Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Learning to Build Quantum Kernels: A Reinforcement Learning Framework for Quantum SVC Optimization

Luigi Barbato
National Research Council
Giuseppe Buonaiuto
National Research Council
Lidia Marassi
University of Naples Federico II
Stefano Marrone
University of Naples Federico II
Carlo Sansone
University of Naples Federico II
Massimo Esposito
National Research Council
Francesco Gargiulo
francesco.gargiulo@cnr.it

National Research Council

Research Article

Keywords: Quantum Machine Learning, Reinforcement Learning, Support Vector Classifier, eXplainable Artificial Intelligence, Kernel Optimization

Posted Date: March 4th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-8722632/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

Learning to Build Quantum Kernels: A Reinforcement Learning Framework for Quantum SVC Optimization

Luigi Barbato[1,3], Giuseppe Buonaiuto [*], Lidia Marassi[2], Stefano Marrone[2], Carlo Sansone[2], Massimo Esposito[1], Francesco Gargiulo[1*]

[1]Institute for high performance computing and networking (ICAR) of Italian National Research Council (CNR), Naples, Italy.
[2]University of Naples Federico II (UNINA), Naples, Italy.
[3]University Campus Bio-Medico of Rome(UCBM), Rome, Italy.

*Corresponding author(s). E-mail(s): francesco.gargiulo@cnr.it;

Abstract

This work introduces a novel hybrid quantum-classical machine learning framework in which a Reinforcement Learning (RL) agent autonomously constructs quantum circuits to optimize kernels for a Quantum Support Vector Classifier (QSVC). Unlike traditional approaches that rely on fixed or manually designed quantum feature maps, the proposed method employs a Deep Q-Network (DQN) agent to dynamically select quantum gates and their parameters. These configurations define a parameterized quantum circuit whose fidelity-based kernel is used to perform classification tasks. The RL agent is trained to maximize classification accuracy by interacting with an environment that evaluates circuit performance. Experiments conducted on a realistic and potentially biased educational dataset demonstrate that the RL-optimized QSVC outperforms both classical SVC and standard QSVC baselines in terms of accuracy and fairness. Furthermore, SHAP-based explainability analysis reveals a reduction in model bias, suggesting improved interpretability and robustness. This study highlights the potential of reinforcement learning for automated quantum circuit design and kernel optimization in quantum machine learning.

Keywords: Quantum Machine Learning, Reinforcement Learning, Support Vector Classifier, eXplainable Artificial Intelligence, Kernel Optimization.

[*]author's current affiliation email: giuseppe.buonaiuto@fujitsu.com

1 Introduction

In recent years, advancements in artificial intelligence and quantum computing have paved the way for innovative solutions to complex problems. In particular, one of the many branches of AI, reinforcement learning, has shown remarkable effectiveness in solving sequential decision-making problems across vastly different domains, such as robotic control and portfolio gain maximization. Simultaneously, quantum computing has emerged as a revolutionary technology, with the potential to solve problems that classical computers struggle with, due to the computational complexity of the problems themselves. In this paper, an RL agent will be used not only for optimizing the circuit parameters, as already done in previous works, but also for identifying an optimal configuration for the quantum circuit, and creating from it a kernel used by a Support Vector Classifier (SVC) to perform classification tasks, demonstrating the superiority of the Quantum Support Vector Classifier (QSVC) over the classical SVC not only in terms of overall performance improvement but also in bias reduction.

The organization of the paper is as follows. **Section 1** introduces the topic. **Section 2** provides background and discusses relevant work connected to Quantum Machine Learning and the use of the shap values for XAI. **Section 3** describes the methodology, i.e., quantum kernel estimation, simulation environment, and reinforcement learning (RL) agent, with a deep description of how it is structured and how the problem was mapped to be seen as a RL Agent problem. **Section 4** presents experimental results comparing a baseline model with that of the quantum support vector classifier (QSVC). **Section 5** illustrates the results, with a focus on the explainable artificial intelligence (XAI) through SHAP analysis, including classification results and SHAP results.

2 Background

2.1 Reinforcement Learning Fundamentals

Reinforcement Learning (RL) represents a foundational paradigm in machine learning wherein an autonomous agent learns optimal decision-making strategies through iterative interaction with its environment [1]. Unlike supervised learning, which relies on labeled training examples, or unsupervised learning, which discovers hidden patterns in unlabeled data, RL addresses sequential decision problems where the agent must balance exploration of unknown states with exploitation of learned knowledge to maximize long-term cumulative rewards. This distinctive characteristic makes RL particularly well-suited for problems involving sequential decision-making under uncertainty, including robotic control, game playing, resource allocation, and, as demonstrated in this work, automated quantum circuit design.

The mathematical foundation of RL is provided by the Markov Decision Process (MDP) framework, which formalizes the interaction between agent and environment as a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$. The state space $\mathcal{S}$ encompasses all possible configurations the environment can assume; in our quantum circuit optimization context, states encode the current circuit structure and its classification performance characteristics. The

action space $\mathcal{A}$ contains all decisions available to the agent, which in our case correspond to quantum gate selections and their associated rotational parameters. The state transition probability $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ models the stochastic dynamics governing how actions transform states, while the reward function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ provides scalar feedback quantifying the immediate desirability of executing a particular action in a given state. Finally, the discount factor $\gamma \in [0, 1]$ controls the relative importance of immediate versus future rewards, with values approaching unity prioritizing long-term outcomes.

The central objective in RL is to derive an optimal policy $\pi : \mathcal{S} \to \mathcal{A}$ that maps states to actions in a manner maximizing the expected cumulative discounted return:

$$J(\pi) = \mathbb{E}_{\tau \sim \pi} \left[\sum_{t=0}^{\infty} \gamma^t r_t \right] \tag{1}$$

where τ represents a trajectory of state-action-reward sequences generated by following policy π, and r_t denotes the reward obtained at time step t. This formulation captures the essence of the credit assignment problem in RL: determining which actions in a sequence are responsible for eventual success or failure.

Q-learning [2] constitutes a seminal model-free RL algorithm that obviates the need for explicit knowledge of the environment's transition dynamics $\mathcal{P}$. The algorithm centers on learning the action-value function $Q^\pi(s, a)$, which estimates the expected return when executing action a in state s and subsequently adhering to policy π. The optimal action-value function $Q^*(s, a)$ satisfies the Bellman optimality equation, which expresses the recursive relationship between the value of a state-action pair and the values of successor states. Traditional Q-learning employs tabular representations where each state-action pair maintains a separate entry, but this approach becomes computationally intractable for high-dimensional or continuous state spaces such as those encountered in quantum circuit design.

Deep Q-Networks (DQN) [3] address this scalability limitation by leveraging deep neural networks as function approximators, parameterizing the Q-function as $Q(s, a; \theta)$ where θ represents the network's trainable weights. The neural network learns to approximate the optimal action-value function $Q^*(s, a)$ through experience, enabling generalization across similar states and drastically reducing the memory requirements compared to tabular methods. The DQN architecture incorporates two critical innovations that collectively stabilize the learning process and enable convergence in complex domains. First, experience replay maintains a replay buffer $\mathcal{D}$ that stores past experiences as tuples (s, a, r, s'). During training, mini-batches are sampled uniformly at random from $\mathcal{D}$, thereby breaking the temporal correlations inherent in sequential data and improving sample efficiency by allowing the agent to learn from each experience multiple times. This mechanism is analogous to how biological systems consolidate memories through rehearsal. Second, the algorithm employs a separate target network $Q(s, a; \theta^-)$ with parameters θ^- that are updated periodically by copying weights from the primary network. This stabilization technique prevents the oscillations and

divergence that can occur when the target values used in gradient descent are themselves constantly changing, providing a stable reference point for computing temporal difference errors.

The network parameters θ are optimized by minimizing the mean squared temporal difference (TD) error:

$$\mathcal{L}(\theta) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta)\right)^2\right] \tag{2}$$

This loss function quantifies the discrepancy between the network's current Q-value estimate and the bootstrapped target derived from the Bellman equation, effectively teaching the network to predict long-term returns from immediate observations.

A well documented limitation of standard DQN is its tendency to overestimate action values, particularly in the early stages of training when Q-value estimates exhibit high variance [4]. This overestimation stems from the max operator in the Bellman update, which systematically selects the highest noisy estimate rather than the true maximum expected value. Double DQN [5] mitigates this bias through a decomposition strategy that decouples action selection from action evaluation. Specifically, the online network (with parameters θ) identifies the best action, while the target network (with parameters θ^-) evaluates that action's value:

$$Q_{\text{target}} = r + \gamma Q(s', \arg\max_{a'} Q(s', a'; \theta); \theta^-) \tag{3}$$

By formulating the networks this way, we avoid the systematic overestimation that would occur due to the propagation of evaluation errors to selection errors, which would cause instability and less precise value estimates.

This adaptability and generality of the DQN model, as well as its variants, have helped the model achieve significant successes in solving problems in diverse domains where automated design is possible. Some significant successes achieved using the model include solving combinatorial optimization problems, where the model is used to solve the traveling salesman problem or the graph partitioning problem, where the model is used to generate good-quality solutions to the problem through the selection of the edges in the graphs [6]. Another domain where the model is used is in the design of molecules, where the model is used in drug discovery to generate molecules with desirable pharmacological properties [7]. Another domain where the model is used is in the compilation of quantum circuits, where the model is used to optimize the quantum circuit gates in order to reduce the depth of the circuit as well as the error rates in the quantum circuit [8]. In this paper, we propose the application of the reinforcement learning model in the domain where the model is used to generate good-quality quantum circuits used in the optimization process in the quantum kernel optimization domain. The model is used to generate high-quality quantum feature maps, thus representing a significant move from the traditional model used in this domain.

2.2 Quantum computing and quantum machine learning Fundamentals

Quantum computing represents a transformative paradigm in computation, leveraging the principles of quantum mechanics to address complex problems that surpass the capabilities of classical systems [9]. Central to quantum computing is the qubit, which differs from classical bits by existing in a superposition of the states 0 and 1, granting exceptional computational flexibility. Quantum computers excel in solving intricate tasks by harnessing phenomena like superposition, entanglement, and interference [10]. Both theoretical and empirical studies suggest their superiority in fields such as machine learning, optimization, and simulations, outperforming classical algorithms in efficiency [11, 12].

Quantum Machine Learning (QML) combines quantum information processing with machine learning techniques, addressing tasks such as classification, regression, and clustering [13–16]. This emerging field aims to solve problems that pose significant challenges for classical computers. Quantum-enhanced machine learning algorithms [17] hold promise for improving supervised, unsupervised, and reinforcement learning tasks [18–20], offering potential quadratic or exponential speedups in certain applications. Thanks to their capacity to manage high-dimensional data and detect intricate patterns, QML methods often surpass classical machine learning (CML) algorithms in specific domains [21], including applications like Grover's Algorithm [20]. This research investigates the performance of the Quantum Support Vector Classifier (QSVC) compared to its classical version using a realistic dataset containing bias.

The Quantum Support Vector Classifier (QSVC) classification algorithm is rooted in Support Vector Machines (SVM), adapted to leverage quantum computing principles. While classical SVMs employ pre-defined kernel functions, QSVCs derive kernels from quantum circuits [22]. In the QSVC model, input data is encoded into quantum states, which are then mapped into a high-dimensional quantum feature space. A crucial aspect of QML's development is the choice of quantum feature maps, including the Z-feature map, ZZ-feature map, and Pauli-feature map [23]. Among these, the ZZ-feature map has been identified as the most effective by Simões et al. [21], making it the preferred option in this study.

To facilitate the implementation of QML algorithms, the Qiskit Machine Learning library provides a comprehensive suite of tools [14, 24]. These include Quantum Kernels, Quantum Neural Networks (QNNs), and various learning algorithms, offering a versatile platform for exploring the potential of quantum-enhanced machine learning.

The use of Shap Values for XAI Among the most important ideas in machine learning model interpretation is the application of Shapley values, which provide a means of quantifying the contribution of each feature to the decision made by the model. Developed initially in cooperative game theory, Shapley values have been extended to interpret predictions from machine learning models such as support vector machines (SVMs). Recently, their application has been further extended to quantum circuits in an effort to improve the interpretability of quantum machine learning

(QML) models [25]. The application of Shapley values within the context of the Quantum Support Vector Classifier (QSVC) may yield interesting information regarding the decision-making of the model and increase the transparency of its predictions.

3 Related Works

In the past, numerous efforts have been directed towards utilizing various fields of artificial intelligence, in particular Support Vector Classification (SVC), which is a supervised learning algorithm used for classification tasks. It finds the optimal hyperplane that maximally separates different classes in a dataset. If the data is not linearly separable, SVC can use kernel functions to map the data into a higher-dimensional space where separation is possible. It aims to maximize the margin between classes while minimizing classification errors, in particular to improve the efficiency of quantum systems[26]. It was demonstrated that it is possible to use an SVC with a precomputed kernel to resolve a classification problem. In particular, a QSVC is used to compare the result of a classification problem between a trapped-ion quantum computer and a noiseless simulated quantum computer. The work shows that the QSVC is capable of increasing the accuracy on different datasets, both on the simulated quantum computer and on the real trapped-ion quantum computer [27].
Other works use also a deep reinforcement learning to optimize the parameter of a quantum circuit such as number of qubit or depth of the circuit, in fact is possible to train an RL agent to optimize an arbitrary circuit on a 12 qubits circuit, the agent is capable to reduce the depth of the circuit by a 25% and the number of gate in the circuit by a 15% [8]
It is also possible to optimize not only the depth and number of qubits for a quantum circuit but also the angle seen as a parameter for some rotational gates. In this case, we are talking about Variational Quantum Eigensolvers (VQEs), which are promising for utilizing near-term quantum devices in practical applications. The RL algorithm autonomously navigates the space of possible variational ansätze to identify circuits that maintain accuracy in estimating ground-state energies while minimizing circuit depth. The approach uses an intrinsically motivated design to improve accuracy iteratively without significantly increasing complexity. Demonstrated on the lithium hydride (LiH) molecule, a benchmark problem, the algorithm achieves chemical accuracy and reduces circuit depth, marking a state-of-the-art performance in VQE optimization[28].

4 Methodology

One possible application of quantum circuits is the creation of quantum kernels for data representation, leveraging their unique properties. However, designing a quantum circuit for a specific task requires both time and expertise. In this study, we propose the use of a reinforcement learning (RL) agent to dynamically construct a quantum circuit from which a kernel is evaluated to solve a classification problem using a Quantum Support Vector Classifier (QSVC).
The following Figure 1 represents the training phase in which a Reinforcement Learning (RL) algorithm optimizes a Quantum Support Vector Classifier (QSVC).

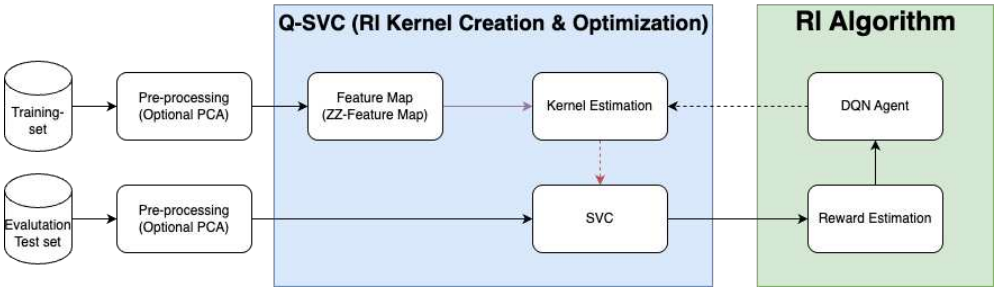


Fig. 1 The diagram illustrates the interaction, during the training phase, between the Q-SVC module (blue) responsible for quantum kernel creation and evaluation, and the RL Algorithm module (green) that optimizes the kernel structure.

The process begins with a preprocessing step, where the input data undergoes optional Principal Component Analysis (PCA) to reduce its dimensionality and improve computational efficiency. The preprocessed data is subsequently passed to the Feature Map module, where classical data is encoded into quantum states through a parameterized ZZ-Feature Map. This step constitutes the core of the quantum kernel method, as the feature map determines the transformation of the data into a higher-dimensional Hilbert space. Once the data is encoded, the quantum circuit is executed to estimate the kernel matrix, which quantifies the fidelity between pairs of quantum states. The resulting kernel matrix serves as the input for the Support Vector Classifier (SVC), which performs the final classification task based on the pre-computed kernel values. Simultaneously, a Deep Q-Network (DQN) Agent interacts with the environment by selecting circuit gates and their corresponding parameters, influencing the structure of the feature map. The quality of the kernel and the classifier's accuracy are assessed by the Reward Estimation module, which provides a feedback signal to the agent.

The Training phase is followed by the Evaluation phase 2, where the best kernel found by the agent is used to perform the classification task. In this phase, the agent is no longer involved, as its role in exploring and optimizing the kernel has already been completed. The model simply applies the selected kernel to classify new, unseen data.

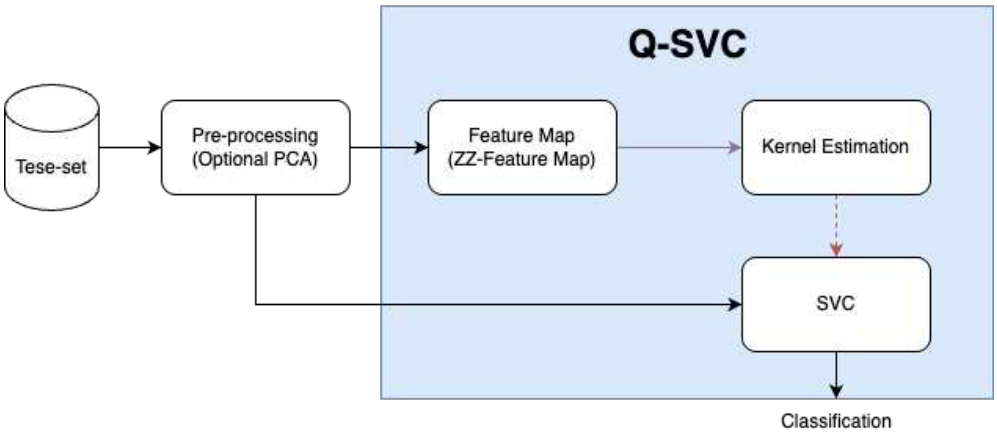


Fig. 2 Diagram of the Evaluation Phase, in which the test set is used to evaluate the final quantum circuit created.

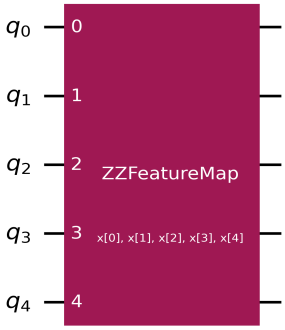


Fig. 3 Classical ZZFeatureMap presented in the qiskit library.

Through this iterative process, the RL agent starts from a simple feature map like Figure 3 and learns to optimize the circuit, improving the kernel's discriminative power and the classifier's overall performance. The steps done by the agent are illustrated in the following Figure 4:

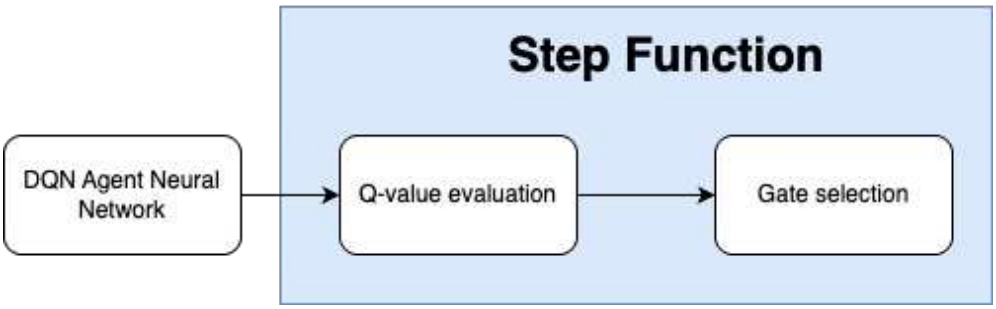


Fig. 4 Diagram of a single step done by the RL-agent

As follows is also reported the final circuit created by the agent and used in the evaluation phase (Figure 5):

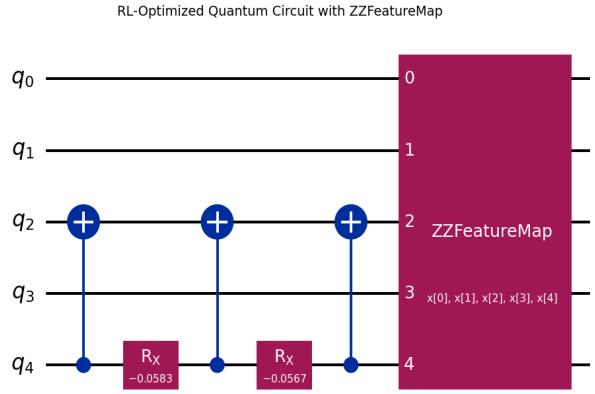


Fig. 5 Final circuit created by the agent

4.1 Quantum kernel estimation

We used in our analysis the ZZFeatureMap based on Qiskit, this map employs a sequence of controlled-Z (CZ) gates to enable the encoding of classical data into a quantum state. Specifically, the ZZFeatureMap takes in a list of classical features and applies a sequence of single-qubit rotations and two-qubit entangling gates, which has the effect of projecting the classical information onto a high-dimensional quantum Hilbert space. This encoding has particular value in high-dimensional classification tasks and allows the quantum system to take advantage of the expressiveness of the

quantum state space. Due to its effectiveness with respect to circuit depth and because it can represent intricate relationships among features, it was chosen as the feature map for the quantum support vector classifier (QSVC) in this study.

In this work, the **Quantum Circuit** starting from the feature map is integrated with a series of gates chosen by the agent from the action space through the neural network. In this way, the circuit is better equipped to recognize the dataset's features for a classification problem, respect to the classical approach of qiskit where the classical QSVC implemented in the qiskit_machine_learning library use only the feature map. The agent start with the feature map that represents the econded data as a quantum state and does not vary, while the part of the circuit created by the agent is continuously modified and reset to an empty **QuantumCircuit** at the end of each episode using the *reset()* function.
In kernel theory, given a set of training inputs x from the input space $X \subseteq \mathbb{R}^n$, pattern recognition relies on a similarity measure called the kernel function k. Formally, a kernel is defined as a mapping:

$$k : X \times X \to \mathbb{R}$$

Let $\mathcal{H}$ represent a Hilbert space, often referred to as the feature space, and let x be a sample point from X. A feature map Φ is introduced, which projects input points into this high-dimensional space:

$$\Phi : X \to \mathcal{H}$$

The similarity between two inputs, after embedding in the feature space, is determined by their inner product, which also defines the kernel function:

$$k(x_i, x_j) = \langle \Phi(x_i), \Phi(x_j) \rangle$$

For a dataset of N input points $\{x_1, x_2, \ldots, x_N\} \in X$, the kernel matrix $K \in \mathbb{R}^{N \times N}$, also known as the Gram matrix, contains all pairwise kernel evaluations:

$$K = [k(x_i, x_j)]_{ij}$$

The advantage of kernel methods lies in their ability to implicitly operate in high-dimensional or even infinite-dimensional spaces through the kernel function, without explicitly computing the feature map Φ. A common example is the Radial Basis Function (RBF) kernel, which evaluates the kernel matrix as:

$$K(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right)$$

In the quantum domain, Quantum Kernels (QK) utilize quantum states to compute the kernel matrix. Inputs $x \in \mathbb{R}^n$ are encoded into quantum states via a feature map implemented as a unitary transformation $U_\phi(x)$, acting on an initial n-qubit ground state:

$$\phi : x \mapsto |\phi(x)\rangle = U_\phi(x)|0\rangle^{\otimes n}$$

The quantum kernel matrix is then defined by the pairwise inner product of quantum states:

$$\begin{aligned} K(x_i, x_j) &= |\langle\phi(x_i)|\phi(x_j)\rangle|^2 \\ &= |\langle 0|^{\otimes n} U_\phi^\dagger(x_i) U_\phi(x_j)|0\rangle^{\otimes n}|^2 \end{aligned}$$

The entries of the quantum kernel matrix are influenced by the unitary transformation U_ϕ, which determines the embedding of the input data into the quantum Hilbert space. This adaptability allows Quantum Kernels to capture intricate relationships in the data, potentially offering advantages over classical kernels[29].

4.2 Simulation

The parts mentioned above are combined into a single circuit that is transpiled using the built-in transpiler of Qiskit. After that is possible to create a kernel from it using a qiskit function called *FidelityStatevectorKernel()* or *FidelityQuantumKernel()*, which the first allows to generate a quantum kernel using the **StateVector** so without considering the noise, while the second allows to add the noise map from a real backend. The reasons for the use of this kind of kernels are the following :

- **Quantum Fidelity**: The fidelity between two quantum states represented by state vectors ψ and ϕ is defined as $F(\psi, \phi) = |\langle\psi|\phi\rangle|^2$. This quantifies how "close" or "similar" two states are. Similar to classical machine learning, this would be a form of a kernel function, like the Gaussian Kernel. The fidelity can be calculated by the inner product of the state vectors; thus, explicit comparisons between all pairs of data points will no longer be necessary.
- **The indroduction of real noise** : It is possible with *FidelityQuantumKernel()* function to use a fidelity created by the *computeUnCompute* function that take in argument a sampler to which is possible to pass the Sampler with the noised simulator backend, but as said before the complexity of this function is high in fact it is $O(N^2)$.
- **Reducing Complexity via Parallelization and Similarity Simplification**: In contrast, the FidelityStatevectorKernel can use the inherent parallelization available in quantum hardware and properties of fidelity to compute the similarities of state vectors in parallel. For properly prepared and stored quantum states, fidelity can

be computed in $O(N)$ using only one evaluation per state against some common reference or leveraging the structure of the quantum system itself.

This kind of approach reduces computational time, especially for large datasets, another way to do this is to use a *FidelityQuantumKernel()* but in this case the computational complexity raise to $O(N^2)$. It is important to consider that if the *FidelityQuantumKernel()* is chosen is also possible to calculate the kernel on a real backend or a simulated one through *AerSimulator()* using the *SamplerV2()*, is necessary to consider that if the circuit is too complex the virtual representation of this one is twice bigger than the original.
After the evaluation part, the **SVC** is used; in this case, a patch is used to accelerate the **SVC**, in particular, the patch created by Intel [30]. In particular, the **SVC** uses a *"precomputed"* kernel; in this way, it was possible to evaluate the kernel using the multi-thread approach. It is important to mention that the **qiskit-machine-learning** library exists the **QSVC** implemented like a normal **SVC** and just needs the circuit as input, but it uses as default the *FidelityQuantumKernel()* and due to the amount of data is not used in this paper.

4.3 RL Agent

The agent is created using a Deep Q-Network (DQN)[31] neural network composed of two neural networks, the *policy_net* and *target_net*. In particular, the *policy_net*, which represents the main network actively trained, is responsible for making decisions based on the selected state by following the state-action pair. Specifically, in our case, it calculates the Q-values for all the actions present in the action space and returns the one with the highest value, i.e., the next gate that needs to be used. Its update occurs by calculating the loss between the values predicted by the *policy_net* and those predicted by the *target_net*.
The *target_net* is used to compute the target Q-values in a stable manner, without being updated continuously. When the agent observes the next state after executing an action, the *target_net* evaluates the best Q-value for the next state, which will be used to calculate the loss.
In detail, the states used to compute the Q-values are saved and inserted into the BATCH to compute the Q-values represent a one-hot encoding for the used gate for that step, using a fixed dictionary and also a one-hot encoding for represent which qubits are manipulated by the inserted gate, a selection of three angles predicted by the *policy _net* if the last gate inserted is a rotational gate. As said before, the *target_net* is not updated at each step; in fact, it can be updated in two ways.

- **Soft update**: Every time the agent performs an optimization step, there is a small update of the *target_net* weights, which occurs through an interpolation of the weights between the *policy_net* and the *target_net*.
- **Hard update**: Every *target_update* episodes, the *target_net* is completely synchronized with the *policy_net* by copying the weights from the *policy_net* into the *target_net*.

The target value $Q_{\text{target}}(s')$ is computed using a Double DQN approach:

$$Q_{\text{target}}(s') = \begin{cases} r + \gamma \cdot Q_{\text{target-net}}\left(s', \arg\max_{a'} Q_{\text{policy-net}}(s', a')\right) \\ 0 \end{cases}$$

For non-terminal states, the target is defined as the sum of the immediate reward r and the discounted future Q-value using γ, where the action selection is performed by the policy network ($\arg\max_{a'} Q_{\text{policy-net}}(s', a')$), while the value estimation is delegated to the target network ($Q_{\text{target-net}}$). For terminal states, the target is set to zero. This strategy mitigates the overestimation bias inherent in traditional Q-learning.

The current Q-value $Q_{\text{current}}(s, a)$ combines two components: the discrete action-value $Q_{\text{policy-net}}(s, a)$ for action selection, and a continuous parameter $\theta_{\text{policy-net}}(s, a)$ predicted by the network for rotation angles. The integration occurs through direct summation, enabling simultaneous learning of policies for discrete actions (e.g., quantum gate selection) and continuous parameters (e.g., rotation angles). The $\theta_{\text{policy-net}}$ component is generated directly by the neural network's output layers.

$$\begin{aligned} Q_{\text{current}}(s, a) &= Q_{\text{policy-net}}(s, a) + \theta_{\text{policy-net}}(s, a) \\ \theta_{\text{policy-net}}(s, a) &\sim \text{Output neural network} \end{aligned}$$

The total loss function used to update the network combines two distinct terms. The action loss ($\mathcal{L}_{\text{action}}$) employs the SmoothL1 Loss, a hybrid of L1 and L2 loss that reduces sensitivity to outliers, applied to the difference between current Q-values and target values. The rotation loss ($\mathcal{L}_{\text{rotation}}$) uses the Mean Squared Error (MSE) between predicted rotation parameters and those stored in the replay buffer. The total loss ($\mathcal{L}_{\text{total}}$) results from the linear combination of these components, automatically balancing the dual learning objectives.

$$\begin{aligned} \mathcal{L}_{\text{action}} &= \text{SmoothL1}\left(Q_{\text{current}}(s, a), Q_{\text{target}}(s')\right) \\ \mathcal{L}_{\text{rotation}} &= \text{MSE}\left(\theta_{\text{policy-net}}(s, a), \theta_{\text{stored}}\right) \\ \mathcal{L}_{\text{total}} &= \mathcal{L}_{\text{action}} + \mathcal{L}_{\text{rotation}} \end{aligned}$$

The system employs a hybrid strategy for updating the target network. The soft update applies an exponential moving average ($\theta_{\text{target}} \leftarrow \tau\theta_{\text{policy}} + (1 - \tau)\theta_{\text{target}}$) at each step, ensuring stability through incremental adjustments. Periodically (every N steps), a hard update is executed, directly copying the policy network's weights to the target network to enforce full synchronization. The parameter τ (typically 0.005) controls the soft update rate.

$$\begin{aligned} &\text{Soft update: } \theta_{\text{target}} \leftarrow \tau\theta_{\text{policy}} + (1 - \tau)\theta_{\text{target}} \\ &\text{Hard update: } \theta_{\text{target}} \leftarrow \theta_{\text{policy}} \quad \text{each } N \text{ step} \end{aligned}$$

In the following part, we outline the set of hyperparameters used to tune the agent's learning process. Every hyperparameter has a significant role in defining the agent's behavior and its capacity to search the action space efficiently.

Several hyperparameters of the reinforcement learning agent were adjusted from their default values to better suit the task of quantum circuit construction. In particular, the *epsilon threshold* was initially set to a higher value to encourage broader exploration of the large action space and was gradually decreased during training. The *replay memory* size and *batch* size were increased to allow the agent to learn from a more diverse and representative set of past experiences. The *learning rate* and *discount factor* were slightly tuned to balance convergence speed and stability. Additionally, the *target network update frequency* was modified to reduce oscillations in Q-value estimates. These adjustments were made based on preliminary experiments aimed at enhancing the agent's exploration capability, which was essential given the high dimensionality and combinatorial nature of the quantum gate action space.

4.3.1 Action Space

The action space available to the agent is created before the training phase and includes all the possible gates that the agent can apply to the circuit in the step phase. The combinations are created from a finite set of gates. When Q-values are computed using the *policy_net* and the *target_net*, the values returned by these are the identifying numbers of the actions; the action space during the training phase does not change, otherwise it will be impossible for the agent to perform a correct training. The agent's goal is to explore the action space to maximize the reward, which is done using an **epsilon-greedy** policy. The decreasing ϵ allows exploration of new actions, not immediately maximizing the reward but permitting the agent to explore new possibilities before focusing on actions that progressively yield better results. The math expression for the ϵ that is calculated in a dynamic way is the following:

$$\text{eps_threshold} = \epsilon_{\text{end}} + (\epsilon_{\text{start}} - \epsilon_{\text{end}}) \cdot e^{-\frac{\text{steps_done}}{\epsilon_{\text{decay}}}}$$

Where ϵ_{end} represents the final value of epsilon, ϵ_{start} is the initial value of epsilon, `steps_done` indicates the number of steps taken so far, and ϵ_{decay} denotes the epsilon decay constant.
After the agent has selected an action from the action space the gate associated to the action number is applied to the circuit and the circuit is evaluated using the procedure explored in Circuit Simulation section, after that the episode is considered finished if the accuracy has reached a precise value otherwise the episode is considered not done and the agent will provide to insert a new gate until the limit of step for the episode or the accuracy target is reached.
The function is defined as follows *observation, reward, terminated, measures = env.step(actions)* where the *actions* is a combination of the action selected and selected angle. The values that the function returns are:

- Observation: The difference between the accuracy calculated for the circuit and the goal accuracy.
- Reward: The reward calculated for that circuit.
- Terminated: Is a Boolean function that indicates if the episode is finished
- Measures: A function that contains different information about the circuit.

4.3.2 Reward

The objective of the Reinforcement Learning (RL) agent is to maximize the expected cumulative reward to learn which gates are the best for the classification problem. The reward is defined as:

The objective function $J(\theta)$ is defined as the expectation of the sum of discounted rewards over time:

$$J(\theta) = \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t R_t\right]$$

where γ is the discount factor, which belongs to the interval $[0, 1]$, and R_t is the reward at time t, determined by the accuracy of the Quantum Support Vector Classifier (QSVC). This accuracy is computed by a function f_{accuracy}, which depends on the quantum circuit C_t at time t and the dataset $\mathcal{D}$ used for training the QSVC:

$$R_t = f_{\text{accuracy}}(C_t, \mathcal{D})$$

Previous approaches using RL agents for the dynamic design of quantum circuits focused solely on optimizing the rotational angles, keeping the feature map as the only component of the quantum circuit. In this work, we aim to extend this approach by not only utilizing the feature map but also allowing the agent to select additional gates, with their corresponding angles also determined by the agent itself.
It was chosen to use accuracy as the reward since the agent's objective is to train itself to construct a circuit that improves the performance of a QSVC, with the performance of this mechanism being measured in terms of accuracy.

4.3.3 Action Space and Gate Set Definition

The reinforcement learning agent works by choosing quantum gates from an action space we've designed to balance expressiveness with practical computation. We build this space from the Clifford+T universal gate set—a well-established foundation—and add continuous parametric rotation gates to give the circuit more representational power. Why this particular combination? Clifford gates combined with T gates give us efficient universal quantum computation, while parametric rotations let us fine-tune quantum state transformations, which is crucial for encoding features properly.

Our agent can choose from a gate repertoire $\mathcal{G}$ that falls into two main categories. For single-qubit operations, there are the fundamental Clifford gates: the Identity gate (I), Hadamard gate (H), and the three Pauli gates (X, Y, Z). Each transforms individual qubit states in its own way. Beyond these, we include three families of continuous parametric rotations: $R_X(\theta)$, $R_Y(\theta)$, and $R_Z(\theta)$, where the rotation angle θ

can be anything in the range $[0, 2\pi]$. These rotations let the agent explore a continuous space of quantum transformations, adapting to whatever feature structures the dataset throws at it. For two-qubit operations, we have the controlled-NOT gate (CNOT or CX), the controlled-Z gate (CZ), and the SWAP gate. Each of these creates quantum correlations between qubit pairs through entanglement or controlled operations.

Mathematically, we formalize the action space to capture both which gate to use (a discrete choice) and what parameters to set (continuous values). We represent each action as a tuple (g, q_1, q_2, θ) where $g \in \mathcal{G}$ tells us the gate type, the indices $q_1, q_2 \in \{0, \ldots, n-1\}$ specify which qubit(s) the gate acts on (we use $n = 5$ qubits here, matching the five principal components we keep after PCA), and $\theta = (\theta_X, \theta_Y, \theta_Z) \in \mathbb{R}^3$ gives the rotation angles for the three parametric gates. This gives us the action space:

$$\mathcal{A} = \{(g, q_1, q_2, \theta) \mid g \in \mathcal{G}, q_1, q_2 \in \{0, \ldots, n-1\}, \theta \in \mathbb{R}^3\} \tag{4}$$

Single-qubit gates only need the q_1 index (leaving q_2 unused), while two-qubit gates require both indices to specify control and target qubits or which pair to swap.

To make this work efficiently with our neural network, we use a structured encoding scheme. Each gate type gets a one-hot encoding—basically, we convert the categorical gate choice into a binary vector $\mathbf{e}_i \in \{0, 1\}^{|\mathcal{G}|}$ where only the i-th component equals 1 and everything else is 0. The index i follows a predefined mapping shown in Table 1.

Gate	Index
CNOT	0
Controlled-Z	1
SWAP	2
Identity	3
Hadamard	4
Pauli-X	5
Pauli-Y	6
Pauli-Z	7
R_X	8
R_Y	9
R_Z	10

Table 1 Mapping of gates to indices for one-hot encoding.

This mapping keeps things consistent across all training episodes. We also encode qubit selection with a binary vector $\mathbf{q} \in \{0, 1\}^n$ that shows which qubits are involved in the current operation. This gives the neural network explicit information about the circuit topology being built, which helps it learn effective operation sequences.

We use a fully connected topology for the circuit architecture, meaning any qubit can directly interact with any other through two-qubit gates—no topological restrictions. This choice prioritizes flexibility in exploring different circuit configurations over hardware efficiency. We're betting that quantum devices will continue their rapid progress toward all-to-all connectivity, making this a reasonable assumption. Yes, full

connectivity makes the action space substantially larger than restricted topologies like nearest-neighbor interactions would, but it lets the agent discover optimal entanglement patterns without geometric constraints getting in the way. If we count up the action space cardinality, we get approximately this: for each of 11 gate types, the agent picks from $n^2 = 25$ possible qubit configurations (covering both single-qubit placements and two-qubit pairs), giving us roughly $|\mathcal{A}| = 11 \times 25 = 275$ discrete gate-qubit combinations. This combinatorial complexity is exactly why we need the deep neural network approximation in our DQN architecture.

For the parametric rotation gates, we don't discretize the continuous angle parameters $\theta = (\theta_X, \theta_Y, \theta_Z)$—instead, the neural network predicts them directly through a dedicated output layer. The DQN architecture has a specialized fully connected layer (`fc_rotation`) with three output neurons that generate real-valued predictions for the rotation angles. The agent learns these continuous parameters simultaneously with the discrete gate selection policy, which means it can fine-tune both the structural composition of the quantum circuit and the precise parametrization of rotation gates at the same time. This hybrid discrete-continuous action space is actually one of the key innovations in our approach: it combines the efficiency of discrete gate selection with the expressiveness of continuous parameter optimization, all within a unified reinforcement learning framework.

4.3.4 Hyperparameter Configuration

Table 4.3.4 summarizes all hyperparameters used in the RL training process. These values were selected based on preliminary experiments aimed at balancing exploration efficiency with computational feasibility.

Hyperparameter	**Symbol**	**Value**
Learning Rate	α	1×10^{-5}
Discount Factor	γ	0.99
Batch Size	B	32
Replay Memory Capacity	M	10^7
Initial Epsilon	ϵ_{start}	1.0
Final Epsilon	ϵ_{end}	0.01
Epsilon Decay Steps	ϵ_{decay}	8600
Target Network Update (Soft)	τ	0.005
Target Network Update (Hard)	N_{target}	Every 5 episodes
Number of Episodes	-	50
Maximum Steps per Episode	-	20
Neural Network Hidden Layers	-	$[128, 128]$
Optimizer	-	AdamW (amsgrad=True)
Loss Function (Action)	-	SmoothL1Loss
Loss Function (Rotation)	-	MSE

Table 2 Reinforcement Learning Hyperparameters

- *Learning Rate ($\alpha = 10^{-5}$)*: A low learning rate was chosen to ensure stable convergence in the high-dimensional quantum circuit space, preventing destructive updates that could destabilize learned policies.
- *Epsilon Decay*: The exploration schedule was designed to allow extensive initial exploration (up to 8,600 steps) given the combinatorial nature of the quantum gate action space, gradually transitioning to exploitation as the agent identifies promising circuit configurations.
- *Replay Memory*: A large memory capacity (10^7 transitions) was employed to store diverse experiences across multiple episodes, enabling the agent to learn from a rich distribution of circuit-performance pairs.
- *Batch Size*: Set to 32 to balance gradient estimate variance with computational efficiency during mini-batch updates.
- *Target Network Updates*: Dual update strategy combining soft updates ($\tau = 0.005$ at each step) for stability and periodic hard updates (every 5 episodes) for synchronization.

5 Experimental Results

5.1 Dataset Description and Preprocessing

The experiments were conducted using the *Student Performance Data* dataset, publicly available on Kaggle[1]. This dataset contains 2,392 samples of U.S. student academic records with 15 features capturing demographic, academic, and behavioral characteristics as follow some example:

- *Demographic*: Age (15-18 years), Gender (Male/Female), Ethnicity (Caucasian, African American, Asian, Other)
- *Academic Context*: ParentalEducation (None, High School, Some College, Bachelor's, Higher), StudyTimeWeekly (0-20 hours), Absences (0-30 days), GPA (2.0-4.0)
- *Support Systems*: Tutoring (Yes/No), ParentalSupport (None, Low, Moderate, High, Very High)
- *Extracurricular*: Extracurricular activities, Sports, Music, Volunteering (all binary)
- *Target Variable*: GradeClass (0-4, representing grade categories from F to A)

To have a complete overview of the list of the features is possible to visit the link above. Some preprocessing operations were done to the dataset and consisted of the following steps:

- **Feature Engineering**: The target variable was transformed using $y' = 4 - y$ to reverse the grade ordering for computational convenience.
- **Feature Selection**: Based on correlation analysis with the target variable, three features with negligible correlation ($r < 0.05$) were removed: Age, Sports, and Music. This resulted in a dataset with 12 features.

[1] https://www.kaggle.com/datasets/rabieelkharoua/students-performance-dataset

- **Normalization**: All features were scaled to the range $[0, 1]$ using MinMaxScaler from scikit-learn:
$$X_{\text{scaled}} = \frac{X - X_{\min}}{X_{\max} - X_{\min}} \tag{5}$$
- **Dimensionality Reduction**: Principal Component Analysis (PCA) was applied to reduce the feature space from 12 to 5 dimensions, preserving approximately 65% of the total variance. This reduction was necessary to:
  - Minimize quantum circuit complexity (5 qubits vs. 12)
  - Reduce computational overhead in kernel evaluation
  - Mitigate noise effects in quantum simulation
- **Train-Test Split**: The dataset was partitioned using stratified sampling with `random_state=42` and `random_state=123` to ensure reproducible splits and maintain class distribution:
  - Training set: 70% (1,674 samples)
  - Validation set: 18% (430 samples)
  - Test set: 12% (288 samples)

The complete preprocessing code and dataset splits are available in the supplementary material repository. The dataset was chosen because it is expected to yield an accuracy below 90 percent, even with the best classical SVC model. Additionally, this dataset likely contains bias within certain features. It is possible to demonstrate that the RL-QSVC can reduce the bias. Prior to data preprocessing, a statistical overview of the dataset was created in order to find out which features are correlated with each other. It was noticed that some features had no correlation with the target variable, or at least are not relevant from a statistical point of view. For that reason, the following features were dropped: Age, Sports, Music.

Due to the computational problem of dealing with all the 12 features, a PCA is conducted to reduce the dimension of the dataset. A function is created to plot and visualize the cumulative explained variance of the principal components obtained from PCA. It calculates the cumulative sum of the explained variance ratios, which indicates how much variance in the original dataset is captured by the principal components as more components are included.

The number of features is reduced to 5 to make the circuit less complex and also to reduce the amount of noise presented into the circuit. With 5 features, it is possible to preserve approximately 65% of the variance, but thanks to the quantum kernel, it is possible to use fewer features to express more information.

In this section, the results obtained from the experiment are shown. In particular, they are compared to the result of a classical SVC with the addition of a SHAP values analysis to show that the QSVC is also capable of reducing the bias.

5.2 Comparison of Classification Models

To evaluate the performance of the proposed reinforcement learning-based Quantum Support Vector Classifier (RL-QSVC), we conducted an extended comparison against multiple baseline models. We first performed a comprehensive analysis of classical

Support Vector Classifiers (SVC) with different kernel functions, and then compared the best-performing classical model against the QSVC implementation provided by the Qiskit Machine Learning library and our RL-QSVC approach.

5.2.1 Classical SVM Baseline Comparison

We evaluated three classical SVM configurations with different kernels: linear, Radial Basis Function (RBF), and polynomial. Each model was optimized via grid search with 5-fold cross-validation over the following parameter spaces:

Table 3 Hyperparameter search grid for SVM models.

Kernel	Parametro	Valori testati
Linear SVM	**C**	$0.1, 1, 1.2, 1.5, 2, 2.5, 5, 30$
	tol	$10^{-3}, 10^{-2}, 10^{-4}, 10^{-5}$
RBF SVM	**C**	$0.1, 1, 1.5, 2, 5, 10, 30$
	gamma	`scale`, `auto`, $0.001, 0.01, 0.1, 1$
	tol	$10^{-3}, 10^{-2}, 10^{-4}$
Polynomial SVM	**C**	$0.1, 1, 1.5, 2, 5, 10, 30$
	degree	$2, 3$
	gamma	`scale`, `auto`, $0.01, 0.1$
	coef0	$0.0, 0.1, 1.0$
	tol	$10^{-3}, 10^{-2}$

Table 5.2.1 presents the performance comparison of the three classical SVM variants after hyperparameter optimization on the test set.

Model	Accuracy (%)	F1-Score (%)	Precision (%)	Recall (%)	Time (s)
Linear SVM	73.61	71.62	69.79	73.61	0.05
RBF SVM	76.39	74.80	75.27	76.39	0.06
Polynomial SVM	75.00	73.40	73.85	75.00	0.08

Table 4 Performance Comparison of Classical SVM Models with Different Kernels (Test Set)

The optimal hyperparameters selected for each model were:

- **Linear SVM**: $C = 30$, `kernel` = `linear`, `tol` = 0.001
- **RBF SVM**: $C = 30$, `kernel` = `rbf`, `gamma` = `scale`, `tol` = 0.001

- **Polynomial SVM**: $C = 10$, `kernel` = `poly`, `degree` = 3, `gamma` = `scale`, `coef0` = 1.0, `tol` = 0.001

As shown in Table 5.2.1, the RBF kernel achieved the best performance among classical approaches, with 76.39% accuracy and a training time of 0.06 seconds. The polynomial kernel showed intermediate performance (75.00%), while the linear kernel achieved the lowest accuracy (73.61%). Notably, all three models exhibited comparable computational efficiency, with training times under 0.1 seconds.

5.2.2 Quantum vs. Classical Comparison

Building upon the classical baseline analysis, we compared the best-performing classical SVM (RBF kernel) against two quantum approaches: the standard QSVC from Qiskit Machine Learning and our proposed RL-QSVC. Table 5 reports the comprehensive comparison on the test set.

Model	Accuracy (%)	Training Time (s)	Performance Metrics
RBF SVM	76.39	0.06	F1-Score: 74.80% Precision: 75.27% Recall: 76.39%
QSVC	74.65	333.74	F1-Score: 73.87% Precision: 74.25% Recall: 75.35%
RL-QSVC	76.73	53592.50	F1-Score: 78.79% Precision: 79.46% Recall: 79.17%

Table 5 Comparison between the best classical SVM (RBF kernel), standard QSVC, and RL-optimized QSVC (Test Set).

The results reveal several important insights. First, the standard QSVC implementation underperformed compared to the classical RBF SVM (74.65% vs. 76.39% accuracy) while requiring significantly longer training time (333.74 seconds vs. 0.06 seconds). This suggests that without proper optimization, quantum kernels may not provide advantages over well-tuned classical approaches.

However, our RL-optimized QSVC demonstrated superior performance across all key metrics. The RL-QSVC achieved 76.73% accuracy, comparable to the RBF SVM's 76.39%, while significantly excelling in F1-Score (78.79% vs. 74.80%), Precision (79.46% vs. 75.27%), and Recall (79.17% vs. 76.39%). This indicates that the RL-optimized quantum approach provides better balanced classification performance and is particularly effective at minimizing both false positives and false negatives.

The trade-off for this improved performance is substantially increased training time (53,592.50 seconds, approximately 14.9 hours). This computational cost reflects the iterative nature of reinforcement learning optimization for quantum circuit parameters. While this training time is considerably longer than both classical and standard quantum approaches, it represents a one-time investment during model development. Once

trained, the RL-QSVC can be deployed for inference with performance comparable to other quantum circuits.

These results demonstrate that reinforcement learning can effectively optimize quantum kernels to achieve superior classification performance compared to both classical SVMs and non-optimized quantum approaches. The improved precision and recall metrics are particularly valuable in applications where balanced performance across all classes is critical, justifying the additional computational investment during training.

6 Discussion

6.1 XAI With SHAP

SHapley Additive exPlanations (SHAP) values originate from cooperative game theory and provide a method to fairly distribute the "payout" among the features based on their contributions to the prediction. For any given prediction, the Shapley value of a feature represents its average contribution across all possible combinations of features. SHAP values are additive, meaning that the sum of the SHAP values for all features equals the difference between the model's prediction and the average prediction. This ensures that explanations are both consistent and interpretable, offering local explanations for individual predictions while allowing for global insights by aggregating SHAP values across many predictions.
SHAP has numerous benefits when interpreting machine learning models. At the core advantage is that it is model-agnostic, meaning it can be used on a broad array of machine learning frameworks, making it a highly versatile tool. SHAP is also monotonic; that is, an increase in a feature's contribution to the model's prediction will be reflected as an increase in the SHAP value. Moreover, by virtue of using principles from game theory, SHAP ensures fairness in feature contribution assignment and thus provides a fairer explanation of model predictions. Still, SHAP has its disadvantages. A significant drawback is that it is computationally expensive, making the computation of SHAP values quite resource-intensive, particularly for models with large numbers of features. In addition, the theoretical foundations of Shapley values can be extremely challenging for people without a background in game theory, introducing an additional layer of complexity in their interpretation.

6.1.1 The result of SHAP

In this section, the result of SHAP applied to the dataset is analyzed. To observe how bias is reduced through the quantum optimization process, we selected the Linear SVC as our classical baseline for interpretability analysis. The Linear SVC was chosen because its simpler decision boundary provides a clearer baseline for understanding feature interactions before quantum kernel optimization. The plot is created using the *KernelExplainer()* function that permits explaining the output of any function; in this case, the function used is the one from the **Linear SVC**. The following plot (Figure 6) illustrates how the features interact with one another.

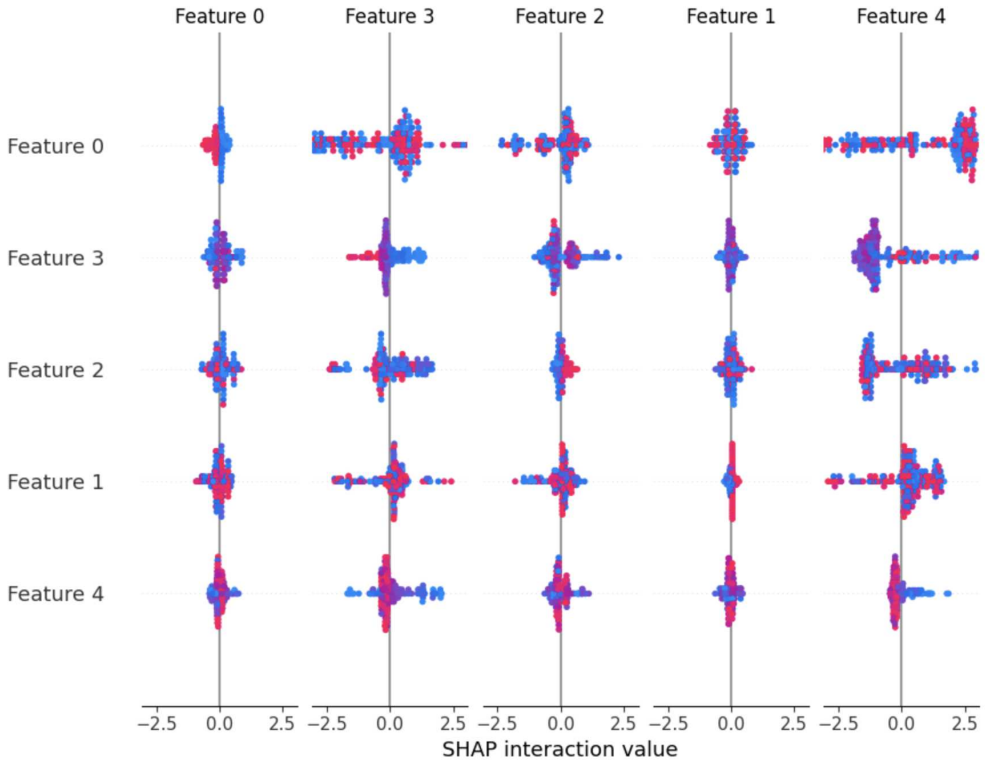


Fig. 6 Plot that shows the interaction in SHAP values between the features for the classical Linear SVC

In this figure, it is possible to see that some components, if the values of other components are high, hurt the model. In this particular case, we can see that component 4 hurts the prediction if the value of component 3 is high. In particular, it is possible to see how much a component is influenced by a feature of the dataset with some lines of code; here is the result. So we can observe that the component number 2 leads to a less precise classification if the component number 3 is high.
The SHAP values have also been computed for the RL-QSVC model, as shown in the following Figure 7:

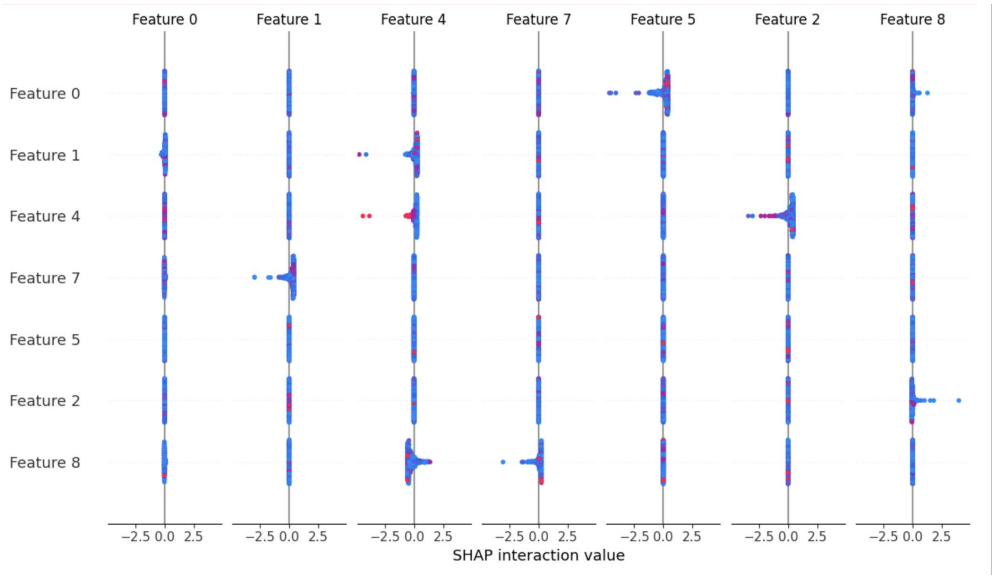


Fig. 7 Plot that shows the interaction in SHAP values between the features for the RL-QSVC

As can be observed by making a comparison, the SHAP values have significantly decreased, although some features still present high SHAP values, which is attributable to the dataset characteristics.

6.2 Statistical Analysis of SHAP Values

To further investigate the interpretability impact of using a reinforcement learning-based quantum kernel and to observe how bias is reduced through the optimization process, we performed independent two-sample t-tests on the SHAP values of each feature across all classes. The analysis compares the Linear SVC baseline against the RL-optimized QSVC model, with SHAP values extracted from the Linear SVC serving as the reference point for measuring bias reduction. The Linear SVC was selected as the comparison baseline because its linear decision boundary provides a clear reference for understanding how the quantum kernel optimization affects feature weighting and bias patterns.

The null hypothesis for each test assumed that the SHAP values for a given feature and class were drawn from the same distribution under both models. A p-value lower than 0.05 indicates a statistically significant difference, thereby suggesting that the RL-QSVC model processes or weighs that feature differently than the Linear SVC, potentially reducing inherent biases present in the classical linear approach.

The results show that only a subset of the features exhibited statistically significant differences in SHAP values when comparing the Linear SVC baseline to the RL-QSVC. For instance, in Class 0, only Feature 2 showed a p-value below 0.05. Similarly, Feature

Class	Feature	p-value	Significant (p ¡ 0.05)
0	Feature 2	0.0131	Yes
1	Feature 1	0.0473	Yes
2	Feature 3	0.0415	Yes
0	Feature 1	0.3864	No
2	Feature 0	0.6208	No

Table 6 Results of t-tests on SHAP values comparing the Linear SVC and RL-QSVC for selected feature-class combinations. Significant p-values indicate that the SHAP value distributions differ between the models, suggesting bias reduction through quantum optimization.

1 in Class 1 and Feature 3 in Class 2 also indicated significant differences. These findings suggest that the RL agent does not uniformly alter the importance of all features but instead induces selective changes that impact specific class-feature relationships, thereby reducing bias patterns observed in the linear baseline.

This selective modulation of feature influence implies that the RL-optimized quantum kernel has altered the model's internal representation in a nontrivial way compared to the Linear SVC. By examining the SHAP values from the Linear SVC as the baseline, we can observe how the quantum optimization process reduces bias and improves focus on discriminative features. The statistical differences confirm that certain features that exhibited problematic bias patterns in the linear model are now weighted differently, potentially contributing to the improved precision and recall metrics observed in the RL-QSVC. Overall, this analysis confirms that the RL-QSVC not only outperforms the classical linear approach in terms of overall performance metrics but also leads to a measurable shift in model interpretability and bias reduction.

7 Conclusion

This study introduced a novel approach to kernel optimization in Quantum Machine Learning (QML) by leveraging a Reinforcement Learning (RL) agent to autonomously design and refine quantum circuits for use in a Quantum Kernel Support Vector Classifier (QSVC). Unlike conventional methods relying on fixed or manually constructed feature maps, the proposed framework enabled the RL agent to explore the space of quantum gates while optimizing both discrete operations and continuous parameters, thereby generating a tailored quantum kernel for integration into a classical Support Vector Machine (SVM).

To rigorously evaluate the proposed RL-QSVC approach, we conducted an extensive comparison against multiple baseline models. First, we performed a comprehensive analysis of classical SVM variants with different kernel functions: linear, RBF, and polynomial, each optimized through systematic grid search with 5-fold cross-validation. The RBF kernel emerged as the best-performing classical approach, achieving 76.39% accuracy with minimal training time (0.06 seconds) on the test set. Subsequently, we compared this optimized classical baseline against both the standard QSVC from Qiskit Machine Learning and our RL-optimized QSVC.

The experimental results on a real-world educational dataset revealed important insights into the quantum advantage landscape. While the standard QSVC underperformed compared to the classical RBF SVM (74.65% vs. 76.39% accuracy), the RL-optimized QSVC demonstrated competitive accuracy (76.73%) while significantly outperforming all baselines in balanced performance metrics. Specifically, the RL-QSVC achieved superior F1-Score (78.79% vs. 74.80%), Precision (79.46% vs. 75.27%), and Recall (79.17% vs. 76.39%) compared to the RBF SVM. This indicates that the RL-QSVC excels at providing balanced classification performance, effectively minimizing both false positives and false negatives a critical advantage in applications where class-balanced performance is essential.

The incorporation of explainability analysis through SHAP values provided additional insights into the model's behavior. By comparing SHAP value distributions between the Linear SVC baseline and the RL-QSVC, we observed how the quantum optimization process reduces bias patterns present in classical linear approaches. Statistical analysis revealed that the RL agent selectively modulated feature influence for specific class-feature combinations, with several features showing statistically significant differences ($p < 0.05$) in their SHAP value distributions. This selective optimization suggests that the RL-QSVC adaptively reweights features to reduce inherent biases and enhance focus on discriminative patterns, contributing to the improved precision and recall observed in the final model.

The primary trade-off of the proposed approach lies in its computational cost during training. The RL optimization process required approximately 14.9 hours, substantially longer than both classical methods (< 0.1 seconds) and the standard QSVC (5.6 minutes). However, this represents a one-time investment during model development, after which the trained quantum circuit can be deployed for inference with computational efficiency comparable to other quantum approaches. For applications where balanced classification performance and bias reduction are priorities, this training cost may be justified by the resulting improvements in model quality and interpretability.

These findings establish that reinforcement learning offers a powerful paradigm for automated quantum kernel optimization in hybrid quantum-classical settings. The study demonstrates that RL-QSVC can achieve performance characteristics distinct from classical approaches, particularly in terms of balanced metrics and reduced bias rather than simply replicating classical results. This work establishes a foundation for future research directions, including: (1) scalability studies to larger and more complex datasets, (2) deployment and validation on physical quantum hardware to assess noise resilience, (3) exploration of multi-objective RL optimization targeting both accuracy and fairness simultaneously, (4) investigation of transfer learning approaches to reduce training time across related tasks, and (5) extension to other quantum-enhanced machine learning architectures beyond SVMs.

The successful integration of RL-QSVC into classical SVMs highlights a promising pathway for advancing hybrid quantum-classical machine learning. By combining the expressive power of parametrized quantum circuits with the automated optimization capabilities of reinforcement learning and the well-established performance of kernel-based methods, this approach opens new avenues for developing quantum machine

learning systems that offer tangible advantages over purely classical alternatives in specific performance dimensions such as balanced classification and interpretability.

Funding

This research received no external funding.

References

[1] Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction, 2nd edn. Adaptive Computation and Machine Learning. The MIT Press, Cambridge, Massachusetts (2018)

[2] Watkins, C.J., Dayan, P.: Q-learning. Machine learning **8**, 279–292 (1992)

[3] Mnih, V., *et al.*: Human-level control through deep reinforcement learning. nature **518**(7540), 529–533 (2015)

[4] Thrun, S., Schwartz, A.: Issues in using function approximation for reinforcement learning. Proceedings of the Fourth Connectionist Models Summer School, 255–263 (1993)

[5] Van Hasselt, H., Guez, A., Silver, D.: Deep reinforcement learning with double q-learning. In: AAAI (2016)

[6] Khalil, E.B., Dai, H., Zhang, Y., Dilkina, B., Song, L.: Learning combinatorial optimization algorithms over graphs. In: Advances in Neural Information Processing Systems, vol. 30, pp. 6348–6358 (2017)

[7] Zhou, Z., Kearnes, S., Li, L., Zare, R.N., Riley, P.: Optimization of molecules via deep reinforcement learning. Scientific Reports **9**(1), 10752 (2019) https://doi.org/10.1038/s41598-019-47148-x

[8] Fösel, T., Niu, M.Y., Marquardt, F., Li, L.: Quantum circuit optimization with deep reinforcement learning (2021). https://arxiv.org/abs/2103.07585

[9] Hidary, J.D.: Quantum Computing: An Applied Approach, 1st edn. Springer, Cham, Switzerland (2019). https://doi.org/10.1007/978-3-030-23922-0

[10] Khrennikov, A.: Quantum computing for complex challenges. Journal of Quantum Information (2021)

[11] Biamonte, J., Wittek, P., Pancotti, e.a. N: Quantum machine learning. Nature **549**(7671), 195–202 (2017)

[12] Cho, C., Chen, C., Chen, e.a. KC: Quantum computation: Algorithms and applications. Chinese Journal of Physics **72**, 248–269 (2021)

[13] Dunjko, V., Taylor, J., Briegel, H.: Quantum-enhanced machine learning. Physical Review Letters **117**(13), 130501 (2016)

[14] Pattanayak, S.: Quantum machine learning for clustering, regression, and classification. Quantum Information Processing (2021)

[15] Alvarez-Rodriguez, U., Lamata, L., Escandell-Montero, e.a. P: Supervised quantum learning without measurements. Scientific Reports **7**(1), 13645 (2017)

[16] Ramezani, M., Karimi, B., Darabi, H.: Quantum machine learning for high-dimensional data. Journal of Quantum Information Science **2020**, 1–15 (2020)

[17] Rebentrost, P., Mohseni, M., Lloyd, S.: Quantum support vector machine for big data classification. Physical Review Letters **113**, 130503 (2013)

[18] Dong, D., Chen, C., Li, e.a. H: Quantum reinforcement learning. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) **38**(5), 1207–1220 (2008)

[19] Lamata, L.: Quantum reinforcement learning. Scientific Reports **7**(1), 13645 (2017)

[20] Mandviwalla, A., Oh, S., Ramesh, R.: Grover's algorithm for quantum machine learning. Quantum Information Processing **17**, 1–23 (2018)

[21] Heredge, J., Hill, C., Hollenberg, e.a. L: Quantum support vector machines for continuum suppression in b meson decays. Computing and Software for Big Science **5**(1), 27 (2021)

[22] Havlíček, V., Córcoles, A., Temme, e.a. K: Supervised learning with quantum-enhanced feature spaces. Nature **567**(7747), 209–212 (2019) https://doi.org/10.1038/s41586-019-0980-2

[23] Hellstem, G.: Hybrid quantum network for classification of finance and mnist data. In: IEEE 18th International Conference on Software Architecture Companion (ICSA-C), pp. 1–4. IEEE, ??? (2021)

[24] contributors, Q.: Qiskit: An Open-Source Quantum Computing Framework (2023). https://qiskit.org/

[25] Heese, R., Gerlach, T., Mücke, S., Müller, S., Jakobs, M., Piatkowski, N.: Explaining quantum circuits with shapley values: towards explainable quantum machine learning. Quantum Machine Intelligence **7** (2025) https://doi.org/10.1007/s42484-025-00254-8

[26] Boser, B.E., Guyon, I.M., Vapnik, V.N.: A training algorithm for optimal margin classifiers. In: Proceedings of the Fifth Annual Workshop on Computational Learning Theory. COLT '92, pp. 144–152. Association for Computing

Machinery, New York, NY, USA (1992). https://doi.org/10.1145/130385.130401 . https://doi.org/10.1145/130385.130401

[27] Suzuki, T., Hasebe, T., Miyazaki, T.: Quantum support vector machines for classification and regression on a trapped-ion quantum computer (2024). https://arxiv.org/abs/2307.02091

[28] Ostaszewski, M., Trenkwalder, L.M., Masarczyk, W., Scerri, E., Dunjko, V.: Reinforcement learning for optimization of variational quantum circuit architectures (2021). https://arxiv.org/abs/2103.16089

[29] Repetto, V., Ceroni, E.G., Buonaiuto, G., D'Aurizio, R.: Quantum enhanced stratification of Breast Cancer: exploring quantum expressivity for real omics data. arXiv. http://arxiv.org/abs/2409.14089 Accessed 2024-11-18

[30] Intel Corporation: Intel Extension for Scikit-learn. https://github.com/intel/scikit-learn-intelex. Accessed: 2023-11-03 (2023)

[31] Roderick, M., MacGlashan, J., Tellex, S.: Implementing the Deep Q-Network (2017). https://arxiv.org/abs/1711.07478

Algorithm 1 RL-Optimized Quantum SVC Training

Input: Training data $(X_{\text{train}}, y_{\text{train}})$, test data $(X_{\text{test}}, y_{\text{test}})$
Input: Feature map Φ_{ZZ}, gate set $\mathcal{G}$, hyperparameters $\{\alpha, \gamma, \epsilon_{\text{start}}, \epsilon_{\text{end}}, \epsilon_{\text{decay}}, \tau, B, M\}$
Output: Optimized quantum circuit C^* and trained QSVC model
Initialize policy network Q_θ and target network $Q_{\theta'}$
Initialize replay memory $\mathcal{D}$ with capacity M
$\theta' \leftarrow \theta$
$steps_done \leftarrow 0$
for episode $= 1$ to N_{episodes} **do**
    $C \leftarrow \Phi_{\text{ZZ}}$
    $s_0 \leftarrow \text{GetState}(C)$
    $list_gates \leftarrow []$, $list_qubits \leftarrow []$
    for step $t = 1$ to $T_{\max}$ **do**
        $\epsilon \leftarrow \epsilon_{\text{end}} + (\epsilon_{\text{start}} - \epsilon_{\text{end}}) \cdot \exp(-steps_done/\epsilon_{\text{decay}})$
        if random() $< \epsilon$ **then**
            $a_t \leftarrow \text{RandomAction}(\mathcal{A})$
        else
            $a_t \leftarrow \arg\max_a Q_\theta(s_t, a)$
        end if
        $(g, q_1, q_2) \leftarrow \text{DecodeAction}(a_t)$
        $\theta_{\text{rot}} \leftarrow Q_\theta^{\text{rotation}}(s_t, a_t)$
        $C \leftarrow C \oplus \text{ApplyGate}(g, q_1, q_2, \theta_{\text{rot}})$
        $list_gates$.append(g)
        $list_qubits$.append(q_1, q_2)
        $K \leftarrow \text{ComputeKernel}(C, X_{\text{train}}, X_{\text{test}})$
        $\text{acc} \leftarrow \text{TrainSVC}(K, y_{\text{train}}, y_{\text{test}})$
        $r_t \leftarrow \text{acc}$
        $s_{t+1} \leftarrow \text{GetState}(C)$
        $done \leftarrow (t = T_{\max}) \vee (\text{acc} \geq \text{acc}_{\text{target}})$
        Store $(s_t, a_t, r_t, s_{t+1}, done)$ in $\mathcal{D}$
        $steps_done \leftarrow steps_done + 1$
        if $|\mathcal{D}| \geq B$ **then**
            Sample mini-batch $\{(s_i, a_i, r_i, s'_i, d_i)\}_{i=1}^{B}$
            $Q_{\text{current}} \leftarrow Q_\theta(s_i, a_i)$
            $a'_{\text{best}} \leftarrow \arg\max_{a'} Q_\theta(s'_i, a')$
            $Q_{\text{target}} \leftarrow r_i + (1 - d_i)\gamma Q_{\theta'}(s'_i, a'_{\text{best}})$
            $\mathcal{L} \leftarrow \text{SmoothL1}(Q_{\text{current}}, Q_{\text{target}})$
            $\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}$
            $\theta' \leftarrow \tau\theta + (1 - \tau)\theta'$
        end if
        if *done* **then**
            break
        end if
    end for
    if *episode* mod $N_{\text{target}} = 0$ **then**
        $\theta' \leftarrow \theta$
    end if
end for
return C^*, trained QSVC model