process. The adaptation of this approach in the context of CFN is referred to as GA3C-CADRL [14] and has been tested for mobile robot navigation. In this case, the observation of the environment is not presented as a picture image but rather as the position, velocity, and orientation of the robot. Also, an LSTM [15] is used to encode temporal and spatial data in this scenario. Alternatively, A2C is the synchronous

Table 1 An overview of the characteristics of the environments used to evaluate the performance of existing quantum reinforcement learning methods

Environment	Observation space	Observation shape	Action space	Action shape
Acrobot	Continuous	(6,)	Discrete (3)	(1,)
Blackjack	Tuple (Discrete (32), Discrete (11), Discrete (2))	(3,)	Discrete (2)	(1,)
Cartpole	Continuous	(4,)	Discrete (2)	(1,)
Frozen lake	Discrete (16)	(1,)	Discrete (4)	(1,)
Mountain car	Continuous	(2,)	Discrete (3)	(1,)
Pendulum	Continuous	(3,)	Continuous	(1,)

version of the A3C model, that waits for each agent to finish its experience before conducting an update. In this respect, A2C-CADRL-p has been recently introduced [5] as a synchronous variant of GA3C-CADRL that utilizes the output of path planning and the perceived environment as input to the network, as opposed to end-to-end learning, enabling the agent to navigate the environment with a specific goal.

Beyond A2C and A3C approaches, whose performances are comparable, recent work has highlighted proximal policy optimization (PPO) as a standout approach for self-driving cars [16, 17]. PPO excels due to its stability, sample efficiency, and adaptability to continuous action spaces. By preventing large policy updates, it ensures stable learning, particularly crucial for safety-critical applications. Moreover, its compatibility with parallel processing accelerates learning in data-intensive scenarios.

At the time of writing this paper, Pennylane [18] with the PyTorch plugin did not support parameter broadcasting.[1] Consequently, the quantum version of PPO, which requires batch processing of data through quantum circuit simulations, incurred a substantial time cost on classical computers, rendering training these algorithms impractical. Therefore, for practical purposes, our focus is on the classical baseline NavA2C (see Sect. 3.3) based on A2C-CADRL-p [5]. This approach combines a symbolic path planner and a modified A2C agent in one architecture, with the DRL agent solely responsible for the speed action. Nevertheless, the Nav-Q methodology (described in Sect. 4) utilizes an actor-critic model and can be seamlessly adapted to the PPO approach, provided that the available quantum software tools support efficient parameter broadcasting.

2.2 Quantum reinforcement learning

The standard approach in quantum reinforcement learning (QRL) consists of replacing a classical Neural Network with a PQC to enhance performance in terms of trainability, stability, and learning policy.

[1] https://docs.pennylane.ai/en/stable/code/api/pennylane.broadcast.html

Among the various alternatives, the use of a quantum Q-learning algorithm has been investigated and compared to a deep Q-network, revealing a polynomial improvement in terms of parameter space complexity of PQCs with respect to classical DQN [8]. This investigation focused on PQCs operating in a straightforward deterministic environment with a single input value, where both the observation space and output space were restricted to the same number of quantum bits. In a subsequent study [19], alternative PQC architectures and encoding schemes were explored, extending the algorithm's testing to slightly more complex environments. Another advancement involved a novel data encoding scheme using trainable weights, enhancing quantum Q-learning performance [9]. Additionally, a Quantum Soft Actor-Critic (Q-SAC) algorithm has been proposed [7], where the actor is implemented as a PQC and the critic utilizes a classical neural network. Applications of Q-SAC include controlling a simulated robotic arm with continuous action and state spaces [20]. Another study proposed the Softmax-PQC, employing a softmax policy to successfully address various standard RL tasks based on REINFORCE [6, 21]. Additionally, a QRL algorithm based on PPO [16] has been introduced, with the actor implemented using a PQC and the critic with a classical neural network [22, 23]. Lastly, works on the adoption of PQCs for generic POMDPs [24–26] suggested a learning advantage over selected classical counterparts when tested on simplified environments.

All current QRL methods (summarized in Table 1) have been assessed in the context of OpenAI Gym environments. These environments are characterized by being static, fully observable, and featuring a low-dimensional state space. However, the CFN problem of self-driving cars is dynamic, partially observable, and involves high-dimensional observations and state spaces. As a result, existing QRL algorithms fall short in effectively addressing CFN problems. Moreover, PQCs primarily determine the actions an agent should take within an environment. Even when adapting existing QRL methods for more intricate problems, incorporating a quantum processing unit (QPU) into the car becomes a necessity in algorithm design. However, this requirement is impractical given the current challenges associated with building a

real QPU. Therefore, our goal is to introduce a novel method capable of harnessing the potential advantages of quantum computing within the realm of CFN without installing a QPU in cars.

3 Problem formulation

Collision-free navigation (CFN) refers to the task of driving on a drivable path from one location to another while minimizing time to goal and the number of collisions with static and dynamic obstacles. CFN task can be modeled as a POMDP [4, 5] problem that requires a formal definition of the car model, pedestrian model, and environment.

3.1 Car and pedestrian model

The Car state is defined as a 4-tuple $(p_t^c, p_{goal}^c, v_t^c, \theta_t^c)$ where $p_t^c \in \mathbb{R}^2$ is the current position, $p_{goal}^c \in \mathbb{R}^2$ is the goal position, $v_t^c \in \mathbb{R}$ the speed and $\theta_t^c \in [0, 2\pi)$ the orientation of the car at instance t. The car uses the bicycle model to approximately model the car driving on the road. We assume for the sake of this work that the car can detect all unoccluded static and dynamic obstacles within a radius of 50 m. The vehicle has a 360° view of its surroundings. However, in the case of dynamic obstacles, only its current position is visible; the car's perception model is unable to see the obstacle's goal position or the potential direction in which it might move. Additionally, we assume that all available sensor data is noise-free. For goal-directed navigation of the driverless car, we use the *Anytime Weighted Hybrid A* Path Planner* to calculate a driveable path for the car from source to destination. This planner needs a 2-dimensional discretized cost map of the environment to plan the route. Our basic cost map converts environmental data from the actual scene to a discretized grid map from a bird's eye view, with obstacle costs of car states defined as maximum of 1, 50, and 100, respectively, if the car is entirely on the road, partially on the sidewalk, and collides with an obstacle anywhere else.

The pedestrian state is represented by a 4-tuple $(p_{\mathrm{t}}^{ped}, p_{\mathrm{goal}}^{ped}, v_{\mathrm{t}}^{ped}, \theta_{\mathrm{t}}^{ped})$, where $p_{\mathrm{t}}^{ped} \in \mathbb{R}^2$ is the current position of the pedestrian, $p_{\mathrm{goal}}^{ped} \in \mathbb{R}^2$ is the goal position of the pedestrian, $v_{\mathrm{t}}^{ped} \in \mathbb{R}$ is the speed of the pedestrian and $\theta_{\mathrm{t}}^{ped} \in [0, 2\pi)$ is the orientation of the pedestrian at time instance t. We assume that the pedestrians move towards the goal in a straight line.

3.2 Collision-free navigation problem

The problem of CFN can be formulated as a POMDP(S, A, T, R, γ, Z, O) as follows:

S: This represents the set of states, $\{[c, \mathrm{ped}_1, \ldots, \mathrm{ped}_{|P|}] | c \in \mathbb{R}^2 \times \mathbb{R}^2 \times \mathbb{R} \times [0, 2\pi), \mathrm{ped}_i \in \mathbb{R}^2 \times \mathbb{R}^2 \times \mathbb{R} \times [0, 2\pi), 1 \leq i \leq |P|\}$, where c denotes the car's state at time t, and $\mathrm{ped}_i = [\mathrm{ped}_o^i, \mathrm{ped}_h^i]$ represents the observable and hidden state of pedestrian i ($1 \leq i \leq |P|$).

A: This denotes the set of all possible actions. Each action $a \in A$ is defined as a 2-tuple (α, acc), where α is the steering angle and acc is the speed action. The steering angles are discretized into bins of 25° with a maximum steering angle $\alpha_{\max} = 50°$ in both directions. The set of all possible steering directions is Angles $= \{-50°, -25°, 0°, 25°, 50°\}$. The car can choose between three distinct speed actions: SpeedActions = {accelerate, maintain, decelerate}. The accelerate action corresponds to an increase in speed by 5 kmph, the decelerate action corresponds to a decrease in speed by 5 kmph, and the maintain action corresponds to no change in speed.

T: This represents the transition probability, $T(s_t, a_t, s_{t+1}) = p(s_{t+1}|s_t, a_t) \in [0, 1]$, which defines the likelihood of transitioning from state $s_t \in S$ to state $s_{t+1} \in S$ when taking action $a_t \in A$ at time t. These transitions are fully determined by car movement kinematics and the pedestrian model.

Z: This signifies the set of observations, $\{\{[c, ped_1^o, ped_2^o, ..., ped_n^0]\}\}$, containing information about the car's state and the observable aspects of pedestrians in the scene.

O: This denotes the observation probability, $O(s_{t+1}, a_t, o_{t+1}) = p(o_{t+1}|s_{t+1}, a_t) \in [0, 1]$, which represents the likelihood of observing $o_{t+1} \in Z$ when transitioning to s_{t+1} after executing action a_t.

R: This stands for immediate rewards, $R(s_t, a_t) \in \mathbb{R}$, for executing action a_t in state s_t. The reward function is formally defined in Definition 2

γ: This represents the discount factor within the range [0, 1], and we set $\gamma = 0.99$.

3.3 Classical baseline NavA2C

In this section, we describe NavA2C, a DRL-based architecture that will serve as a baseline for comparison with our proposed quantum-supported method, Nav-Q.

NavA2C, inspired by A2C-CADRL-p [5], combines a symbolic path planner [4] and a modified A2C [12] agent, such that the DRL agent is only responsible for the speed action. The architecture is shown in Fig. 1.

The input for the DRL agent is the car intention I (Fig. 2) which is the semantic segmentation of the environment from a bird's eye view overlayed with the planned and traveled path. The DRL agent outputs the speed action acc_t at every time step using car intention I along with the reward R_t, car's velocity v_t^c, and car's speed action in the previous time step

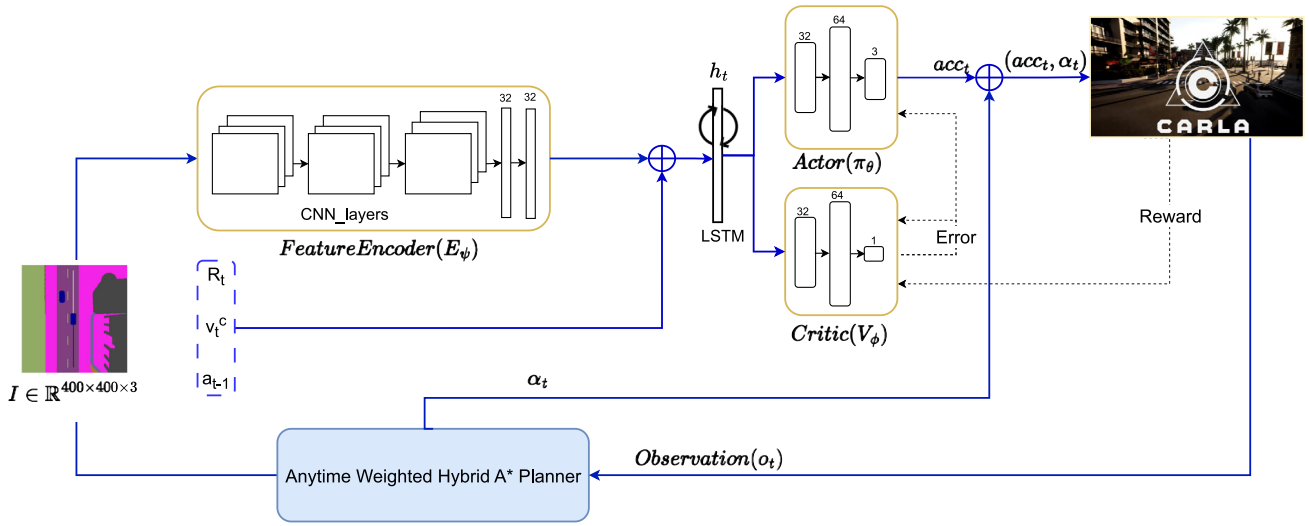


Fig. 1 NavA2C architecture

acc_{t-1}. The steering action α_t, needed to control the car's lateral trajectory, is derived from the path calculated by the *Anytime Weighted Hybrid A* Path Planner* at every time step. The speed action acc_t together with the steering angle α_t is passed on to CARLA simulator as a tuple $a_t = (acc_t, \alpha_t)$ which serves as the control signal for the car simulated in the CARLA environment.

The DRL agent in NavA2C consists of four main components: feature encoder E_ψ, actor π_θ, LSTM, and critic V_ϕ. E_ψ is responsible for learning a low-dimensional representation of car intention I using a long sequence of CNN layers. The output of the feature encoder, $E_\psi(I)$, is concatenated with the reward $R_t \in \mathbb{R}$, the car's velocity $v_t \in \mathbb{R}^2$, and the car's speed action in the previous step $acc_{t-1} \in \mathbb{R}$. This concatenated information is passed as input to the LSTM. The LSTM's hidden state h_t is then passed to the actor and critic networks to determine $\pi_\theta(a_t|h_t, o_t)$ and $V_\phi(h_t, o_t)$, respectively.

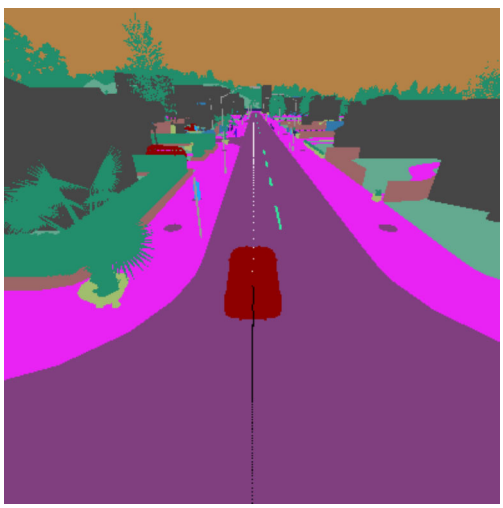


Fig. 2 Car intention $I \in \mathbb{R}^{400\times400\times3}$ is the semantic segmentation of the bird eye view of the environment. The white dotted line represents the planned path while the black dotted line represents the path already traveled by the car

The actor (π_θ) is implemented through a dense neural network. It takes input h_t and produces a 3-dim vector with associated probabilities for speed actions (accelerate, maintain, and decelerate). The critic V_ϕ is also implemented using a dense neural network and evaluates the actions chosen by the actor by estimating the expected cumulative reward associated with a given state-action pair. By leveraging these evaluations, the actor adjusts its policy to prioritize actions with higher rewards.

Specifically, the outputs of the actor and critic are the two components that allow the calculation of the loss function during training, defined as follows.

Definition 1 (NavA2C Loss Function) *Given the observation o_t, action a_t, value function V_ϕ with parameters ϕ, and actor policy function π_θ with parameters θ, the loss functions to train parameters ϕ and θ are given by,*

$$J_V(\phi) = (G_t - V_\phi(h_t, o_t))^2 \tag{1}$$

$$J_\pi(\theta) = \log(\pi_\theta(a_t|h_t, o_t)) \cdot (G_t - V_\phi(h_t, o_t)) + \beta H(\pi_\theta(a_t|h_t, o_t)) \tag{2}$$

where β is a hyperparameter that controls the weight of the entropy term, G_t is the total discounted reward (See Definition 3*), and*

$$H(\pi_\theta(a_t|h_t, o_t)) = \sum_{a_t \in \mathcal{A}} \pi_\theta(a_t|h_t, o_t) \cdot \log(\pi_\theta(a_t|h_t, o_t)) \tag{3}$$

is the entropy of the actor policy.

Importantly, the critic is utilized in calculating the loss function during training, providing feedback on the quality of selected actions. Its use is not required at the time of testing. Nevertheless, opting for a critic with limited expressive power may pose challenges in capturing complex relationships, leading to slow learning, and difficulties in generalization and stability. To address these issues, selecting a critic with ample capacity to represent environmental intricacies, through advanced models, is crucial. Thus, the primary distinction between the training architecture depicted in Fig. 1 and the corresponding testing architecture lies in the absence of the critic.

Another crucial aspect is the design of an effective reward function, as it directly impacts the learning process and the behavior of the autonomous vehicle. In fact, a meticulously crafted reward function must align the learning objectives with the desired behavior, contributing to the development of a reliable and safe autonomous driving system. Drawing inspiration from prior works [5, 14], we define the following reward function for NavA2C.

Definition 2 (NavA2C Reward) *Given the current state $s_t \in S$ and previous action $a_{t-1} \in \mathbb{A}$, the reward $R_t(s_t, a_{t-1})$ is,*

$$R_t(s_t, a_{t-1}) = \begin{cases} r_{goal\_reached} = +200 & \text{for reaching goal position, i.e., } p_c(t) = p_c^{goal} \\ r_{hit} = -100 * \beta & \text{For colliding with the obstacles (pedestrian, cars, etc.) } \beta \text{ is proportional to the impact speed} \\ r_{obstacle} = -obstacleCost & \text{Penalty if there is a pedestrian or cars in the hit area and } v_t^c \neq 0 \\ r_{near\_miss} = -10 & \text{If there is a pedestrian in the near-miss area of car} \\ r_{over\_speeding} = -10 & \text{For crossing the maximum speed limit of 50 kmph} \\ r_{not\_goal} = -goal\_dist/1000 & \text{Dynamic penalty based on distance from goal} \\ r_{braking} = -1 & \text{Penalty if the car brakes when it's not moving} \\ r_{steer} = -1 & \text{Penalty if } \alpha_t \neq 0 \end{cases}$$

Given the reward function, we can easily define the total discount reward (*return*) as follows.

Definition 3 (Total Discounted Reward) *Given the partially observable Markov decision process (POMDP) denoted by $(S, A, T, R, \gamma, Z, O)$, the total discounted reward G_t starting from time t is defined as follows:*

$$G_t = E\left[\sum_{i=t}^{\infty} \gamma^{i-t} R_t\right].$$

One drawback of NavA2C [5] is the consistent acceleration during the test phase when actions are determined deterministically. To address this issue, we mitigated $r_{goal\_reached}$ and introduced a penalty for overspeeding, denoted as $r_{over\_speeding}$. Drawing inspiration from HyLEAR [5], we incorporated r_{near_miss} into the reward function to penalize near misses, encouraging the car to maintain a safe distance from pedestrians. Additionally, r_{hit} penalizes crashes or collisions with pedestrians, promoting the safety of the autonomous car. The term $r_{obstacle}$ is designed to encourage the autonomous car to avoid collisions with other obstacles. Furthermore, r_{steer} promotes comfortable and smoother navigation by penalizing the agent for choosing actions that necessitate excessive steering. Finally, r_{not_goal} incentivizes reaching the goal more expeditiously.

4 Methods

In this section, we introduce Nav-Q, the first quantum-supported deep reinforcement learning architecture that

combines classical and quantum neural networks to effectively train a reinforcement learning agent for the safe navigation of self-driving cars. Nav-Q is primarily based on an actor-critic model, wherein the critic component is realized through a parameterized quantum circuit and a fully connected classical layer. This design enables us to harness the capabilities of a quantum algorithm exclusively during the training phase, eliminating the need for a QPU in the vehicle during testing.

Additionally, we introduce a novel quantum embedding technique that extends the data reuploading strategy to efficiently encode high-dimensional vectors using an arbitrary number of available qubits.

4.1 Nav-Q architecture

Nav-Q is inspired by NavA2C (Sect. 3.3), which comprises two major components: an *Anytime Weighted Hybrid A* Path Planner* [4], determining the steering angle for the car, and DRL components-a hybrid actor-critic model responsible for determining speed actions (accelerate, maintain, decelerate). The objective is to reach the destination in the shortest time possible while avoiding collisions with other agents that may be present in the scene (such as other cars, pedestrians).

To be able to generalize any possible scenario that the car might encounter, the DRL agent needs to be trained for thousands of episodes, which implies running the training procedure for several days even when considering a limited set of scenarios. Furthermore, DRL algorithms are very unstable with respect to weight initialization, which means that it is necessary to run the training procedure several times with different initial parameters to achieve reliable performance in terms of driving policies.

The idea of Nav-Q is to introduce a quantum component into the DRL agent of NavA2C to obtain a quantum-supported architecture that can improve classical performance in terms of trainability and/or stability while having (at least) the same performance in terms of learned policy (time to goal, number of crashes, etc.). Also, the quantum component has not to be included at the time of testing, since this would require placing a QPU in the car, and invalidate the practical use of the proposed method.

To fulfill these requirements, Nav-Q introduces a parametrized quantum circuit to implement the critic within the actor-critic framework. This strategic choice allows us to exclusively employ the quantum algorithm during the training phase, where the critic evaluates the policy selected by the actor. In contrast, during the testing phase, only the actor is utilized, as it produces the action. Therefore, the training process, recognized as the most challenging aspect of RL, is where the introduction of quantum components is expected to have the maximum impact.

The quantum-supported DRL agent of Nav-Q is visually represented in Fig. 3.

4.2 Hybrid quantum-classical critic

The critic $V_\phi(h_t, o_t)$ in Nav-Q is realized through a hybrid quantum-classical algorithm made up of two components. The first one is a PQC which processes h_t—the hidden state of the LSTM—through unitary operations to generate a parametrized quantum embedding in the Hilbert space. The quantum embedding is then converted into an q-dimensional classical vector (where q is the number of qubits of the quantum circuit) using the expectation measurement of Pauli-Z observables for each single qubit. At this point, there is

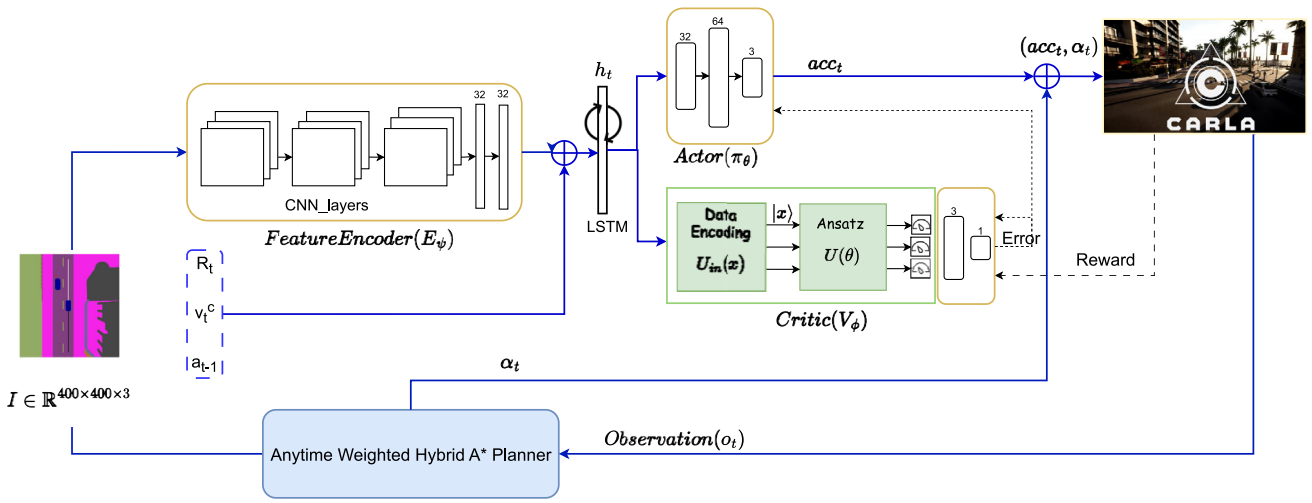


Fig. 3 Quantum-supported deep reinforcement learning agent in Nav-Q. In contrast to the complete architecture of NavA2C illustrated in Fig. 1, the primary distinction from the classical model lies in the DRL agent's critic, which is replaced by a hybrid quantum-classical algorithm. Its design is essential for training and evaluating the loss function

a mismatch between the output of the PQC ($\mathbb{R}^q$) and the required dimension of the Value function ($V_\phi(h_t, o_t) \in \mathbb{R}$). To map the q-dimensional vector resulting from the PQC to a single value of V_ϕ, the second part of the critic is implemented. This part is essentially a fully connected (FC) neural network layer made up of q input nodes and one output node (with no hidden layers), which calculates a linear combination of the q-dimensional vector from the quantum circuit without applying any kind of non-linear operation.

Despite the reduction in problem dimensionality given by the combination of the feature encoder and the LSTM, the complexity of solving real-world issues, such as the CFN, poses a challenge in terms of data encoding within a parametrized quantum circuit. To tackle this challenge, we propose an implementation of the data reuploading technique [11], named *Qubit Independent Data Encoding and Processing* (QIDEP), which enables the encoding of a high-dimensional vector using an arbitrary number of qubits.

Note that the utilization of θ parameters for the actor and ϕ parameters for the critic aligns with established RL literature on actor-critic approaches. However, starting from the next section, we employ the parameters θ to denote the trainable parameters of the quantum circuit implementing the critic to be consistent with quantum computing literature.

4.3 Qubit Independent Data Encoding and Processing

Data reuploading [11] is a useful strategy for quantum embedding that enables the approximation of any continuous function using a single-qubit circuit. We implement its extension to multi-qubit quantum circuits and named this approach Qubit Independent Data Encoding and Processing (QIDEP). This implementation strategy uses a multi-qubit system to encode a high-dimensional vector with a flexible number of qubits. A formal definition of this strategy is provided in Definition 4.

Definition 4 (Qubit Independent Data Encoding and Processing) *Let $x \in \mathbb{R}^p$ be the input classical data, $U(\theta)$ be a parameterized quantum circuit with trainable parameters θ, $U_{in}(x)$ be the unitary for data encoding based on Angle Encoding, q be the desired number of qubits and $L \in \mathbb{N}$ be the desired number of layers in the quantum circuit. In QIDEP, each layer U_l can be written as a composition of sublayers S_m^l*

$$U_l = \prod_{m=1}^{m=k} S_m^l \tag{4}$$

where S_m^l is the m-th sublayer of the l-th layer defined as $S_m^l = U(\theta_m^l)U_{in}(x^m)$, with $x^m = \{x_{3q(m-1)+1}, \ldots, x_{3qm}\}$ indicating a specific subset of elements in the input feature x. Note that the number of sublayers $k = \lceil \frac{p}{3q} \rceil$ is fixed and given the input feature and the number of qubits, the vector must be padded with a zero vector of length $3qk - p$.

Importantly, the architecture allows to freely choose the number of layers, as well as the number of trainable parameters within each sublayer, depending on the task at hand. Thus, a L-layered, q-qubit QIDEP quantum circuit that has to process a p-dimensional vector can be written as follows:

$$U(\theta, x) = \prod_{l=1}^{L} \prod_{m=1}^{m=k} U\left(\theta_m^l\right) U_{in}\left(x^m\right). \tag{5}$$

Thus, the total number of trainable parameters using QIDEP strategy is determined as $(L \times \lceil \frac{p}{3q} \rceil \times 2q) + (q+1)$ where L is the number of layers, p is the dimensionality of data, and q is the number of qubits.

An example of QIDEP strategy is shown in Fig. 4. In data reuploading [11], each layer can be seen as a single unit of U_{in} and $U(\theta)$, and the number of layers in the circuit is considered a hyperparameter (as in classical neural networks). In QIDEP, we introduce the concept of sublayer S. Specifically, each layer l consists of k sublayers S_m^l, each comprising unitaries $U_{in}(x)$ and $U(\theta)$. Here, $U_{in}(x)$ is employed to encode a contiguous sub-sequence of features of x, while $U(\theta)$ represents the parametrized ansatz. Figure 5 shows an example of a sublayer considering a 4-qubit circuit and a hardware-efficient [27] parametrized ansatz.

5 Performance evaluation

In this section, we assess Nav-Q using the CARLA-CTS benchmark, a widely recognized virtual environment for evaluating the performance of DRL models in the context of self-driving cars. Additionally, we compare the results of Nav-Q with those of its classical baseline, NavA2C, which relies solely on conventional computational resources. Finally, we train Nav-Q on a simulated noisy quantum system with the parameter-shift rule to estimate the expected performance on real quantum hardware.

5.1 Experimental settings

Environment

For comparative experimental evaluation of CFN methods of self-driving cars in critical traffic scenarios, we use the virtual CARLA-CTS benchmark [5], an extension of OpenDS-CTS [4] for the driving simulator CARLA [28]. This benchmark contains about 30,000 scenes of 12 scenarios mainly based on the GIDAS [29] study of road accidents with pedestrians in Germany and simulated in CARLA on a test drive of about 100 ms. In Fig. 6, the blue boxes with solid arrows

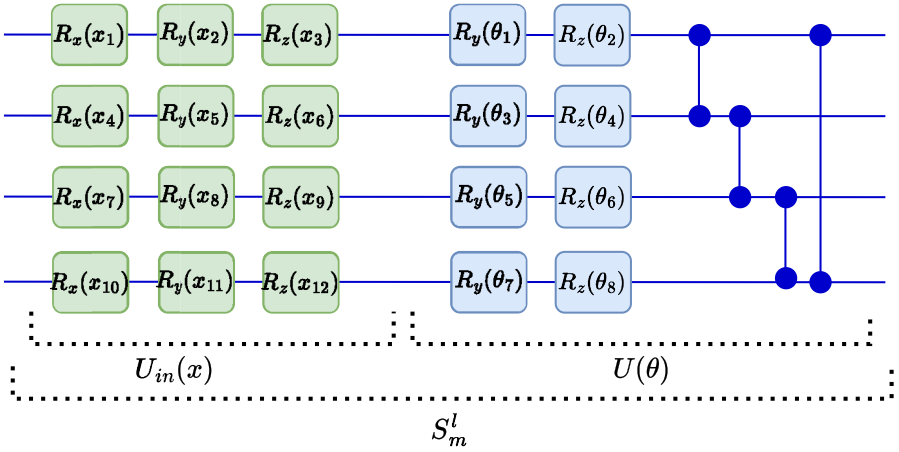


Fig. 4 Example of a quantum circuit implementing the QIDEP strategy to encode a 24-dimensional vector using 4 qubits and 3 layers. The quantum circuit is segmented into layers (gray boxes U_1, U_2, U_3), each containing identical quantum gates but with varying trainable parameters in each ansatz. In particular, each layer consists of sublayers S_m^l, enclosed by the red dashed line (detailed description provided in Fig. 5). Notice that the additional classical fully connected layer connected to the measurements of the 4 qubits is not part of the specific QIDEP design but rather a component of the Nav-Q architecture

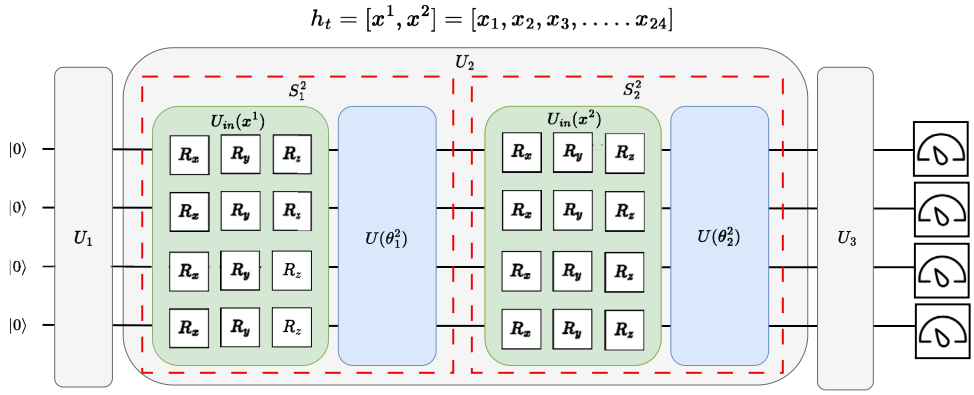


Fig. 5 Graphical representation of the sublayer concept in QIDEP, comprising $U_{in}(x)$ and $U(\theta)$. The quantum gates of $U_{in}(x)$ for data encoding are illustrated in green, while the ansatz $U(\theta)$ with trainable parameters is depicted in blue. In this example, the $U(\theta)$ structure adopts a hardware-efficient architecture, consisting of parameterized R_y and R_z gates, followed by a daisy chain of CZ gates. On the other hand, the quantum gates of U_{in} align with those proposed in the classical data reuploading technique [11] consisting of a sequence of the three Pauli rotation gates, whose parameters are non-trainable and depend on the input feature x

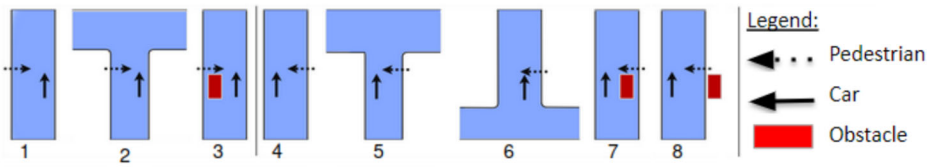


Fig. 6 Accident scenarios on the basis of GIDAS [29] study. The autonomous vehicle is represented by the solid lines, stationary objects that block its view are represented by the red squares, and pedestrian movement is represented by the dotted lines. Scenario id is represented by the numbers at the bottom of each scenario

denote autonomous car movements, the red boxes with solid arrows denote static or dynamic obstacles such as parking or incoming cars, and the dotted arrows denote pedestrian movement.

Since Nav-Q is trained using quantum simulations, the training time is considerably high; thus, to to reduce the complexity of the tested environment, we adopt training scenarios 1, 3, 4, 5, 6, and 8 and test our model on all scenarios from 1 to 8. A scene is represented by a 6-tuple (scenario, p^c_{start}, p^c_{goal}, θ^c_{start}, P, C), where scenario $\in$ Scenarios correspond to the scenario being simulated in the scene, $p^c_{\text{start}} \in \mathbb{R}$ denotes the starting position of the car, $\theta^c_{\text{start}} \in \mathbb{R}^2$ represents the starting orientation of the car, $p^c_{goal} \in \mathbb{R}^2$ indicates the goal position of the car, P is the set of pedestrians in the environment and C is the set of cars in the environment apart from the autonomous car c. Each scene is essentially an instantiation of a particular scenario with a specific configuration for pedestrian spawn point and pedestrian speed. For training, we consider pedestrian speed in the range of $0.6 - 2.0$ m/s with a step size of 0.1 and pedestrian distance from 0 to 40 m with a step size of 1. For testing, we consider pedestrian speed in the $0.25 - 2.85$ m/s range and pedestrian distance in the $4.75 - 49.25$ m range. As a result, there were 3690 scenes in the training set and 9936 scenes in the test set.

Classical baseline

In this study, we evaluate the performance of Nav-Q in comparison to its classical counterpart, NavA2C, which draws inspiration from A2C-CADRL-p [5] (Sect. 3.3). Notably, we introduce several enhancements that aim to improve its performance compared to earlier iterations.

One key modification involves the incorporation of layer normalization [30] into the neural network architecture of the feature encoder (E_ψ), actor (π_θ), and critic (V_ϕ). This addition was absent in the original A2C-CADRL-p model [5]. The inclusion of layer normalization is intended to promote more stable and expedited training. Furthermore, in the initial version of A2C-CADRL-p, the LSTM produced a 256-dimensional vector ($h_t \in \mathbb{R}^{256}$). Given the challenges of encoding such high-dimensional vectors into a quantum state, we made adjustments to E_ψ and LSTM to produce a 32-dimensional vector instead. Additionally, we fine-tuned the reward function to facilitate more consistent and stable learning (see Sect. 3.3).

Since the critic is the main difference between classical and quantum architectures, we fine-tuned the number of hidden neurons in the classical architecture, directly impacting the number of parameters (see Appendix C for results).

It is worth noting that, in the realm of CFN for self-driving cars, our study is pioneering in its investigation of hybrid quantum-classical reinforcement learning algorithms. As a result, there are no quantum baselines to compare with, as we are the first to delve into this domain.

Implementation

To implement the parameterized quantum circuit in the critic of Nav-Q, we conducted experiments using 1, 2, 4, and 6 qubits, varying the number of layers from 1 to 3 to find the best configuration (see Appendix B for more details). We followed a similar approach to find the best classical critic (Appendix C). Once the best quantum and classical models were identified, we trained Nav-Q and NavA2C for 10,000 episodes to assess the performance in terms of training policy of both approaches. To evaluate convergence rate and training stability, we run five different weight initializations of the overall architecture, training for up to 5000 episodes and analyzing the average training behavior.

For both NavA2C and Nav-Q, gradients were computed after each episode, and the network weights were updated using the Adam optimizer with a learning rate of 0.0005. Each episode had a maximum duration of 500 time steps and ended prematurely if the car reached its goal or collided with any obstacle. Given the challenge of encoding high-dimensional classical data into a quantum state, we set the output of the LSTM layer for both Nav-Q and NavA2C to produce $h_t \in \mathbb{R}^{32}$, which was initially a 256-dimensional vector in A2C-CADRL-p [5] (the inspiration for NavA2C).

In the case of Nav-Q, we utilized the "default.qubit" device of PennyLane for quantum simulation and employed backpropagation with gradient descent to update the weights of the entire model.

It is important to note that using backpropagation for training quantum circuits would not be feasible on quantum hardware [31, 32]. To estimate the performance as if we had a real quantum device, we also trained Nav-Q using the parameter-shift rule [31], an alternative method to compute partial derivatives of quantum expectation values with respect to gate parameters on quantum hardware. However, gradient calculation using the parameter-shift rule is computationally expensive because it requires two circuit evaluations per parameter—one with a positive shift and one with a negative shift—to compute the partial derivative. Due to the extensive computational time required on quantum simulators, we simplified the problem by reducing complexity: we decreased the LSTM output dimension from $h_t \in \mathbb{R}^{32}$ to $h_t \in \mathbb{R}^6$ and used a PQC with only 2 qubits and 1 layer. Additionally, we limited the training to 2500 episodes on the initial scenario of the CARLA-CTS1.0 benchmark. Although this configuration is suboptimal, it allows us to investigate how Nav-Q would perform in the presence of noise in a scenario where a real quantum device is used.

Finally, we also trained and tested the aforementioned simplified version of Nav-Q under two simulated noise models,

quantum gate error [33] and depolarising error [34]. To simulate quantum gate error, we add uniform random noise to the parameters of rotation gates such that $\theta \leftarrow \theta + 0.01 \times \theta \times \delta$, where $\delta \sim \text{Uniform}(0, 1)$. This kind of error arises due to imperfection in the realization of rotation gates on quantum hardware. Depolarising error represents a random error in between each operation on a qubit. These errors are injected in the form of bit-flips (x error), phase-flips (z error), or both at the same time (y error). We use the in-built library of Pennylane [18] to simulate this error.

Metrics

To assess the performance Nav-Q with the classical baselines, we define three categories of metrics:

Trainability: This aspect refers to the capacity of an RL model to be effectively trained to perform a specific task or acquire a particular behavior within the environment. A conventional method for evaluating trainability involves analyzing the *Return vs. Episodes* curve, illustrating the smoothed cumulative reward (i.e., *return*) as the number of episodes increases. The smoothing procedure is carried out using Polyak-Ruppert averaging with weight set to 0.995 (Appendix E). Furthermore, we calculate several descriptive statistics of the area under the curve (AUC) [8, 9] to assess the stability and convergence rate of different architectures across various initializations.

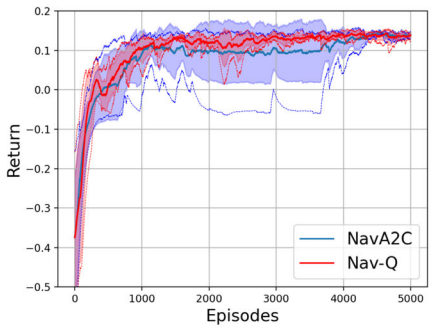
Driving policy: The performance in terms of driving policy is measured using the following metrics: (a) the overall safety index (SI) defined as the total number of scenarios in which the method is below given percentages of crashes and near misses and (b) the crash and near-miss rates (%) and time to goal (TTG) in seconds [5].

Learnability: To methodologically assess the ability of both quantum and classical models to learn complex and diverse functions beyond the immediate problem statement, we employ the calculation of the effective dimension [35, 36]. This complexity measure, rooted in information geometry, estimates the size of the space containing all possible functions for a particular model class, utilizing the Fisher information matrix (FIM) as the metric. This approach, however, requires assuming that the conditional distribution of the target variable of interest is Gaussian. Moreover, the spectral analysis of the FIM enables the examination of the curvature of loss surfaces, providing insights into the nature of loss surfaces for both classical and quantum models [36]. Indeed, the shape of the optimization landscape plays a crucial role in understanding whether a specific model can effectively find a suitable solution to the problem (the definition and algorithm for calculating the FIM and the ED are detailed in Appendix A).

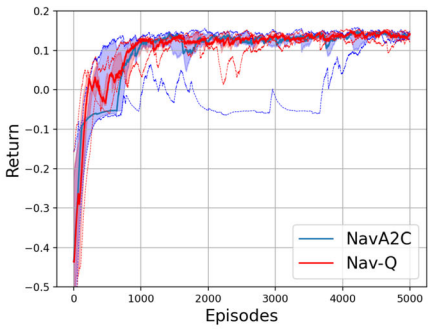
5.2 Results

Trainability

The selection of the best architecture for both quantum and classical models is described in Appendices C, B. Thus, we compared the results of the best configuration of Nav-Q and NavA2C to assess the convergence rate and training stability. Summary of the results for five different training sessions, each with distinct initialization weights are shown in Fig. 7 (individual results in Appendix D). This experiment evaluates two critical aspects: the average convergence rate (performance improvement as the number of episodes



(a) **Return Distribution with Mean and Standard Deviation**. The thick solid line represents the mean, the colored shaded area represents the standard deviation and the thin dotted line represents the maximum and minimum return values.



(b) **Return Distribution with Median and Interquartile Range**. The thick solid line represents the median, the colored shaded area represents the IQR and the thin dotted line represents the maximum and minimum return values.

Fig. 7 Comparison of the smoothed *Reward vs. Episodes* curves for Nav-Q with 4 qubits and 2 layers, and NavA2C averaged of five runs. The red-colored plots in the graph correspond to Nav-Q while the blue colored plots refer to NavA2C. Descriptive statistics of the AUC are provided in Table 2. In some cases, the mean ± standard deviation exceeds the maximum or falls below the minimum due to the asymmetry of the distribution of the return values at a given episode

Table 2 Summary statistics of the AUC (mean, median, standard deviation, and interquartile range) over 5 runs comparing Nav-Q (quantum) and NavA2C (classical) models. Smoothed return curves are shown in brackets, and unsmoothed values are presented without brackets

Method	Mean	Median	Standard deviation	Interquartile range (IQR)
Nav-Q	**550.83** (**448.98**)	**558.14** (**466.74**)	**561.09** (**484.03**)	**20.02** (**44.84**)
NavA2C	461.19 (359.75)	515.16 (384.90)	593.70 (536.47)	230.42 (220.04)

increases) and stability (sensitivity of the two models to different weight initializations in the neural networks).

The trend of the curves remains consistent for both quantum and classical models, suggesting similar overall performance in terms of driving policy, as evidenced by the resemblance in average rewards between the two approaches.

To evaluate convergence and stability quantitatively, we calculate the mean, median, standard deviation, and interquartile range (IQR) of the AUC of the *Return vs. Episode* curve across the five runs. A higher mean and median AUC suggest better convergence, while a lower standard deviation and IQR indicate enhanced stability (Table 2).

Nav-Q reports a slightly higher mean and median AUC than NavA2C. Upon examining variability measures such as the standard deviation and the IQR, Nav-Q demonstrates lower variability compared to NavA2C. While these results are based on only five runs and may not be generalizable, they suggest that the quantum model exhibits a higher convergence rate and greater stability compared to its classical counterpart.

Driving policy

We trained the models that achieve high convergence after 5000 (though results are very similar) episodes for Nav-Q and NavA2C till 10, 000 episodes and evaluated the performance in terms of driving policy. Results are shown in Table 3.

Both approaches seemed to have learned a similar policy. Nav-Q has a slightly lesser mean time to goal compared to NavA2C, which implies that the model trained using a quantum critic reaches the goal faster compared to the fully classical approach. When looking at collisions (crashes and near misses), the classical approach slightly outperforms Nav-Q, providing a lower percentage for crashes and near misses. Still, the results provided by the two models are consistent with other state-of-the-art approaches tested on similar data [5]. However, the SI—defined in the range [1, 8] that quantifies the total number of scenarios in which the agent exhibits the number of crashes and near misses below 20%—is the same. In general, the differences in terms of driving policy seem to be not very significant when comparing Nav-Q with NavA2C. This implies that the advantages provided by Nav-Q in terms of convergence rate and stability come without affecting the performance when considering the behavior of the car during the test phase.

Finally, the last column of Table 3 reports the number of days required to train the two models. As observed, while the quantum-supported architecture reduces complexity in terms of the number of parameters, the training times are notably longer. This suggests that the current optimization strategy, which includes the forward pass-requiring the simulation of the full quantum circuit after each update-and standard backpropagation, which is usually well-suited for classical neural networks, might require custom amendments to perform optimally when dealing with quantum-supported solutions on real-world data. Nevertheless, training times are not a good metric for comparing classical and quantum models when using quantum simulation, which is intrinsically hard for classical computation. Instead, to make a fair comparison, which is also consistent with standard approaches in classical RL, *Return vs. Episodes* curve is used.

Learnability

The normalized effective dimension (ED) [36], denoted by $\overline{d}_{\gamma,n} = \frac{d_{\gamma,n}}{d}$, provides the proportion of active parameters relative to the total number of parameters in the model. A high normalized effective dimension suggests that the model is utilizing a substantial portion of the model space, with many parameters actively involved in fitting the data. Conversely, a low effective dimension indicates that the model is using only a limited part of the model space, with a smaller proportion of parameters actively contributing to the model's performance. Results comparing Nav-Q and NavA2C in terms of ED are shown in Table 4.

We observed that $\overline{d}_{\gamma,n}$ of Nav-Q is 10 times higher than NavA2C. This implies that the quantum critic has potentially a greater capacity to model complex functions compared to

Table 3 Comparison of driving policy of NavA2C and Nav-Q with 4 qubits and 2 layers after 10,000 episodes of training

Method	#Parameters	Time to goal (s)	Crashes (%)	Near misses (%)	Safety index (SI)	Mean return	Training time (days)
Nav-Q	53	**11.74**	13.23	23.62	**3**	0.10	10.87
NavA2C	2305	11.77	**12.47**	**22.39**	**3**	**0.12**	**2.7**

The reported number of parameters (#parameters) includes only those implemented for the critic, which is the only difference between the classical and quantum architectures. The parameters shared by both approaches are as follows: feature encoder (2,828,576), LSTM (8960), and actor (2435)

Table 4 Comparison of all the configurations tested for Nav-Q and NavA2C in terms of the number of parameters, and the effective dimension

Method	#Qubits	#Layers	#Parameters	ED ($d_{\gamma,n}$)	Normalized ED $\overline{d}_{\gamma,n} = \frac{d_{\gamma,n}}{d}$
Nav-Q	4	2	**53**	10.79	**0.20**
NavA2C	NA	NA	2305	56.65	0.02

Notice that the reported number of parameters includes only those implemented for the critic, which is the only difference between the classical and quantum architectures. The trainable parameters shared by both approaches are as follows: feature encoder (2,828,576), LSTM (8960), and actor (2205)

the classical critic. Understanding the capacity of a quantum model can be advantageous as it provides insight into the model's capability to learn patterns even before training, which is a resource-intensive task.

In addition, we analyze the eigenspectrum of the FIM to gain insights on the shape of the optimization landscape. Figure 8 shows the eigenvalue distribution of FIM obtained for the critics of Nav-Q and NavA2C.

We arrange the eigenvalues in descending order and plot the top 50 eigenvalues for both cases. In the case of NavA2C, a few exceptionally high eigenvalues are observed, with the majority clustering near zero. This suggests that the loss curvature is steep along the eigenvector corresponding to the highest eigenvalue. Given that most eigenvalues are close to zero, we can infer that the loss surface is primarily flat, featuring minor ridges and valleys. For Nav-Q, similar to NavA2C, a single prominent eigenvalue is noticeable, with the majority of the eigenvalues clustered close to zero. This phenomenon results in extremely small gradients, thereby slowing down optimization via gradient descent. This issue resembles the vanishing and exploding gradient problem encountered in classical neural networks. This observation diverges from the findings in [36], where it is claimed that the eigenvalues in the case of PQCs should be more evenly spread. We speculate that the anomaly may be attributed to the benchmark design. However, we acknowledge that further investigation is required to fully understand this aspect.

Simulated noisy environment and parameter-shift rule

All the previously described experiments utilized classical backpropagation for optimizing the parameters in Nav-Q. However, this approach is only feasible when using quantum simulation and would be impractical when dealing with real quantum devices. Furthermore, real near-term quantum devices are affected by noise which introduces errors and limits the coherence of quantum states, impacting the accuracy and reliability of quantum computations. To assess these two aspects, we conducted training procedures using the parameter-shift rule and introducing noise on a reduced version of Nav-Q ($h_t \in \mathbb{R}^6$), with a PQC that has only 2 qubits and 1 layer, and trained on the first scenario of the CARLA-CTS 1.0 benchmark for 2500 episodes.

Figure 9 compares the different approaches using the *Return vs. Episodes* and *Entropy vs. Episodes* curves of NavA2C and Nav-Q.

Nav-Q trained using backpropagation and NavA2C exhibit similar performance, consistent with previous experiments. However, when training Nav-Q using the parameter-shift rule, the convergence rate is considerably slower than with backpropagation. In this regard, the entropy curve indicates that the parameter-shift rule is causing the problem of overshooting due to a large step size, as the entropy curve keeps oscillating around the value of 0.4.

When introducing noise, we observe a deterioration in the stability of learning. The convergence rate of Nav-Q trained with the parameter-shift rule and introducing noise is almost the same as without noise in the initial part of the training. However, the problem of overshooting for the noisy model becomes even more pronounced, as evident from its entropy curve. Therefore, we can conclude that, although the parameter-shift rule enables precise calculation of the gradient at each iteration on a methodological level, the performance remains below that of using backpropagation.

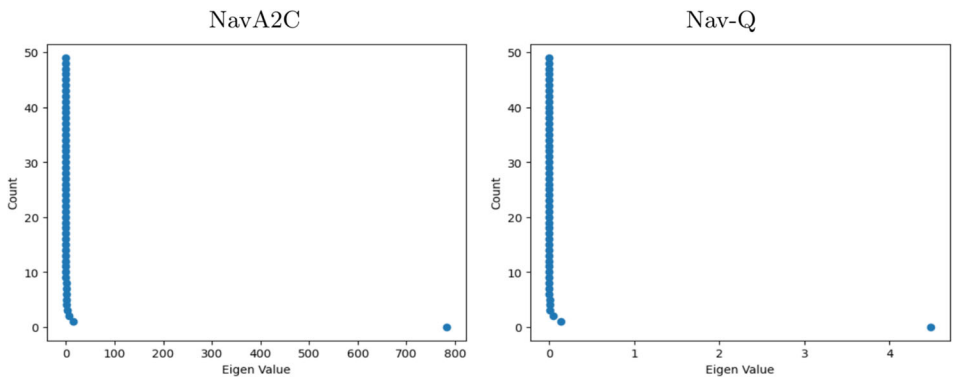


Fig. 8 Eigenvalue distribution of Fisher information matrix for the critic of NavA2C (right) and Nav-Q (left)

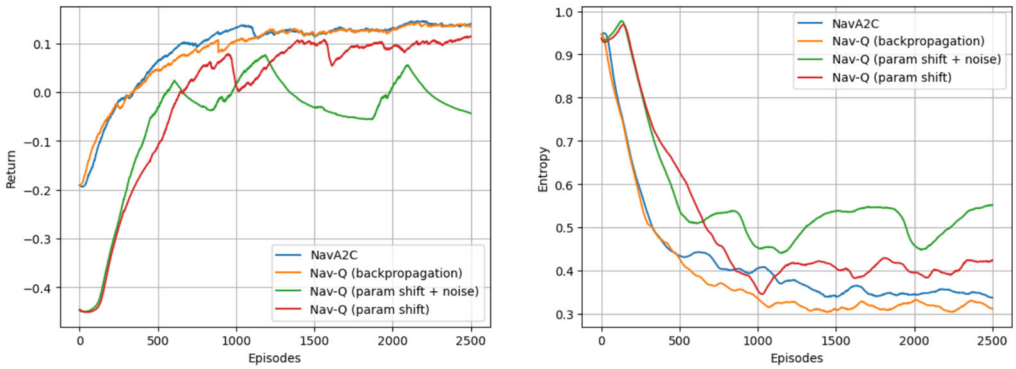


Fig. 9 Training performance is evaluated using the *Rewards vs Episodes* curve (left) and the *Entropy vs Episodes* curve (right), with each curve representing results from a single run for each configuration. Both graphs display results for four different approaches: NavA2C as the classical baseline, and two variants of Nav-Q depending on the optimization strategy used-classical backpropagation or the parameter-shift rule. Additionally, one variant of Nav-Q is trained with different noise models integrated into the circuit simulation, using the parameter-shift rule for gradient calculation

This could be attributed to the need to set the magnitude of the gradient shift at each step in advance. Additionally, as expected, introducing quantum noise leads to a deterioration in performance in terms of both stability and convergence rate. Nevertheless, from the *Entropy vs. Episodes* curve, we observe that due to noise, the RL agent exhibits a higher exploratory nature since the value of entropy keeps oscillating even after a high number of episodes.

6 Discussion and conclusion

In this work, we introduced Nav-Q, the first quantum-supported deep reinforcement learning approach designed to address the challenge of collision-free navigation for self-driving cars. Nav-Q is built on an actor-critic algorithm to effectively learn an optimal policy for controlling the speed of self-driving cars. The critic is a hybrid quantum-classical algorithm constructed using an innovative technique named Qubit Independent Data Encoding and Processing (QIDEP), allowing the quantum embedding of high-dimensional classical data without being constrained by the number of qubits. Consequently, QIDEP provides the flexibility to adapt the number of qubits to suit specific quantum circuit requirements. Moreover, Nav-Q represents a valid alternative to existing classical algorithms, as its architecture design does not require a quantum processing unit at test time, enhancing its usability for real-world applications.

To the best of our knowledge, Nav-Q is the first proposal of its kind, introducing a quantum component designed to enhance training performance without requiring a QPU during testing. This approach can be easily adapted to address typical problems utilizing the actor-critic model, particularly those struggling with trainability and stability during the training phase, and can derive benefits from quantum machine learning models.

We trained Nav-Q using the CARLA-CTS1.0 benchmark [5] and systematically compared its performance with the classical baseline, NavA2C. Regarding trainability, the results show that, on average, Nav-Q exhibits a slightly superior average return and better stability compared to NavA2C. Specifically, when running both approaches with different weight initializations, Nav-Q demonstrated a higher mean and median AUC of the *Return vs Episodes* curves than NavA2C, along with a lower standard deviation and interquartile range, which suggests better stability. Considering the two models when trained for 10, 000 episodes, we evaluated their performance in terms of the driving policy of the car. In this regard, Nav-Q and NavA2C showed similar performance, with Nav-Q tending to achieve the final goal faster, while NavA2C was slightly better in terms of collision avoidance.

In terms of learnability, Nav-Q exhibited superior performance when considering the effective dimension and showed similar behavior when examining the distribution of eigenvalues of the Fisher Information Matrix. In this context, the quantum model appears to be significantly impacted by the barren plateau problem, making the optimization procedure challenging in finding a suitable local minima.

However, these results only hold for the specific experimental setting considered and cannot be generalized. Still, they provide a good starting point for testing quantum models for real-world classical applications. Future works should explore a broader range of settings and optimization strategies to better understand the potential and limitations of quantum models in various practical scenarios.

Finally, we conducted smaller-scale training experiments with a 2-qubit, 1-layer Nav-Q using a suitable optimization

procedure for real quantum hardware, known as the parameter-shift rule. We investigated the impact of noise using two distinct noise models: the Quantum Gate Error [33] and Depolarising Error [34]. The comparison of the resulting *Return vs. Episodes* curves clearly illustrates that the trainability of Nav-Q was adversely affected by the combination of the parameter-shift rule and noise, highlighting the challenges posed by these factors on near-term quantum hardware.

For this reason, a primary challenge to address in the near future is overcoming the issue of barren plateaus. Various approaches can be considered, including different parameter initialization techniques [37], layerwise learning [38], and parameter correlations [39]. Another interesting direction is to design a quantum PPO architecture based on the principles of Nav-Q. In this case, it will be possible to leverage the efficiency of PPO in terms of more stable learning and faster convergence rate, while facilitating the training procedure through quantum computation.

Furthermore, a promising avenue involves extending the application of the Nav-Q architecture to diverse problems beyond the CFN of self-driving cars. The Nav-Q algorithm can be applied to problems requiring an RL agent challenging to train using classical neural networks alone. As observed, even a relatively small quantum circuit possesses greater descriptive power than a large neural network. Identifying specific applications where training can be accelerated using quantum computation is crucial for both reinforcement learning and quantum computing.

Appendix A: Fisher information matrix

In this section, we define and describe how to empirically calculate the Fisher information matrix (FIM) and the effective dimension (ED).

Definition 5 (Fisher information matrix) *For a dataset* $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$ *and a model with parameters* θ*, where* y_i *is a continuous target variable, the Fisher information matrix* ***(F)*** *is given by the following:*

$$F_{ij} = \mathbb{E}_{(\mathbf{x},y)\sim\mathcal{D}}\left[\left(\frac{\partial \log p(y|\mathbf{x};\theta)}{\partial\theta_i}\right)\left(\frac{\partial \log p(y|\mathbf{x};\theta)}{\partial\theta_j}\right)\right]. \tag{A.1}$$

For continuous target variables, a common choice for the conditional distribution is Gaussian, leading to the following likelihood function:

$$p(y|\mathbf{x};\theta) = \frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{(y - f(\mathbf{x};\theta))^2}{2\sigma^2}\right), \tag{A.2}$$

where $f(\mathbf{x};\theta)$ *is the continuous function and* σ *is the standard deviation of the Gaussian noise.*

When we take the logarithm of the likelihood function and ignore constant terms, we arrive at the negative mean squared error (MSE) loss [40]*:*

$$\log p(y|\mathbf{x};\theta) \propto -\frac{1}{2\sigma^2}(y - f(\mathbf{x};\theta))^2 \propto -MSE. \tag{A.3}$$

This implies that optimizing the model's parameters to maximize the log-likelihood is equivalent to minimizing the MSE loss, up to a constant factor.

As a result, the derivatives of the log-likelihood with respect to the model parameters are proportional to the derivatives of the MSE loss, leading to the equivalence in their gradients:

$$\frac{\partial \log p(y|\mathbf{x};\theta)}{\partial\theta_i} \propto \frac{\partial MSE}{\partial\theta_i}. \tag{A.4}$$

Thus, the Fisher information matrix derived from the log-likelihood can also be interpreted in terms of the derivatives of the MSE loss, facilitating the analysis of the curvature of the loss surface using the FIM framework.

The first step to calculate the effective dimension is calculating the Fisher information matrix (FIM) (see Definition 5).

Algorithm 1 Calculate fisher information matrix.

Require: Number of episodes $N \in \mathbb{R}^+$, Number of steps per episode $T_{\max} \in \mathbb{R}^+$, Costmap of the environment (see Definition 3.1)
function CALCULATEFIM(θ, ψ)
  Input: pre-trained weights for Actor π_θ, Feature Encoder E_ψ, and Critic V_ϕ
  Output: Fisher Information Matrix $F \in \mathbb{R}^{N\times d\times d}$
  for e $\leftarrow$ 1 to N **do**
    $\phi \leftarrow RandomUniformDistribution(min = -0.5, max = 0.5)$ ▷ Random initialisation of weights of V_ϕ
    $t \leftarrow 1$
    while $t \le T_{\max} \wedge \neg goal \wedge \neg accident$ **do**
      $path_t \sim$ AnytimeWeightedHybridAStar(p_{c_t}, $p_{c_{\text{goal}}}$, costmap)
      Steering angle α_t associated with $path_t$
      acc_t, $V_t \leftarrow$ RL_Model(o_t) ▷ V_t = Value at state s_t
      $a_t \leftarrow (\alpha_t, acc_t)$
      $o_{t+1} \leftarrow$ Carla_step(a_t)
      $R_{t+1} \leftarrow$ Reward(s_{t+1}, a_t)
      $t \leftarrow t + 1$
    end while
    Get set of transitions $\{(o_t, a_t, V_t, R_{t+1}, o_{t+1})\}$ for episode e
    Calculate discounted returns, G_t for each timestep using transitions for episode e ▷ (See Definition 3)
    $L_e(t) = MSE\_Loss(G, V)$ ▷ $L_e \in \mathbb{R}^{T_{max}}$
    $\Delta\phi_e(t) = CalculateGradients(L_e, \phi)$ ▷ $\Delta\phi_e \in \mathbb{R}^{T_{max}\times d}$
    $F_e(t) = \Delta\phi_e \bigotimes \Delta\phi_e$ ▷ $F_e \in \mathbb{R}^{T_{max}\times d\times d}$
    $\overline{F}_e = \frac{1}{T_{max}} \cdot \int_1^{T_{max}} F_e(t)dt$ ▷ $\overline{F}_e \in \mathbb{R}^{d\times d}$
  end for
  return F ▷ $F \in \mathbb{R}^{N\times d\times d}$
end function

Algorithm 1 highlights the algorithm to calculate the FIM. For a fair comparison of the two approaches, we fix the weights of the actor π_θ and feature encoder E_ψ while calculating the F for critics of NavA2C and Nav-Q. We chose $N = 50$ and $T_{max} = 50$ in this work.

Next, we calculate the effective dimension ($d_{\gamma,n}$) (see Definition 6). We take $\gamma = 1$ and number of data samples (n) corresponds to the sequence length of each trajectory (50). The integral of the parameter space of the critic is estimated to be equal to 1 as all the parameters vary between -0.5 and 0.5.

Definition 6 (Effective dimension (ED)) *The effective dimension of a statistical model* $M_\Theta = \{p(\cdot,\cdot;\theta) : \theta \in \Theta\}$ *concerning a parameter space* $\Theta \subset [-0.5, 0.5]^d$ *of dimension* d*, a constant* $\gamma \in (0, 1]$*, and a positive integer* $n > 1$ *representing the number of data samples, is defined as follows:*

$$d_{\gamma,n}(M_\Theta) = \frac{\log\left(\frac{1}{V_\Theta}\int_\Theta \sqrt{det(I + \frac{\gamma n}{2\pi\log(n)}\hat{F}(\theta))}d\theta\right)}{\log(\frac{\gamma n}{2\pi\log(n)})}$$

where $n > 1 \in \mathbb{N}$ *is the number of data samples,* $V_\Theta = \int_\Theta d\theta$ *is the parameter space volume, and* $\hat{F}(\theta)$ *is the normalized FIM defined as follows:*

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

and $F_{ij}(\theta)$ *(See* Definition 5*) is the Fisher information matrix calculated for the critic of Nav-Q and NavA2C using Algorithm 1.*

Appendix B: Best Nav-Q critic architecture

To choose the optimal architecture for Nav-Q and compare it with the classical baseline, we ran various configurations of the PQC for the critic, varying the number of qubits (1, 2, 4, and 6) and layers (1, 2, 3), and assessed the convergence rate and stability in terms of the *Return vs. Episode* curve. Results are shown in Fig. 10.

We observe that, in almost all cases, the model converges after 10,000 episodes, yielding comparable final return value, despite having varying numbers of parameters. This aligns with a well-known observation in the classical domain: an increase in the model's complexity does not necessarily guarantee a performance improvement. However, when using a critic with 1, 2, or 6 qubits, the training performance becomes unstable, with a sudden drop in cumulative reward (*return*) requiring hundreds of episodes to recover an increasing trend. This instability is not observed with a 4-qubit circuit using one or two layers.

In addition, we present a comparison of training times in terms of the number of days and the number of parameters for various configurations of Nav-Q (Table 5). The training times show an increasing trend in the number of qubits and layers, posing a significant challenge and extending to

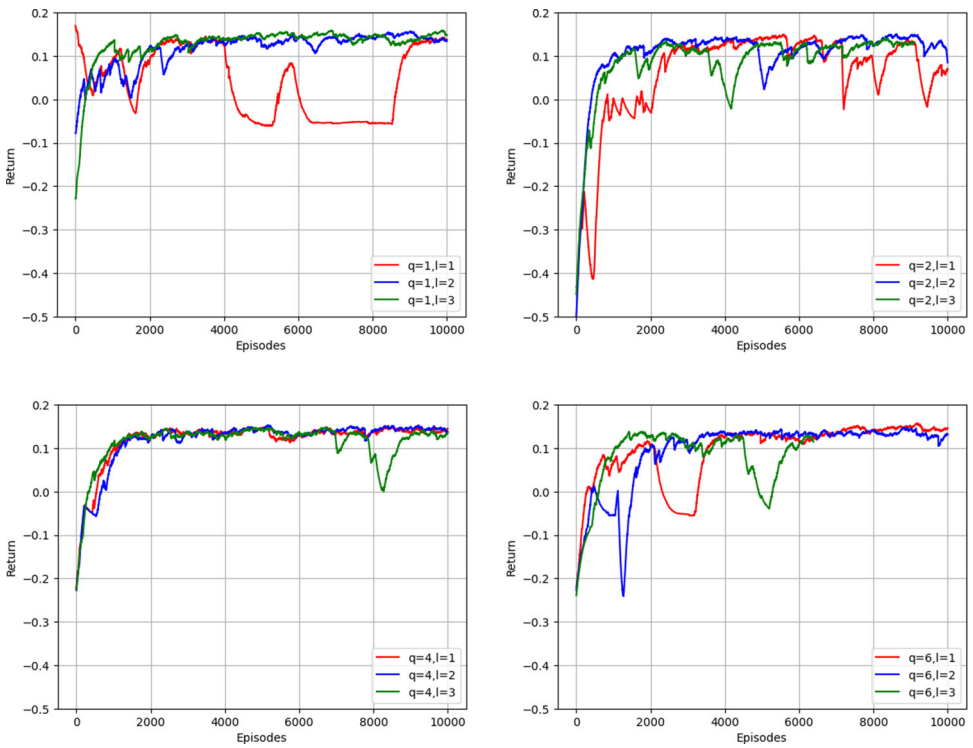


Fig. 10 *Return vs. Episodes* smoothed curves of Nav-Q with 1, 2, 4, 6 qubits (q), each with a different number of layers (l). Each line represents a single run

Table 5 Training times and number of parameters for different critic architectures of Nav-Q

Method	#qubits	#Layers	#Parameters (d)	Training time (days)
Nav-Q	1	1	24	5.78
		2	46	7.66
		3	68	10.50
	2	1	27	8.36
		2	51	9.64
		3	75	11.04
	4	1	29	6.8
		2	53	10.87
		3	77	14.64
	6	1	31	8.14
		2	55	11.74
		3	79	18.46

several days for each model. Nevertheless, when addressing real-world problems, such as the CFN benchmarked on standard classical data, testing a quantum-supported model remains feasible, albeit challenging. Considering the results of the 4-qubit architecture with 1 and 2 layers, we conducted an analysis using five different weight initializations. The results, encompassing mean, standard deviation, and range (max/min) are shown in Fig. 11.

Despite the similarity in average performance in terms of return for both architectures, we observe that the 1-layer model exhibits less stability compared to its 2-layer counterpart. Given that stability, along with average reward, is a crucial factor in analyzing reinforcement learning performance, we choose as a final architecture for the Nav-Q critic using 4 qubits and 2 layers as the best model.

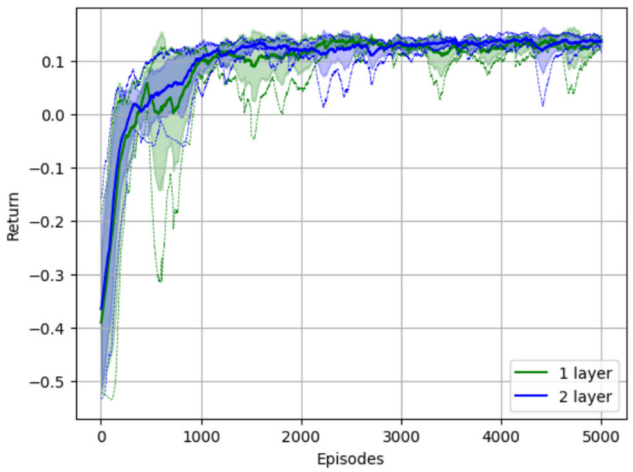


Fig. 11 Comparison of *Return vs. Episodes* graph of two different 4-qubit Nav-Q architectures with 1 and 2 layers, respectively, executed for five different parameters initialization. Notice that, the reported number of parameters (d) includes only those implemented for the critic, which is the only difference between the classical and quantum architectures. The parameters shared by both approaches are as follows: feature encoder (2,828,576), LSTM (8960), and actor (2205)

Appendix C: Best NavA2C critic architecture

In this section, we explore different architectures for the classical critic component within NavA2C. The other components of the NavA2C architecture (Sect. 5.1), including the feature encoder (2,828,576 parameters), LSTM (8960 parameters), and actor (2435 parameters), are adapted from existing works [5].

We tested three different critic architectures to determine the impact of parameter count on performance. The first critic architecture had approximately the same number of parameters as the quantum critic, consisting of 33 parameters. Given the limited performance in terms of convergence and stability of the *Return vs Episode*, a second critic with 1039 parameters has been trained. Finally, the third model had 2305 parameters. Results are shown in Fig. 12.

Our observations indicate that the first two configurations, with fewer parameters, exhibited poor behavior and stability in their learning curves. Only the critic architecture with 2305 parameters demonstrated good training behavior, reflected in a stable and steadily improving return.

Appendix D: Individual runs—Nav-Q vs NavA2C

Figure 13 illustrates the performance of two models—NavA2C (classical) and Nav-Q (quantum)—over five dif-

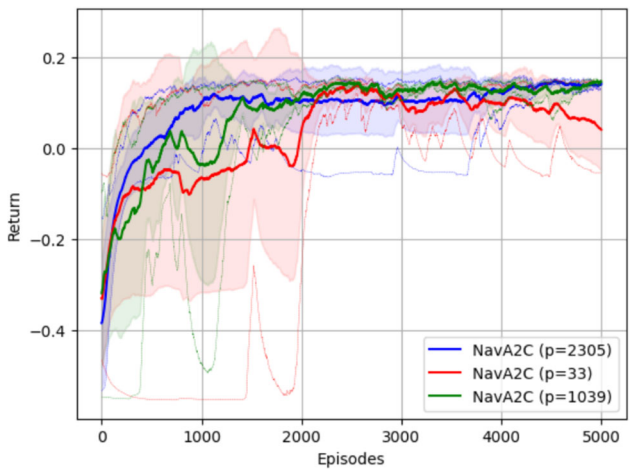


Fig. 12 The plot shows the performance of different critic architectures within the NavA2C framework. The thick lines represent the average return over five different initializations for each model. The shaded areas correspond to the mean plus the standard deviation, while the dotted lines indicate the maximum and minimum returns observed

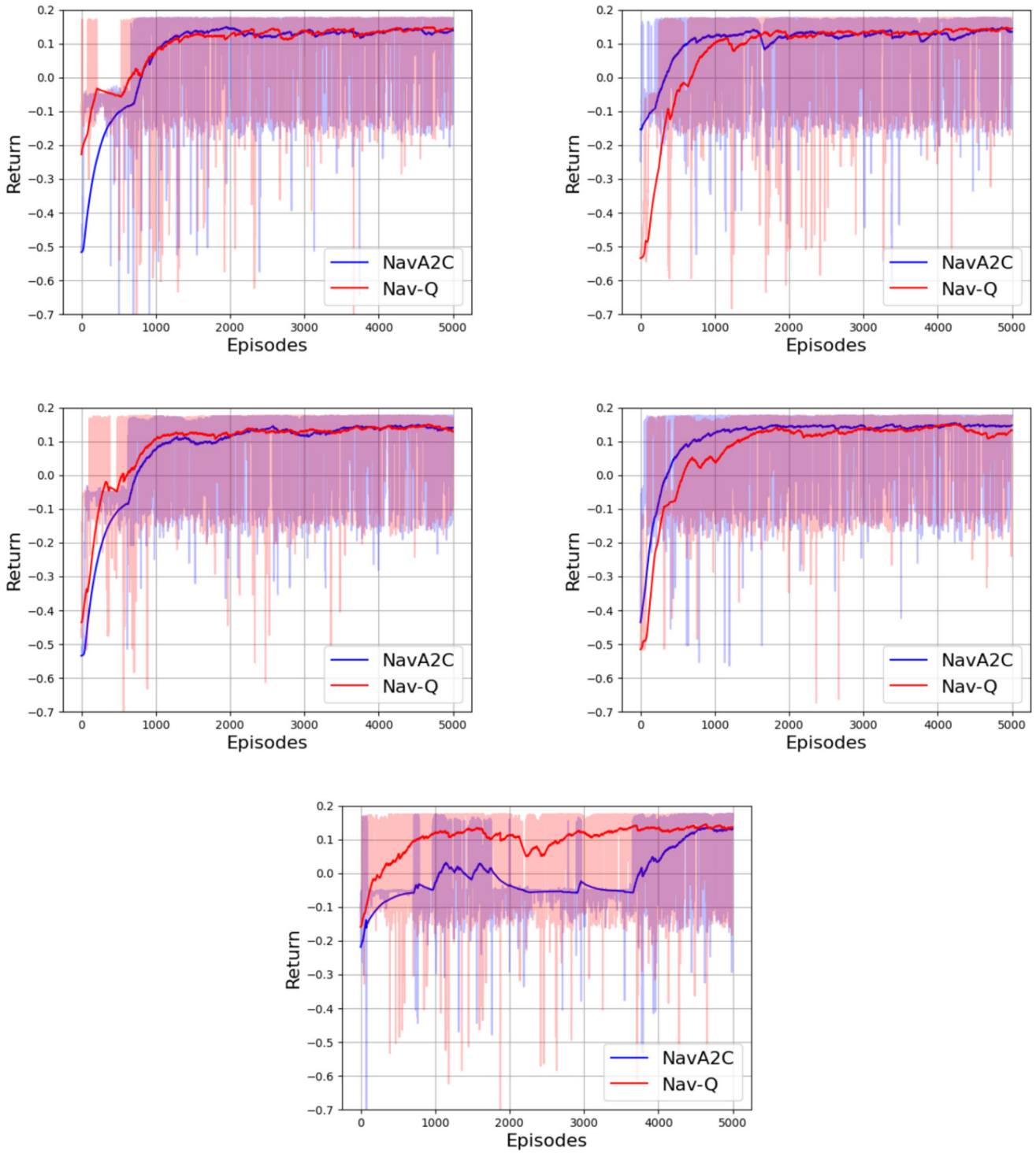


Fig. 13 Comparison of return curves across five different runs for the classical (NavA2C) and quantum (Nav-Q) models. Both models demonstrate similar performance trends, with the return improving and stabilizing throughout 5000 episodes

ferent runs. The return curve for both models shows similar behavior across the runs, with the return gradually improving as the number of episodes increases. Although the quantum model (Nav-Q) tends to exhibit a slightly faster rise in three out of five cases, the overall trend across both models converges to comparable performance. The shaded areas represent unsmoothed values indicating some degree of fluctuation.

Appendix E: Smoothing procedure

In order to detect the underlying performance of an RL method by looking at the return curve by reducing noise and volatility, a standard approach is used to smoothen the raw data points.

Algorithm 2 Polyak-Ruppert Averaging.

Require: Time series data $\{x_1, x_2, \ldots, x_N\}$
Require: Smoothing factor α where $0 < \alpha < 1$
1: Initialize smoothed value $s_1 = x_1$
2: **for** $t = 2$ to N **do**
3: Update smoothed value: $s_t = \alpha \cdot s_{t-1} + (1 - \alpha) \cdot x_t$
4: **end for**

Algorithm 2 shows the procedure Polyak-Ruppert averaging [41] applied to a time series representing the return versus episode curve. The algorithm iteratively smooths the return values, aiming to reduce variance and highlight trends in the learning performance over episodes. In the experiments presented in this paper, the smoothing factor α is set to 0.995, providing a high degree of smoothing to emphasize long-term trends.

Author contribution A.S. and A.M. developed methodological formalism. A.M. and A.S. designed the experimental settings and performed the analytic calculations and numerical experiments. A.M. and A.S. wrote the manuscript. A.M. and M.K. supervised the project. All authors provided critical feedback and helped shape the research, analysis, and manuscript.

Funding This work has been funded by the German Ministry for Education and Research (BMB+F) in the project MOMENTUM.

Data availability All code to generate the data, figures, and analyses in this study is publicly available with detailed information on the implementation via the following repository: https://github.com/roboak/Nav-Q.

Declarations

Conflict of interest The authors declare no competing interests.

References

1. Zhu Z, Zhao H (2021) A survey of deep RL and IL for autonomous driving policy learning. IEEE Trans Intell Transp Syst 23(9):14043–14065
2. Chib PS, Singh P (2023) Recent advancements in end-to-end autonomous driving using deep learning: a survey. IEEE Trans Intell Veh 1–18. https://doi.org/10.1109/TIV.2023.3318070
3. Kiran BR, Sobh I, Talpaert V, Mannion P, Al Sallab AA, Yogamani S, Pérez P (2021) Deep reinforcement learning for autonomous driving: a survey. IEEE Trans Intell Transp Syst 23(6):4909–4926
4. Pusse F, Klusch M (2019) Hybrid online POMDP planning and deep reinforcement learning for safer self-driving cars. In: 2019 IEEE intelligent vehicles symposium (IV), pp 1013–1020. https://doi.org/10.1109/IVS.2019.8814125
5. Gupta D, Klusch M (2023) HyLEAR: hybrid deep reinforcement learning and planning for safe and comfortable automated driving. In: 2023 IEEE intelligent vehicles symposium (IV), pp 1–8. https://doi.org/10.1109/IV55152.2023.10186781
6. Jerbi S, Gyurik C, Marshall S, Briegel H, Dunjko V (2021) Parametrized quantum policies for reinforcement learning. Adv Neural Inf Process Syst 34:28362–28375
7. Lan Q (2021) Variational quantum soft actor-critic. arXiv:2112.11921
8. Chen SY-C, Yang C-HH, Qi J, Chen P-Y, Ma X, Goan H-S (2020) Variational quantum circuits for deep reinforcement learning. IEEE Access 8:141007–141024
9. Skolik A, Jerbi S, Dunjko V (2022) Quantum agents in the gym: a variational quantum algorithm for deep Q-learning. Quantum 6:720
10. Brockman G, Cheung V, Pettersson L, Schneider J, Schulman J, Tang J, Zaremba W (2016) OpenAI Gym. arXiv:1606.01540
11. Pérez-Salinas A, Cervera-Lierta A, Gil-Fuster E, Latorre JI (2020) Data re-uploading for a universal quantum classifier. Quantum 4:226
12. Mnih V, Badia AP, Mirza M, Graves A, Lillicrap T, Harley T, Silver D, Kavukcuoglu K (2016) Asynchronous methods for deep reinforcement learning. In: International conference on machine learning, pp 1928–1937. PMLR
13. Mirowski P, Pascanu R, Viola F, Soyer H, Ballard AJ, Banino A, Denil M, Goroshin R, Sifre L, Kavukcuoglu K et al (2016) Learning to navigate in complex environments. arXiv:1611.03673
14. Everett M, Chen YF, How JP (2021) Collision avoidance in pedestrian-rich environments with deep reinforcement learning. IEEE Access 9:10357–10377
15. Hochreiter S, Schmidhuber J (1997) Long short-term memory. Neural Comput 9(8):1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

16. Schulman J, Wolski F, Dhariwal P, Radford A, Klimov O (2017) Proximal policy optimization algorithms. arXiv:1707.06347
17. Tang Y (2019) Towards learning multi-agent negotiations via self-play. In: Proceedings of the IEEE/CVF international conference on computer vision workshops
18. Bergholm V, Izaac J, Schuld M, Gogolin C, Ahmed S, Ajith V, Alam MS, Alonso-Linaje G, AkashNarayanan B, Asadi A, et al (2018) PennyLane: automatic differentiation of hybrid quantum-classical computations. arXiv:1811.04968
19. Lockwood O, Si M (2020) Reinforcement learning with quantum variational circuit. In: Proceedings of the AAAI conference on artificial intelligence and interactive digital entertainment, vol 16, pp 245–251
20. Acuto A, Barillà P, Bozzolo L, Conterno M, Pavese M, Policicchio A (2022) Variational quantum soft actor-critic for robotic arm control. arXiv:2212.11681
21. Sutton RS, McAllester D, Singh S, Mansour Y (1999) Policy gradient methods for reinforcement learning with function approximation. Adv Neural Inf Process Syst 12
22. Geva S, Sitte J (1993) A cartpole experiment benchmark for trainable controllers. Control Syst IEEE 13:40–51. https://doi.org/10.1109/37.236324
23. Kwak Y, Yun WJ, Jung S, Kim J-K, Kim J (2021) Introduction to quantum reinforcement learning: theory and PennyLane-based implementation. In: 2021 International conference on Information and Communication Technology Convergence (ICTC), pp 416–420. IEEE
24. Kimura T, Shiba K, Chen C-C, Sogabe M, Sakamoto K, Sogabe T (2021) Variational quantum circuit-based reinforcement learning for POMDP and experimental implementation. Math Probl Eng 2021(1):3511029
25. Bar NF, Yetis H, Karakose M (2023) An efficient and scalable variational quantum circuits approach for deep reinforcement learning. Quantum Inf Process 22(8):300
26. Tschiatschek S, Arulkumaran K, Stühmer J, Hofmann K (2018) Variational inference for data-efficient model learning in POMDPs. arXiv:1805.09281
27. Kandala A, Mezzacapo A, Temme K, Takita M, Brink M, Chow JM, Gambetta JM (2017) Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets. Nature 549(7671):242–246
28. Dosovitskiy A, Ros G, Codevilla F, Lopez A, Koltun V (2017) CARLA: an open urban driving simulator. In: Conference on robot learning, pp 1–16. PMLR
29. Bartels B, Erbsmehl C (2014) Bewegungsverhalten von fußgängern im straßenverkehr, teil 1. FAT-Schriftenreihe (267)
30. Ba JL, Kiros JR, Hinton GE (2016) Layer normalization. arXiv:1607.06450
31. Mitarai K, Negoro M, Kitagawa M, Fujii K (2018) Quantum circuit learning. Phys Rev A 98(3):032309
32. Schuld M, Bergholm V, Gogolin C, Izaac J, Killoran N (2019) Evaluating analytic gradients on quantum hardware. Phys Rev A 99(3):032331
33. Rigetti Computing (2023) Quantum gate errors - PyQuil documentation. Website. https://pyquil-docs.rigetti.com/en/stable/noise.html#quantum-gate-errors
34. King C (2003) The capacity of the quantum depolarizing channel. IEEE Trans Inf Theory 49(1):221–229
35. Berezniuk O, Figalli A, Ghigliazza R, Musaelian K (2020) A scale-dependent notion of effective dimension. arXiv:2001.10872
36. Abbas A, Sutter D, Zoufal C, Lucchi A, Figalli A, Woerner S (2021) The power of quantum neural networks. Nat Comput Sci 1(6):403–409
37. Grant E, Wossnig L, Ostaszewski M, Benedetti M (2019) An initialization strategy for addressing barren plateaus in parametrized quantum circuits. Quantum 3:214
38. Skolik A, McClean JR, Mohseni M, Smagt P, Leib M (2021) Layer-wise learning for quantum neural networks. Quantum Mach Intell 3:1–11
39. Holmes Z, Sharma K, Cerezo M, Coles PJ (2022) Connecting ansatz expressibility to gradient magnitudes and barren plateaus. PRX Quantum 3(1):010313
40. Goodfellow I, Bengio Y, Courville A (2016) Deep learning. MIT press
41. Polyak BT, Juditsky AB (1992) Acceleration of stochastic approximation by averaging. SIAM J Control Optim 30(4):838–855