Quantum Re-Uploading for Calorimetry: Optimized Architectures with Extended Expressivity

Léa Cassé[1,2], Bernhard Pfahringer[1], Albert Bifet[1], Frédéric Magniette[2]

[1]The Artificial Intelligence Institute, University of Waikato, Knighton Road, Hamilton, 3240, State, New Zealand.
[2]Laboratoire Leprince-Ringuet, Institut Polytechnique de Paris, Route de Saclay, Palaiseau, 91128, Paris, France.

Contributing authors: lc480@students.waikato.ac.nz; pfahringer.bernhard@waikato.ac.nz; albert.bifet@waikato.ac.nz; frederic.magniette@llr.in2p3.fr;

Abstract

We present an application of a single-qubit Quantum Re-uploading Unit (QRU) model for particle classification tasks in calorimetric experiments, making it highly suitable for current quantum hardware constraints. Evaluated on a novel particle physics dataset, we demonstrate robust classification accuracy in a realistic use case scenario rather than relying on generic benchmarks. We explored hyperparameters tuning analysis and through global optimization methods, we identify significant hyperparameter correlations, enhancing the understanding of their interdependencies and relative importance. Furthermore, our mathematical analysis establishes that the QRU is more expressive than widely-used QML methods such as Variational Quantum Circuit (VQC). These insights show the QRU as a superior choice for practical quantum machine learning tasks.

Keywords: Quantum machine learning, Quantum re-uploading, Hyperparameter optimization, Particle classification, Fourier analysis, Expressivity, Calorimetry.

1 Introduction

The potential of quantum machine learning (QML) [1] to resolve challenging issues in a variety of fields, including particle physics [2, 3] has attracted a lot of attention. The main concept of QML is to use the properties of quantum mechanics to perform multiple calculations at once, which could result in quicker classification, regression, or clustering tasks [4, 5]. In terms of quantum computing [6], we are still in the Noisy Intermediate-Scale Quantum (NISQ) era [7], characterized by limited qubit counts and shallow circuit depths, posing substantial challenges for QML applications dealing with high-dimensional and nonlinear data.

Under these restrictions, the Data Re-Uploading (QRU) model, introduced by [8], can become an attractive proposition. With the QRU model, data is encoded iteratively as rotation parameters over single-qubit circuits. This is leading to improved expressivity of the model while keeping its qubit requirements low. The QRU structure alternates data-encoding gates and trainable parameterized gates, It could be mathematically represented as:

$$U(\theta, x) = \prod_{j=1}^{M} e^{ig_j x} W_j e^{iV_j \theta_j}, \tag{1}$$

where x represents the classical data to be encoded, g_j and V_j are traceless Hermitian operators defining the data encoding and trainable gates, respectively and W_j are fixed unitary transformations. Unlike quantum neural networks [9] and kernel quantum methods [10], the QRU has the universal approximation property (i.e., density of approximation with incrementally encoded data), allowing it to represent complex nonlinearities that are important [11].

Here, we implement and assess a single-qubit QRU for a classification task that makes use of a new simulated dataset-useful in high-energy particle studies. This dataset, never used previously in the literature, contains different classes (electrons, muons and pions) making it ideal to serve as a benchmark for multi-class classification problems that are useful for evaluating QRU performance. The datasets of high-energy particles is chosen in accordance to the recent effort towards classification and prediction of energy with QML, especially for high-dimensional datasets like LHC [2, 12]. We investigate the performance of the model on this dataset, reporting classification accuracy, loss and simulation environment execution time.

An integral part of our work is the systematic search and tuning of QRU hyperparameters (circuit depth, learning rate, batch size, etc.). Due to the error sensitivity of NISQ devices, there is a direct correlation between each hyperparameter and the accuracy and stability of the model [13]. We investigate both empirical tuning to enhance model efficiency as well as preliminary tests of global optimization methods.

The main contribution of this study is a thorough analysis of the QRU performance, particularly suited to the NISQ era due to its single-qubit design. We evaluate

this architecture on a previously unpublished dataset, applying it to a realistic use case rather than a generic dataset and report strong performance results for such a compact quantum circuit. Our extensive investigation into the hyperparameters of both the quantum model and training procedure elucidates their interdependencies and ranks their relative impact, allowing practitioners to finely tune the QRU according to specific criteria.

Additionally, our mathematical analysis establishes that the QRU is inherently more expressive than the widely-used Variational Quantum Circuit (VQC) [14], highlighting VQC's limitations not only due to qubit requirements and noisy entanglement gates but also in term of Fourier frequency boundary, thereby justifying our preference for QRU in this research.

2 Theory

To understand the reason why we chose QRU over other QML algorithms, we compare the expressive capabilities of two quantum architectures with the same number of trainable parameters θ:

- A VQC with $n = 3$ qubits and L variational layers, each composed of CNOT gates and single-qubit Euler rotations. A single data encoding layer applies $RY(x_j)$ gates once to each qubit.
- A QRU with $n = 1$ qubit and L repeated data and parameter blocks. Each block applies the following rotation sequence for each input dimension x_j:

$$RX(\theta_2)\, RY(\theta_1 \cdot x_j)\, RZ(\theta_0)$$

This is totaling $9L$ trainable parameters in both models.

Lemma. *Let each coordinate x_j be encoded K_j times through single-parameter rotations of the form $RY(\theta_{k,j} \cdot x_j)$, where $\theta_{k,j} \in \mathbb{R}$ is a trainable parameter. Then the Fourier expansion of the output of the circuit $f(\mathbf{x})$ contains only terms:*

$$f(\mathbf{x}) = \sum_{\mathbf{n} \in \mathcal{F}} c_{\mathbf{n}}\, e^{i(n_1 x_1 + \cdots + n_d x_d)}, \tag{2}$$

with $\mathcal{F} \subseteq \mathbb{Z}^d$ and

$$n_j \in \mathrm{Span}_{\mathbb{Z}}\{\theta_{1,j}, \ldots, \theta_{K_j,j}\} \tag{3}$$

$$\Rightarrow \quad |n_j| \leq \sum_{k=1}^{K_j} \left|\theta_{k,j}\right|. \tag{4}$$

Implication. The Fourier support $\mathcal{F}$ of a QRU grows linearly with the number of re-uploads L, while that of a VQC remains bounded by the single encoding layer: $K_j = 1$ for all j.

For example, with $d = 1$, we have:

$$\mathcal{F}_{\text{VQC}} \subseteq \{-1, 0, 1\}, \tag{5}$$

$$\mathcal{F}_{\text{QRU}} \subseteq \{-L, ..., 0, ..., L\}. \tag{6}$$

This means that the QRU can, in principle, express functions with higher-frequency components than any mono-encoding VQC, regardless of the number of qubits.

Figure 1 confirms the lemma visually: the QRU accesses a wider harmonic spectrum, providing a strictly larger function class than the VQC. This justifies the use of QRUs in applications where high-frequency resolution or functional flexibility is needed.

To corroborate the Fourier Support Lemma, we performed the following experiment for $d = 3$, $L = 4$, $M = 100$ (number of random set of parameters) and $N = 96$ points on the diagonal $\mathbf{x} = (u, u, u)$ with $u \in [-1, 1]$:

$$p_n = \frac{1}{M} \sum_{k=1}^{M} \mathbf{1}\big(|c_n^{(k)}| > \varepsilon\big), \tag{7}$$

$$c_n^{(k)} = \frac{1}{N} \sum_{t=0}^{N-1} f\big(x_t; \theta^{(k)}\big)\, e^{-i2\pi nt/N}, \tag{8}$$

where $\theta^{(k)}$ is the k-th random draw of trainable parameters and $\varepsilon = 0.05$.

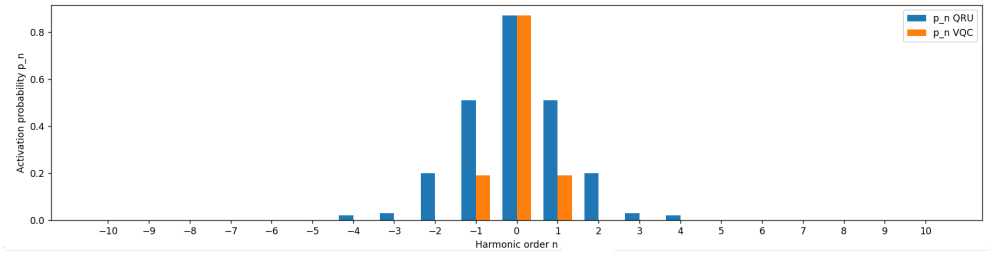


Figure 1: Empirical Fourier amplitudes for QRU (blue) and VQC (orange). Both models use $9L$ trainable parameters.

The QRU empirically activates all harmonics up to order L, whereas the mono-encoded VQC occasionally fails to activate higher-order terms, in perfect agreement with the lemma and the analytical spectra of Figure 1.

3 Methodology

This section details the datasets, model configuration and experimental setup for evaluating QRU on a quantum simulator. We describe the data sources, baseline setups and computational environment we used in our study.

3.1 Dataset

The data used in this study are extracted from the OGCID/D2 simulation dataset [15]. It is a set of single particle interaction with a simplified highgranularity calorimeter inspired by CMS/HGCal [16].

This detector architecture features two main sections: the electromagnetic calorimeter (ECAL) and the hadronic calorimeter (HCAL), optimized for detecting particles of varying energies. The ECAL comprises 26 layers of lead absorbers (6.05 mm thick), while the HCAL has 24 layers of stainless steel absorbers, with 12 layers of 45 mm thickness and 12 of 80 mm, providing high resolution in capturing energy deposits. Active silicon layers are placed between absorbers, segmented into hexagonal cells with a thickness of 0.32 mm and a transverse area of approximately 1 cm2 for ECAL and 4 cm2 for HCAL [3].

The D2 dataset is a collection of simulated interactions between this detector and four types of particles (muon, positive pions, electrons and photons [2, 12]) at different energies and with a flat incidence angle (aligned with the longitudinal detector axis). The dataset includes detailed information on each particle event, namely the energy measurement in each cell but also extracted variables of interest describing the geometry and the energy repartition of the hits.

For this study, only one file of electrons, pions and muons has been used $*\_E10-100\_theta0$) and specifically some variables of interest known for their discriminating power : the total energy deposited in the ECAL, the length of the particle interaction shower and the standard deviation of the energy deposits in the HCAL.

3.1.1 Execution environment

Experiments were run on a cloud server with both CPU and GPU resources, utilizing the `default.qubit` simulator and testing PennyLane Lightning for potential GPU acceleration. Although smaller circuits showed a 2.5x speedup on GPU, no significant gains were observed for larger circuits, likely due to shared cloud resources.

4 Results

This section presents the performance results of the QRU model on the particle classification task. We first analyze the impact of individual hyperparameters, organized into model-related and training-related parameters. Then, we explore the effects of global optimization techniques, including Bayesian optimization and initial tests using Hyperband and investigate correlations between key hyperparameters.

4.1 Impact of individual hyperparameters

This subsection examines the effect of model-related hyperparameters, specifically circuit depth, input normalization, rotation gates and the number of trainable parameters per input on the classification accuracy and computational efficiency of the QRU model.

In order to assess the variability of the QRU model, we analyzed the test accuracy and loss across 50 independent runs. For each run, we reshuffled the dataset and randomly initialized the parameters θ following a Gaussian distribution centered around 0.5. The plots below illustrate the test accuracy and loss for these runs:

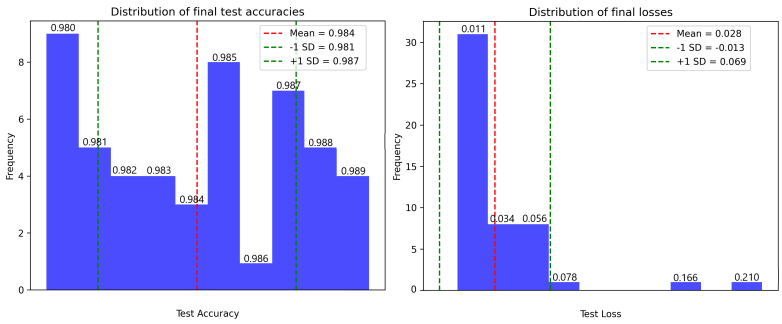


Figure 2: Left: Final test accuracy; right: Final loss over 50 runs

For the test accuracy, the mean is 0.98 with a variability of 0.002881, indicating that the results are highly consistent and centered around this value. On the test losses side, the mean is 0.028 with a variability of 0.041.

Having established the variability in the QRU's performance for classification task, we now move on to analyze how the model behaves when adjusting the depth of the quantum circuit, a critical hyperparameter for model expressivity and learning capacity.

4.1.1 Model hyperparameters

3.1.1.1 Circuit depth

The depth of the circuit represents the number of times the data is re-uploaded into the quantum circuit. It's important to note that a depth of 1 represents a quantum circuit that has not re-uploaded the input and that means it's not technically a QRU.

Figure 3 shows the evolution of the loss over 30 epochs for different values of depth. We observe a significant decrease in the loss as the depth is increasing, particularly

between depths 1 and 3, where the loss drops from 0.2558 to less than 0.0667. Beyond a depth of 4, the loss stabilizes around 0.03, indicating that increasing the depth beyond this threshold does not significantly improve the model's performance in terms of loss minimization.

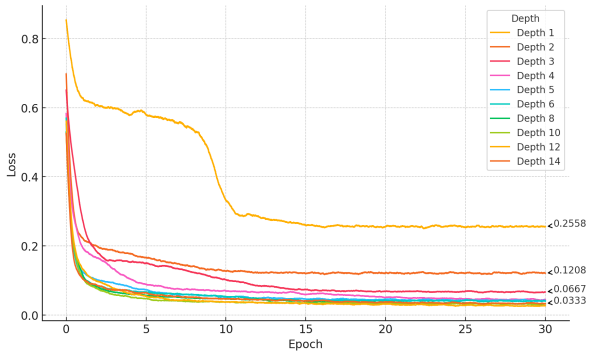


Figure 3: Evolution of the loss over epochs for different circuit depths.

Next, Figure 4 shows the evolution of test accuracy over epochs for different depth values. Similarly, we observe a rapid improvement in accuracy with shallow depths (from depth 1 to depth 3), with accuracy approaching 0.98 from a depth of 4 onwards. Again, higher depths yield only marginal gains in accuracy.

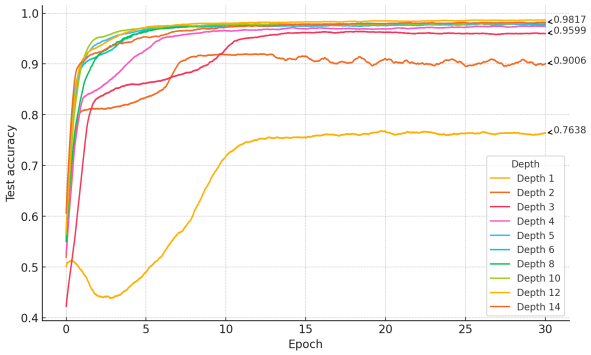
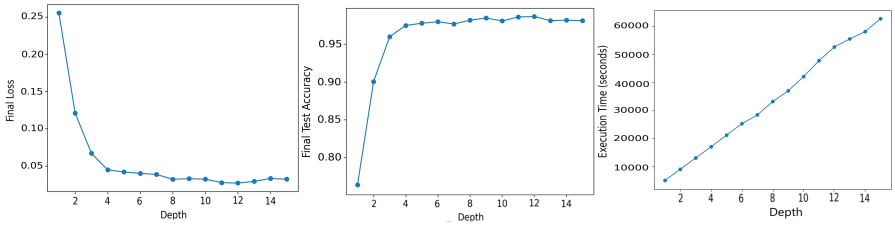


Figure 4: Evolution of test accuracy over epochs for different circuit depths.

The following three graphs (Figure 5) respectively present the execution time, final lossand final accuracy as a function of the circuit depth.



(a) Final loss vs depth (b) Final test accuracy vs depth (c) Execution time vs depth

Figure 5: Comparison of execution time, final loss and final accuracy as a function of circuit depth

We observe a near linear increase in execution time with depth (Figure 5c), with the time exceeding 60,000 seconds for a depth of 14, which is approximately 16 hours. In terms of final loss (Figure 5a), the results show a marked improvement between depths 1 and 4, followed by stabilization from depth 5 onwards. Then, the final test accuracy (Figure 5b) follows a similar trend, plateauing around a depth of 4 with a final accuracy close to 0.98.

These results show that while increasing the circuit depth beyond 4 yields only marginal performance improvements, it comes at a significant cost in execution time. Therefore, it is essential to find a balance between circuit depth and computational efficiency for practical applications.

These results are consistent with findings from the literature, such as [8], which show that increasing the number of re-uploading layers can enhance model expressivity. Importantly, this highlights that expressivity is not simply proportional to the Hilbert space dimension or circuit depth. Instead, it depends on multiple factors including layer depth, gate structure, and the ability of parameter optimization to exploit the non-linear transformations introduced by data re-uploading:

$$\text{Expressivity} \sim g(L) \cdot h(U(\phi, \mathbf{x}), \theta), \tag{9}$$

where $g(L)$ is a non-linear function that accounts for the diminishing returns of the increasing number of layers we re-upload L, $U(\phi, \mathbf{x})$ is the parameterized unitary gates encoding the data and θ represents the tunable parameters of the circuit. Here $\dim(\mathcal{H}) = 2$ as it is for a single qubit and the expressivity is dominated primarily by the competition between the number of re-uploaded layers L and the gate structure.

As seen in our experiments, while the performance improves significantly for depths up to 4, further increases in depth provide only marginal gains. This can be explained by the bias-variance tradeoff [17]: as we increase the depth, the model gains expressivity and reduces bias, but the improvements become smaller. This is likely because our classification problem is relatively simple and the additional complexity introduced by deeper circuits does not translate into significant performance

improvements. The expressivity gained from re-uploading becomes less impactful once the model complexity exceeds the requirements of the task. Therefore, there is a practical upper bound to the benefits of increasing depth in this case.

Having examined the impact of circuit depth on the QRU's performance, we now turn our attention to the effect of the input normalization.

3.1.1.2 Input normalization

In this section, we analyze the effect of input normalization on the performance of the QRU model. Specifically, we compare two normalization ranges: 0 to 2π and $-\pi$ to π. The results are presented in terms of final loss, final test accuracyand trainability.

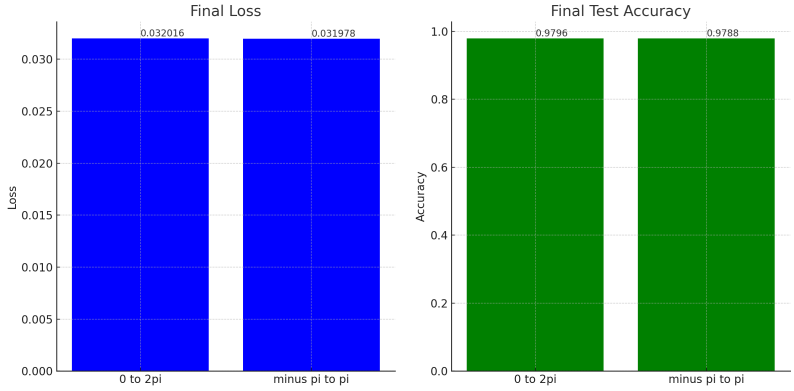


Figure 6: Comparison of final loss and final test accuracy for input normalization ranges 0 to 2π and $-\pi$ to π

As shown in Figure 6, there is no significant difference in performance between the two normalization ranges. The final mean loss is 0.032016 for 0 to 2π and 0.031978 for $-\pi$ to π, indicating virtually identical outcomes. Similarly, the final test accuracy is 0.9796 for 0 to 2π and 0.9788 for $-\pi$ to π, again showing no meaningful discrepancy.

The lack of a significant difference can be attributed to the fact that our quantum circuits γ, which govern the behavior of the QRU, are symmetric functions of the input over a period of 2π. This symmetry is highlighted in the circuit fits shown in Figure 7, where we observe similar patterns in the behavior of the circuit for 9 different configurations of θ over both normalization intervals.

Hypothesis functions [18] generated by QRU models are defined as the expectation value of a measurement observable Z applied to the output quantum state of the circuit. Mathematically, they are given by:

$$h_\theta(x) = \langle 0|U^\dagger(\theta, x)ZU(\theta, x)|0\rangle, \tag{10}$$

where $U(\theta, x)$ is the parameterized quantum circuit that incorporates data encoding through iterative layers of single-qubit rotations. These hypothesis functions can be further expressed as generalized trigonometric polynomials:

$$h_\theta(x) = \sum_{\omega\in\Omega} a_\omega(\theta)e^{i\omega x}, \tag{11}$$

where Ω is the set of available frequencies determined by the data encoding scheme and $a_\omega(\theta)$ are Fourier coefficients dependent on the trainable parameters θ.

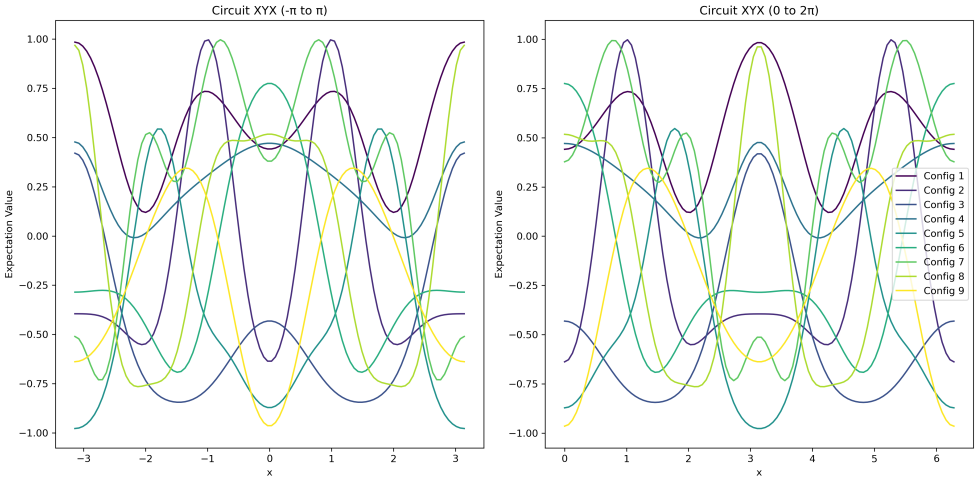


Figure 7: Hypothesis functions $h_\theta(x)$ for 9 configurations of θ for the circuit $R_x - R_y - R_x$ over the intervals $-\pi$ to π (left) and 0 to 2π (right) with a depth of 10

This graph shows that shifting the normalization range by π to the right results in fit functions that remain even functions, exhibiting similar behavior to those normalized to the left of π. As such, the expressivity does not change, which explains the lack of noticeable differences between the two ranges.

Previously, we tested the QRU model on non-normalized data and the model failed to learn properly. It could be attributed to the fact that the non-normalized values, particularly for features 2 and 3, are extremely small, as seen in Figure 8.

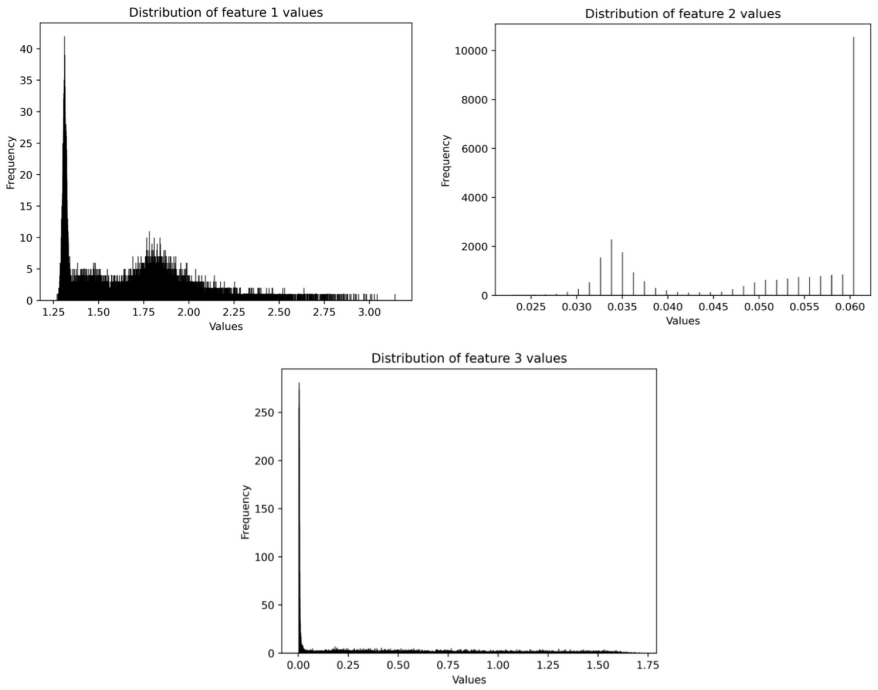


Figure 8: Feature distributions

This can be explained by considering the quantum state of the qubit: when the input values are very small, the qubit undergoes minimal rotations on the Bloch sphere, making it difficult to differentiate between states when the quantum measurement is performed along the Z-axis.

3.1.1.3 Rotation gates

In this experiment, we analyzed the impact of different types of quantum rotations on the performance of the QRU model. We compared six circuits based on successive rotations of types R_x, $R_y$and R_z, with the following structure:

$R_x - R_y - R_x : |q\rangle$ — $R_X(\theta_i)$ — $R_Y(\theta_{i+1} \times x_j)$ — $R_X(\theta_{i+2})$ — c

$R_x - R_z - R_x : |q\rangle$ — $R_X(\theta_i)$ — $R_Z(\theta_{i+1} \times x_j)$ — $R_X(\theta_{i+2})$ — c

$R_y - R_x - R_y : |q\rangle$ — $R_Y(\theta_i)$ — $R_X(\theta_{i+1} \times x_j)$ — $R_Y(\theta_{i+2})$ — c

$R_y - R_z - R_y : |q\rangle$ — $R_Y(\theta_i)$ — $R_Z(\theta_{i+1} \times x_j)$ — $R_Y(\theta_{i+2})$ — c

$R_z - R_x - R_z : |q\rangle$ — $R_Z(\theta_i)$ — $R_X(\theta_{i+1} \times x_j)$ — $R_Z(\theta_{i+2})$ — c

$R_z - R_y - R_z : |q\rangle$ — $R_Z(\theta_i)$ — $R_Y(\theta_{i+1} \times x_j)$ — $R_Z(\theta_{i+2})$ — c

The performance of each type of circuit was evaluated in terms of final loss and final test accuracy. Figure 9 below illustrates the results obtained for these metrics.

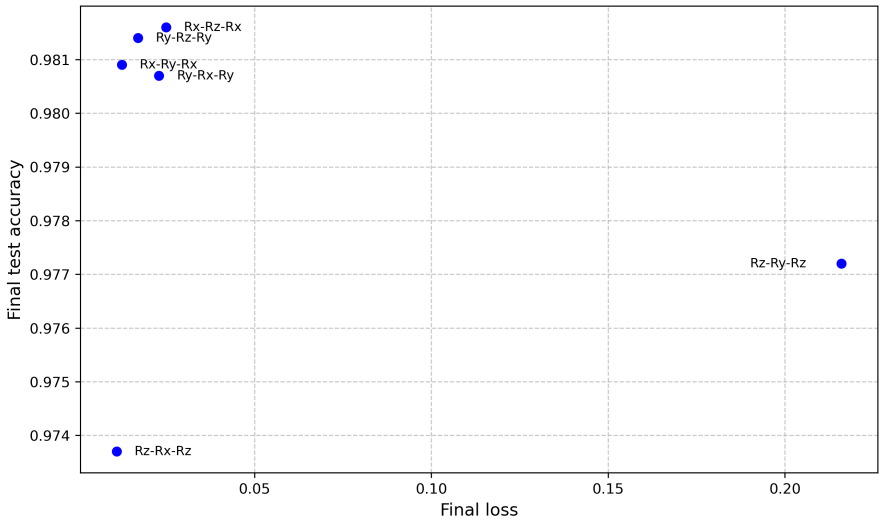


Figure 9: Performance of different circuits based on rotation types

The results show that the $R_x - R_y - R_x$ circuit achieves the best performance in terms of final loss, indicating that this combination of rotations allows for better optimization of the model. On the other hand, the $R_z - R_y - R_z$ circuit has the lowest final test accuracy, significantly lower than the other circuits. Conversely, the $R_x - R_z - R_x$ circuit and the $R_y - R_x - R_y$ circuit achieve high accuracy, with relatively low loss, showing that these rotation configurations offer a good balance between accuracy and stability. However, the $R_y - R_z - R_y$ circuit shows a higher final loss, suggesting that this type of rotation may introduce larger fluctuations in parameter optimization. Overall, the results indicate that rotations involving only R_x and R_y are more stable and perform better than those involving R_z.

Hubregtsen et al. (2021) [19] explored the correlation between the capacity of parameterized quantum circuits (PQC) to explore the Hilbert space and classification accuracy. They define a measure of expressivity, which quantifies the ability of a PQC to uniformly explore the Hilbert space, through the Kullback-Leibler divergence (DKL) between the fidelity distribution of the quantum states generated by the PQC and that of random Haar states. The expressivity is given by the following equation:

$$\text{Expr} = D_{\text{KL}}(P_{\text{PQC}}(F;\Theta)||P_{\text{Haar}}(F)), \tag{12}$$

where P_{PQC} is the fidelity distribution of the PQC and P_{Haar} is that of the random Haar states. The article concludes that rotations R_X, $R_Y$and R_Z should be used together to maximize expressivity and therefore accuracy. None of these rotations is intrinsically superior, but their combination allows full exploration of the Hilbert space. This is directly related to the following theorem:

Theorem 1 (Euler's theorem on rotations)

In a three-dimensional space, any rotation can be represented by a combination of elementary rotations around the x, $y$and z axes, respectively denoted as R_x, $R_y$and R_z. More precisely, any rotation can be expressed as a single rotation by an angle θ around a fixed axis, according to the following decomposition:

$$R(\theta, \hat{n}) = R_z(\phi)R_y(\Theta)R_x(\psi) \tag{13}$$

where θ is the total rotation angle and $\hat{n}$ is the unit vector describing the rotation axis.

Nevertheless, our results show that the combination of R_x and R_y gates in quantum data re-uploading circuits tends to offer better performance in terms of accuracy and loss minimization compared to R_x. This may simply arise from the fact that during a measurement, we project our quantum state onto the Z-axis and therefore, rotating around the Z-axis does not change the projection of the qubit on the Z-axis.

This pushes us to test other types of rotations, which we could call "triplets" ($R_x - R_y - R_z$), involving rotations around all three axes, rather than using simpler "sandwich" rotations like in our case. We plotted the QRU functions for the same depth and the same combination of theta parameters for a "sandwich" circuit and a "triplet" circuit. These results are presented in Figure 10.

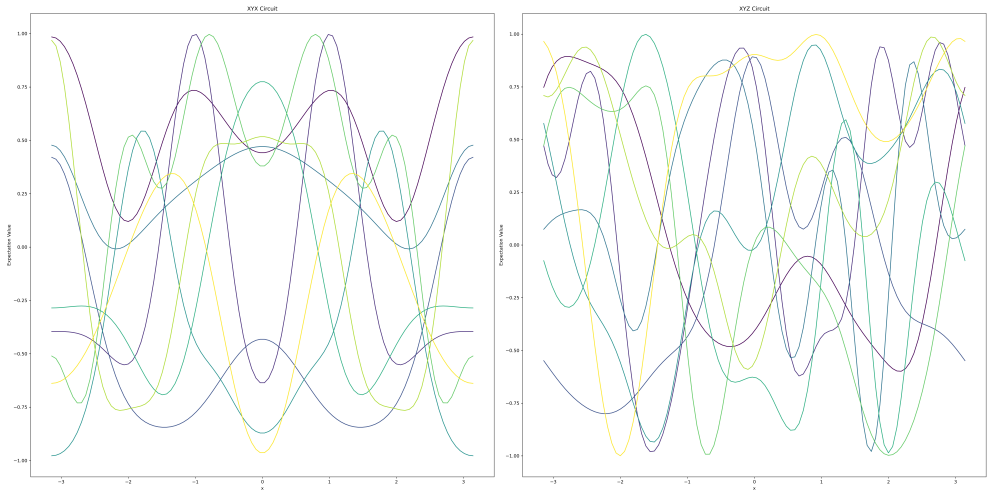


Figure 10: Hypothesis functions $h_\theta(x)$ for 9 configurations of θ for the circuit $R_x - R_y - R_x$ (left) and $R_x - R_y - R_z$ (right).

We observe that "sandwich" circuits are constrained to be even functions of the input, whereas "triplet" circuits have more flexibility. Theoretically, this should result in better expressivity. Figure 11 shows the results obtained with $R_x - R_y - R_x$ quantum circuit.

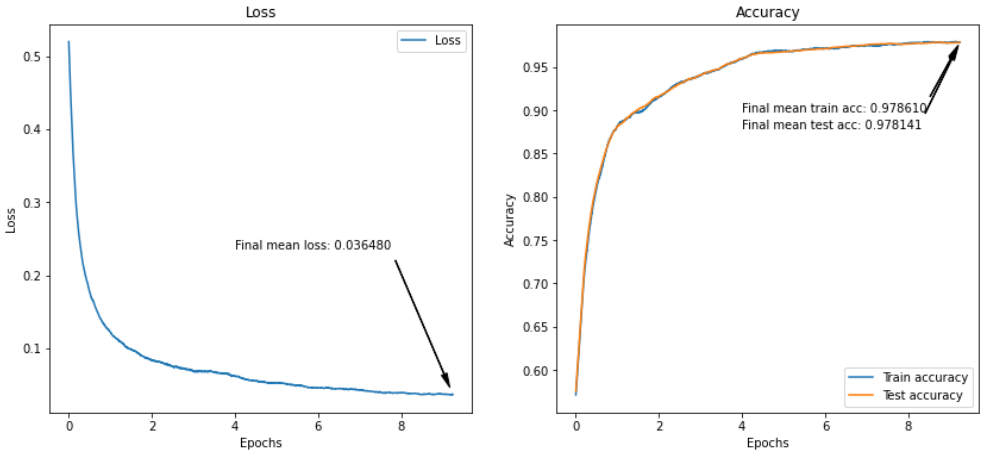


Figure 11: Performance of the "triplet" quantum circuit

The final mean loss is 0.036480, with a final mean train accuracy of 0.978610 and test accuracy of 0.978141. Comparing this with the "sandwich" circuits using only

two rotations, such as $R_x - R_y - R_x$ or $R_x - R_z - R_x$, where the final accuracies were slightly higher and with lower final losses, it becomes evident that the two-rotation circuits tend to outperform in minimizing loss in our case. While three-rotation circuits like $R_x - R_y - R_z$ theoretically offer higher expressivity, in this specific case, the increased complexity does not significantly improve accuracy, but instead results in a slightly higher loss. This suggests that two-rotation circuits provide a simpler yet equally effective approach for our type of classification tasks, balancing between expressivity and optimization stability.

Next, we analyze how the number of trainable parameters per input affects the model's performance.

3.1.1.4 Number of trainable parameters per input

In this part, we analyze the impact of varying the number of parameters θ per input on the performance of the QRU model. We tested five different quantum circuit configurations, each with an increasing number of parameters per input, as described below:

1 parameter θ per input:

$|q\rangle$ — $R_X(\theta_i)$ — $R_Y(x_j)$

2 parameters θ per input:

$|q\rangle$ — $R_X(\theta_i)$ — $R_Y(x_j)$ — $R_X(\theta_{i+1})$

3 parameters θ per input:

$|q\rangle$ — $R_X(\theta_i)$ — $R_Y(\theta_{i+1} \times x_j)$ — $R_X(\theta_{i+2})$

4 parameters θ per input:

$|q\rangle$ — $R_X(\theta_i)$ — $R_Y(\theta_{i+1} \times x_j + \theta_{i+2})$ — $R_X(\theta_{i+3})$

5 parameters θ per input:

$|q\rangle$ — $R_X(\theta_i)$ — $R_Y(\theta_{i+1}^2 \times x_j + \theta_{i+2} \times x_j + \theta_{i+3})$ — $R_X(\theta_{i+4})$

The results of these tests are shown in Figure 12.

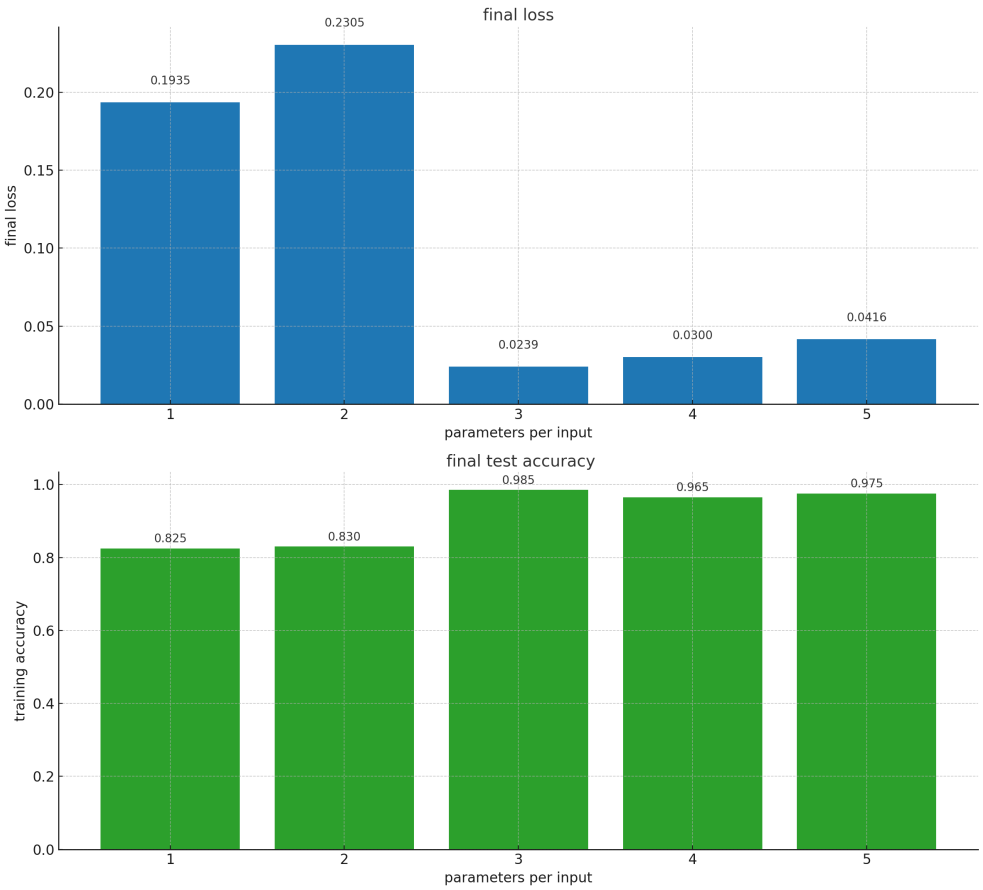


Figure 12: Performance of the QRU model based on the number of parameters per input. Top: final loss; Bottom: final test accuracy.

As seen in Figure 12, the final loss decreases dramatically when moving from 1 or 2 parameters per input (0.1935 and 0.2305, respectively) to 3 parameters, where the final loss drops to 0.0239. However, after 3 parameters per input, the final loss slightly increases again for 4 and 5 parameters (0.0300 and 0.0416, respectively). This indicates that increasing the number of parameters per input beyond 3 does not significantly improve loss minimization and may even complicate optimization.

Similarly, the final test accuracy improves from 0.825 with 1 parameter to 0.985 with 3 parameters. However, as with the loss, the accuracy plateaus and slightly decreases with 4 and 5 parameters (0.965 and 0.975). This suggests that 3 parameters per input represent the optimal configuration for balancing model complexity and performance in terms of both accuracy and loss. In summary, while optimizing the number of parameters per input improves the expressivity and performance of the

quantum circuit, it also increases the computational complexity. In the next section, we explore the use of GPU acceleration to reduce execution time of the QRU model.

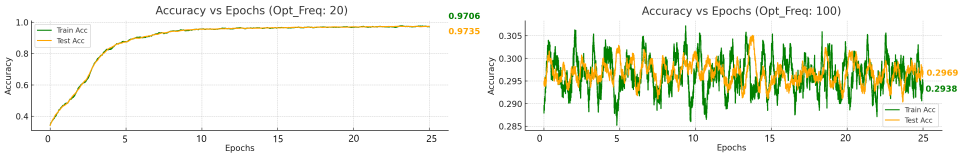
After evaluating the QRU model's performance based on individual hyperparameter variations, we now move to a more comprehensive approach by considering the optimization of all hyperparameters simultaneously.

4.1.2 Training hyperparameters

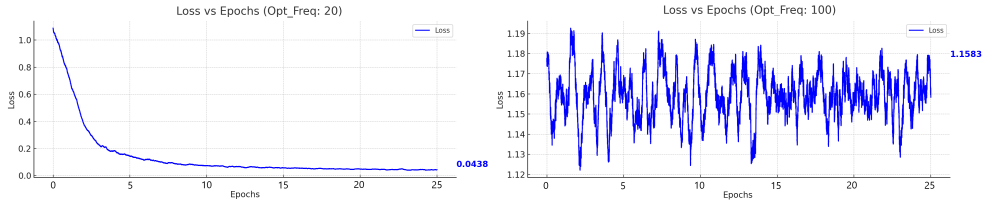
Here, we explore the impact of training-related hyperparameters, including batch size, optimizer, loss function and learning rate, on model convergence, stability and performance.

3.1.2.1 Batch Size

The idea behind using different batch sizes is to accumulate gradients over a certain number of examples before performing an optimization step. As shown in the Figure 13a and 13b, the results vary significantly depending on the batch size. With a batch size of 20, the final test accuracy reaches 0.9735, with a final loss of 0.0438, showing stable and efficient behavior. However, when the batch size is increased to 100, the test accuracy drops dramatically to 0.2969 and the final loss rises to 1.1583, suggesting that the model struggled to learn.



(a) Comparison of the accuracy over epochs for a batch size of 20 data (left) and 100 data (right).



(b) Comparison of the test loss over epochs for a batch size of 20 data (left) and 100 data (right).

Figure 13: Comparison of the loss and accuracy over epochs for a batch size of 20 data (left) and 100 data (right).

Table 1 below summarizes the results for different batch sizes. The trainability metric, calculated as the area under the loss curve during training, measures how easily the model can learn. It is calculated using the following equation:

$$\text{Trainability} = \int_0^E \text{Loss}(e)\, de \tag{14}$$

E denotes the total number of training epochs. A high trainability value indicates that the model maintains a high loss during training, which means it struggles to converge. Conversely, a lower trainability value suggests better convergence, as the loss decreases rapidly during training.

Batch	Train acc	Test acc	Trainab.	Time (s)	Loss
1	**0.980**	**0.980**	**0.066**	259,282	**0.031**
5	0.978	0.977	0.103	256,281	0.035
20	0.973	0.973	0.192	236,999	0.043
100	0.298	0.297	1.695	**232,299**	1.159

Table 1: Metrics for different batch sizes.

Batch size plays in quantum model training [20] and smaller batch sizes generally lead to slower convergence but achieve good final accuracy with more iterations. Larger batch sizes, on the other hand, tend to converge faster and avoid barren plateaus [21], but may not always result in the best final accuracy. We can describe gradient variance with equation 15:

$$O = \frac{1}{N} \sum_{i=1}^{N} Z_i \otimes 1_{\bar{i}} \tag{15}$$

where Z_i is the Pauli-Z operator applied to the i-th qubit and $1_{\bar{i}}$ is the identity operator on the other qubits. This equation shows that increasing the batch size can reduce gradient variance, helping the model avoid barren plateaus and improving the stability of the training process.

Our experiments show a departure from the theoretical expectation: while smaller batch sizes achieve the best final accuracy, larger batches (e.g., 100) lead to significantly worse performance, with the model failing to optimize. We attribute this to the relative simplicity of our classification task. In such cases, the stochasticity introduced by small batches provides a useful regularization effect, whereas large batches reduce this stochasticity and hinder effective learning. This suggests that, for simple tasks, smaller batches or even sample-wise updates are preferable despite higher computational cost. A systematic study of this phenomenon will be the subject of future work.

We now turn to the analysis of the optimizer, a key hyperparameter that influences the model's convergence speed and overall performance.

3.1.2.2 Optimizer

We compare here the performance of different optimizers, including SGD, RMSProp and Adam, based on their impact on the performances of the QRU. Table 2 summarizes the results of the three different optimizers.

Optimizer	Execution time (s)	Final test accuracy	Final loss	Trainability
RMSprop	**142,569.70**	0.844	0.109	**0.181**
SGD	21,217.02	0.334	1.024	2.987
Adam	143,880.52	**0.984**	**0.0207**	0.776

Table 2: Performance comparison of optimizers.

SGD (Stochastic Gradient Descent) is the simplest optimizer, updating parameters using only the gradient of the loss function. It calculates the gradient for each batch and updates the parameters as follows:

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta J(\theta_t) \tag{16}$$

where θ_t is the model parameter at step t, η is the learning rate and $J(\theta_t)$ is the loss function.

RMSProp (Root Mean Square Propagation) adapts the learning rate by dividing it by a moving average of the square of the gradients. This helps RMSProp handle problems where the magnitude of the gradients varies greatly:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{v_t + \epsilon}} \nabla_\theta J(\theta_t) \tag{17}$$

where v_t is the exponentially weighted moving average of the squared gradientsand ϵ is a small value to prevent division by zero.

Adam (Adaptive Moment Estimation) combines ideas from both RMSProp and momentum. It calculates the exponentially weighted average of both the gradients and the squared gradients:

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1 - \beta_1) g_t \\ v_t &= \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \end{aligned} \tag{18}$$

where m_t and v_t are the first and second moment estimates (similar to momentum), β_1 and β_2 are decay rates and g_t is the gradient. The parameter update is then given by:

$$\theta_{t+1} = \theta_t - \eta \frac{m_t}{\sqrt{v_t} + \epsilon} \tag{19}$$

SGD shows the lowest runtime per epoch, as expected from its simpler update rule. However, with the learning rate used in our experiments it reached lower

final accuracy and higher loss than RMSProp and Adam. This likely reflects the greater sensitivity of SGD to learning-rate choice, whereas adaptive methods such as RMSProp and Adam are more robust. RMSProp converges faster than SGD but underperformed Adam in final accuracy and loss. Adam achieved the best overall performance, at the cost of a slightly higher runtime due to the additional moment estimates.

The figure below (Figure 14) compares several optimizers derived from Adam, including Adamax, Nadam, Adagrad, Adadelta and AdamW.

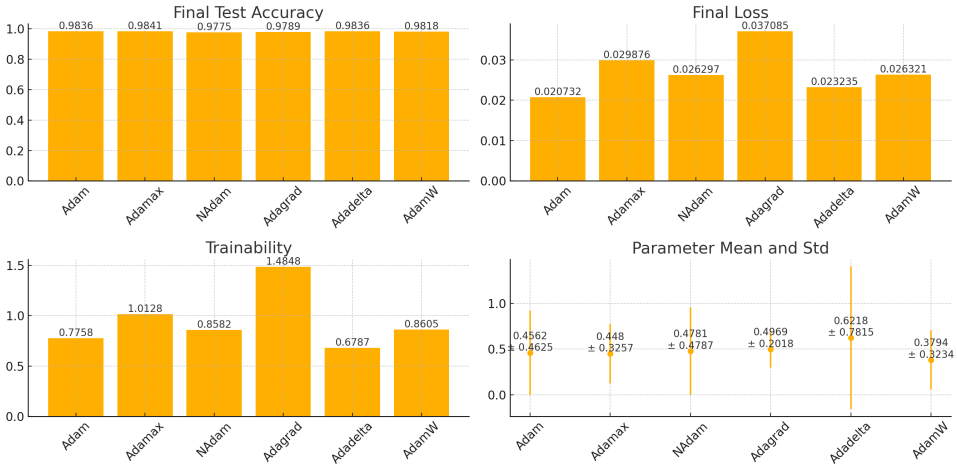


Figure 14: Comparison of Adam-derived optimizers across metrics

Each Adam variant brings slight modifications to the original algorithm. Adamax uses the infinity norm (instead of the second norm), making it more stable in certain cases. Nadam introduces Nesterov momentum into Adam, allowing for faster convergence in some settings. Adagrad adapts the learning rate based on the history of gradients, making it suitable for sparse data but leading to slower learning in the long run. Adadelta dynamically adjusts the learning rate without needing an initial learning rate, making it more resilient to noisy data. And finally, AdamW modifies the weight decay method to achieve better generalization. As seen in the results, the performance of these optimizers is closely tied to the specific characteristics of the quantum circuit. Adagrad, for example, shows higher trainability so it suffers from slower learning, which is reflected in its higher final loss.

With the optimizer performance thoroughly analyzed, we now turn our attention to the loss function used for the training.

3.1.2.3 Loss Function

We explored three different loss functions: L1, L2 and Huber. These are used to measure the discrepancy between predicted values $\hat{y}$ and actual values y. The L1 loss is defined as:

$$L1(y, \hat{y}) = \sum_{i=1}^{n} |y_i - \hat{y}_i| \tag{20}$$

It computes the sum of the absolute differences between predictions and true values, making it robust to outliers but slower to converge. The L2 loss, on the other hand, is given by:

$$L2(y, \hat{y}) = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{21}$$

This function amplifies larger errors by squaring the differences, making it more sensitive to outliers. Lastly, the Huber loss is a hybrid of L1 and L2:

$$L_\delta(y, \hat{y}) = \begin{cases} \frac{1}{2}(y - \hat{y})^2 & \text{if } |y - \hat{y}| \leq \delta \\ \delta(|y - \hat{y}| - \frac{1}{2}\delta) & \text{otherwise} \end{cases} \tag{22}$$

Huber behaves like L2 for small errors and like L1 for large errors, offering a balance between handling outliers and efficient optimization.

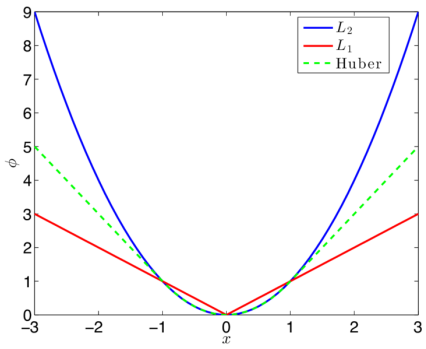


Figure 15: Difference between the three loss functions used

The results of our experiments using these loss functions over 30 epochs are shown in the figure 16. We observe that while the final loss values differ significantly between the functions, the final test accuracy remains nearly the same for all three.

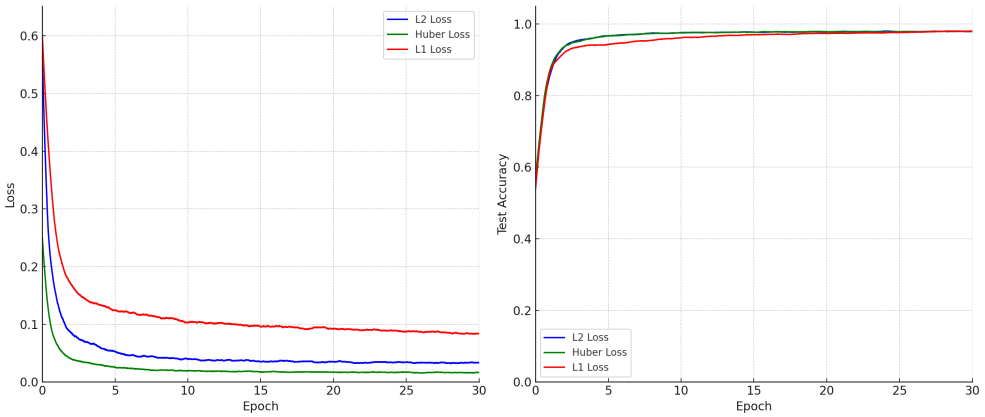


Figure 16: Evolution of the loss and the test accuracy over the epochs with the three different loss function

The difference in final loss values between the loss functions can be explained by the optimization properties of each function. For instance, the Huber loss, which combines aspects of both L1 and L2 losses, is particularly useful for handling outliers, making it less sensitive to quantum noise. This robustness allows for better minimization of the final loss.

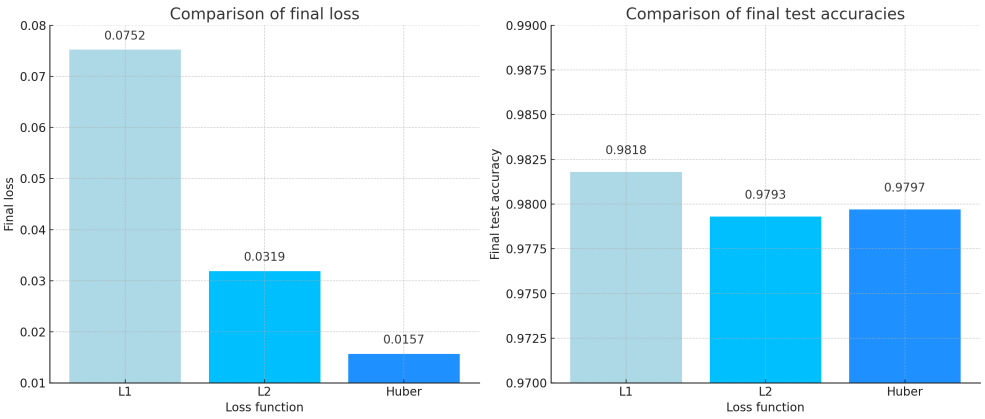


Figure 17: Comparison of the final loss and the final test accuracy with the three different loss function

While different loss functions may yield better minimization in terms of loss, they do not necessarily change the decision boundaries in quantum machine learning models [22]. This explains why we observe similar test accuracy across all the loss functions

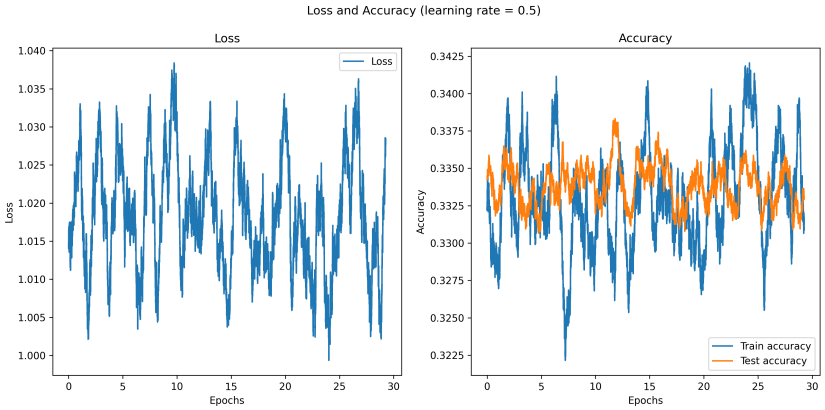
despite different loss values. The stability in test accuracy is also tied to the expressivity and generalization capabilities of quantum classifiers. The decision boundaries, particularly in data re-uploading quantum classifiers, remain stable across different loss functions, provided the quantum circuit architecture is the same, as the data's separability isn't significantly affected by the choice of loss function. This explains why, despite Huber loss achieving lower final loss values, the test accuracy remains close to that of L1 and L2 losses. The choice of loss function in quantum models often optimizes gradients but does not drastically alter the classification performance.

We do not use cross-entropy as the loss function here because our single-qubit QRU produces only one expectation value in $[-1, 1]$, rather than a probability vector over the three target classes. To apply cross-entropy directly, the quantum circuit would need to output at least three class probabilities, which in practice requires adding more qubits (e.g., two qubits already yield four basis states, enough to encode three classes). This, however, would break comparability with our one-qubit setting and alter the reference pipeline. Exploring multi-qubit QRUs with cross-entropy will be the subject of future work.

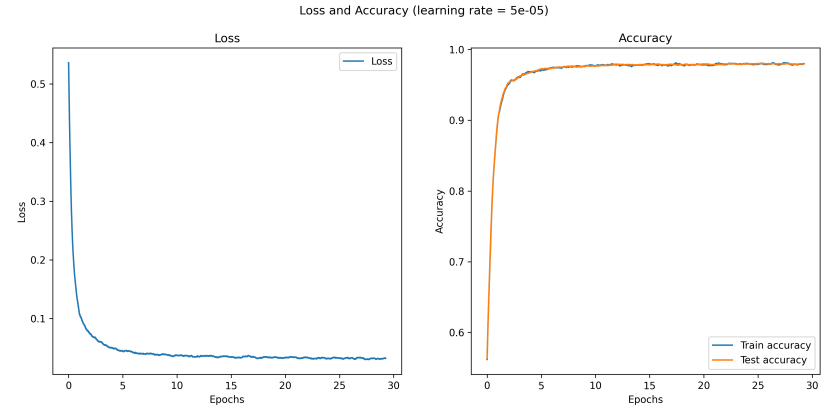
Next, we turn to the performance of the QRU when using different learning rates.

3.1.2.4 Learning Rate

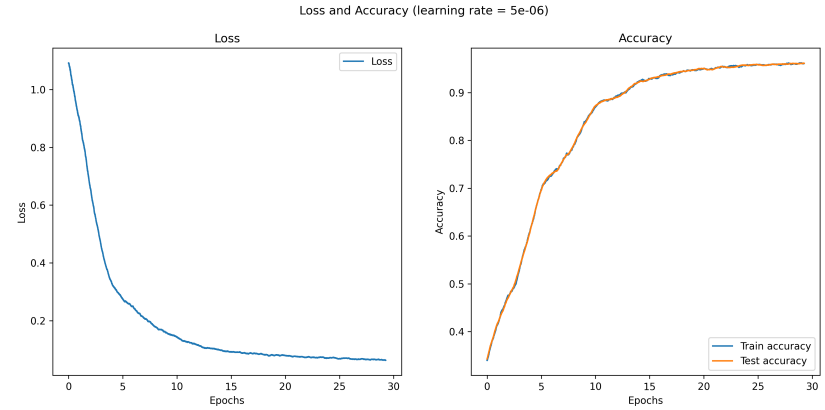
The goal of this section is to examine the impact of different initial learning rates on the model's convergence, final accuracy and loss. The results shown in Figure 18 highlight the impact of different learning rates on model performance.



(a) Learning rate = 0.5



(b) Learning rate = 5e-05

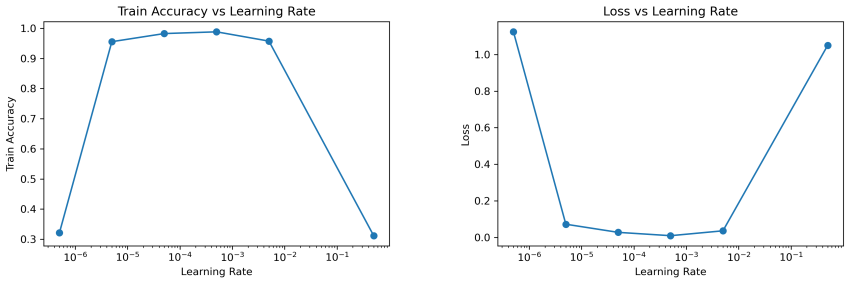


(c) Learning rate = 5e-06

Figure 18: Loss and accuracy for different learning rates: 0.5 (18a), 5e-05 (18b)and 5e-06 (18c).

For a learning rate of 0.5 using the Adam optimizer (Figure 18a), we observe significant oscillations in both loss and accuracy, indicating instability due to large parameter updates, with accuracy remaining around 0.33. In contrast, a learning rate of 5e-05 (Figure 18b) with Adam yields smooth convergence, reaching nearly 1.0 accuracy and showing a good balance between stability and learning speed. Finally, the smaller learning rate of 5e-06 (Figure 18c) also provides stable learning but slower convergence, with accuracy improving more gradually. These results suggest that 5e-05 is optimal for Adam, offering a good trade-off between speed and stability.

As shown in the graphs (Figures 19a and 19b), high learning rates such as 5×10^{-1} result in higher loss and instability in the model's performance. In contrast, moderate learning rates like 5×10^{-5} and 5×10^{-6} show better convergence, with lower final loss and higher accuracy.



(a) Accuracy evolution as a function of lr. (b) Loss evolution as a function of lr.

Figure 19: Comparison of accuracy (19a) and loss (19b) as a function of lr

Table **??** presents the final loss and accuracy metrics for different learning rates:

Learning rate	5×10^{-1}	5×10^{-3}	5×10^{-4}	5×10^{-5}	5×10^{-6}	5×10^{-7}
Loss	1.050	0.0358	**0.0092**	0.0274	0.0718	1.124
Train accuracy	0.312	0.958	**0.988**	0.983	0.956	0.322
Test accuracy	0.323	0.953	**0.985**	0.975	0.960	0.364

Table 3: Final loss, training and test accuracies for different learning rates. Best values are in bold.

The experiments demonstrate that moderate learning rates yield the best performance. With a learning rate of 5×10^{-4}, the model reaches a test accuracy of 98.5%, outperforming both higher and lower rates. The results also show that for very small initial learning rates, there is a degradation in performance. This could be due to the fact that the model has not fully converged within the 30 epochs, or it might be trapped in a local minimum. An ongoing study is looking at whether launching the

experiment with a small initial learning rate and different random seeds might yield a lower final loss compared to using higher initial learning rates.

In our experiments, we implemented an adaptive learning rate schedule to improve model convergence. The goal of this approach is to start with a relatively high learning rate, allowing the model to learn quickly at the beginning and then reduce the learning rate at specific milestones. This reduction helps the model fine-tune its parameters as it approaches convergence, leading to more stable results.

The learning rate schedule is as follows:

1. The initial learning rate is set to 0.005.
2. At the halfway point in the total number of epochs, the learning rate is divided by 10 to slow down updates and enable more precise learning.
3. At three-quarters of the way through the epochs, the learning rate is further divided by 10 to refine the convergence even more.

figure 20 illustrates how the learning rate decreases across epochs, using the described stepwise reduction scheme.

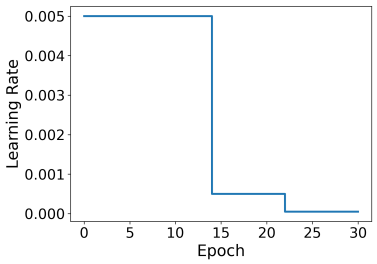


Figure 20: Adaptive learning rate schedule showing the gradual reduction in learning rate over training epochs.

We implemented a manual stepwise learning rate schedule, reducing the rate at mid- and late-training stages. However, this did not yield any improvement over the default adaptive learning rate of the Adam optimizer. This is expected, since Adam already adjusts the learning rate dynamically, whereas such manual scheduling would be more beneficial for non-adaptive optimizers like SGD.

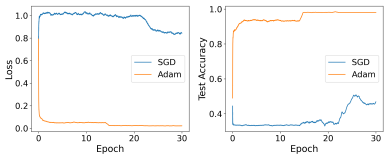


Figure 21: Evolution of the loss with adaptive learning rate

Despite using 100 epochs to ensure convergence between each learning rate change, the results were worse than those obtained with the Adam optimizer in automatic mode.

We now turn to the overall optimization process to evaluate its impact on convergence and accuracy.

4.2 Global optimization

In this subsection, we discuss the application of global optimization methods to the QRU model to enhance hyperparameter tuning. We focus on Bayesian optimization as a primary method and briefly discuss our initial tests with Hyperband. We also analyze the observed correlations between key hyperparameters and their combined effects on model performance.

4.2.1 Bayesian optimization

Bayesian optimization [23] is an efficient method for hyperparameter optimization when model evaluations are expensive. The objective function $f(x)$, which we seek to optimize, is modeled using a Gaussian process (GP). The GP is defined by a mean function $\mu(x)$ and a covariance function $k(x, x')$:

$$f(x) \sim \mathcal{GP}(\mu(x), k(x, x')) \tag{23}$$

The Upper Confidence Bound (UCB) acquisition function guides the selection of the next point to evaluate by maximizing the sum of the predicted mean $\mu(x)$ and an upper confidence bound proportional to the uncertainty $\sigma(x)$:

$$UCB(x) = \mu(x) + \kappa\sigma(x) \tag{24}$$

Where κ is a parameter balancing exploration (regions where $\sigma(x)$ is high) and exploitation (regions where $\mu(x)$ is high). Once a point x_t is evaluated, the GP is updated using the newly observed data. The updated mean $\mu_{t+1}(x)$ is calculated by conditioning the GP on the previous points $X_t = \{x_1, x_2, \ldots, x_t\}$ and their corresponding observations $y_t = \{f(x_1), f(x_2), \ldots, f(x_t)\}$:

$$\mu_{t+1}(x) = k(x, X_t)^\top K(X_t, X_t)^{-1} y_t \tag{25}$$

Here, $K(X_t, X_t)$ is the covariance matrix of the previously evaluated points and $k(x, X_t)$ is the covariance vector between the new point x and the previously evaluated points X_t. This equation leverages the correlations between x and past points to compute the new expected mean. The variance $\sigma^2_{t+1}(x)$ is updated similarly, representing the uncertainty of the GP's prediction at point x after conditioning on the previously observed data:

$$\sigma^2_{t+1}(x) = k(x, x) - k(x, X_t)^\top K(X_t, X_t)^{-1} k(x, X_t) \tag{26}$$

In this equation, $k(x, x)$ represents the variance at point x (before conditioning) and the second term adjusts this variance based on how correlated x is with the previously evaluated points. The result gives the updated uncertainty for x. This process

is repeated until convergence, efficiently finding the optimal hyperparameters.

In our work, we have integrated Bayesian optimization into the training process to search for the best hyperparameters for our model. We defined the hyperparameter search space as follows:

- **depth**: {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
- **learning rate**: {0.5, 0.05, 0.005, 0.0005, 0.00005, 0.000005, 0.0000005}
- **loss function**: {L1, L2, Huber}
- **optimizer**: {SGD, RMSprop, Adam, Adamax, NAdam, Adagrad, Adadelta, AdamW}

The objective function takes a set of hyperparameter values, trains the model with these values and returns the average loss. We used the `gp_minimize` function from `scikit-optimize` to execute Bayesian optimization and find the best hyperparameters: `gp_minimize(objective, space, n_calls=50, random_state=0, acq_func="UCB", kappa=4)`.

Figure 22 shows the obtained results: on the left, the evolution of the final loss, and on the right, the evolution of the final test accuracy as a function of different optimization runs.

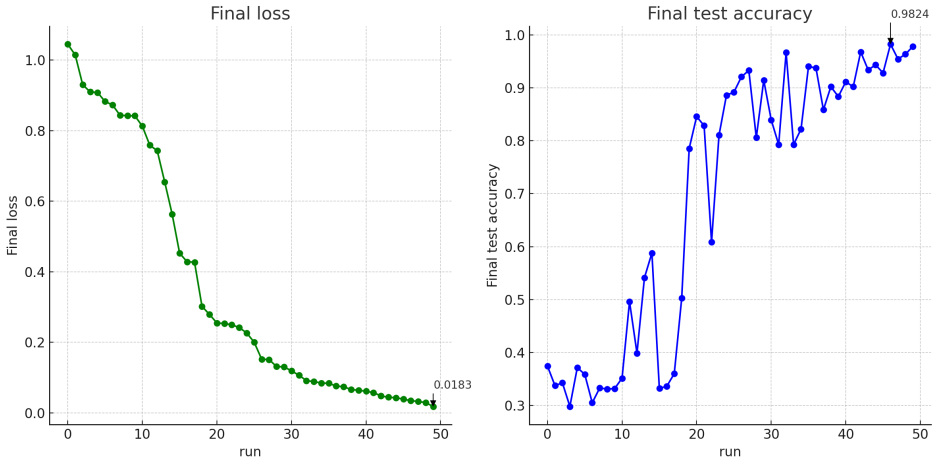


Figure 22: Evolution of the loss (left) and accuracy (right) during Bayesian optimization

The optimal configuration that resulted in the smallest final loss is: `depth=7, learning_rate=0.5, loss_function=Huber, optimizer=Adagrad` with a score of 0.018. Although this configuration minimized the loss, the final accuracy associated with this configuration remains slightly lower than that of other

configurations, reaching 0.978. In fact, the best accuracy achieved was 0.9824, obtained with the configuration `depth=5, learning_rate=0.5, loss_function=L2, optimizer=Adadelta`, but this configuration produced a slightly higher loss : 0.0344.

One notable aspect of the Bayesian optimization results is the relatively high learning rate of 0.5. Such a rate might seem too large in a classical optimization context. However, the use of the Adagrad and Adadelta optimizer compensates for this effect. Adagrad dynamically adjusts the learning rate based on the accumulated gradients during training. This mechanism allows the model to converge quickly at the beginning of training by exploring promising regions of the parameter space. This could be likely why Bayesian optimization identified a configuration with a high initial learning rate as optimal.

These results show that manual optimization can also lead to highly performant configurations, notably with an accuracy slightly higher than that obtained through Bayesian optimization, reaching 0.9854 with a learning rate of $5e^{-4}$. Manual optimization thus provides a better understanding of the individual impact of each hyperparameter, while global optimization allowed us to see which combinations of hyperparameter values were correlated.

In conclusion, although global Bayesian optimization effectively reduced the loss and quickly explored the hyperparameter space, manual hyperparameter optimization allowed for slightly better accuracy maximization. These results demonstrate the importance of understanding hyperparameter impact and adopting a combined approach, using both Bayesian optimization to efficiently explore hyperparameter combinations and manual optimization to fine-tune the most sensitive parameter settings.

4.2.2 Hyperband optimization

We also tested a global hyperparameter optimization method presented by Charles Moussa et al. for quantum neural networks [24]. HyperBand stands out for its adaptive approach, allocating a computational budget to several hyperparameter configurations and progressively eliminating the less promising ones. Optimization can be further improved by integrating techniques such as fANOVA, priors and surrogates.

To analyse the importance of hyperparameters and their interaction, we decomposed the variance of a model performance using an fANOVA model (Functional Analysis of Variance). It portions out this variance for each hyperparameter where the portion corresponds to their marginal effects, i.e., average effect of a given hyperparameter on performance. Priors on previous experiments or empirical knowledge help HyperBand to inform about hyperparameter selection better. While HyperBand with priors samples hyperparameters at random over the search space, it makes use of effective parameter values from other studies and emphasizes portions of the search space, which tends to lead to improved performing configurations. As an example, for a hyperparamter such as learning rate, rather than randomly selecting a value in a

certain range just limit the values to be sampled based on some distribution of what the best observed learning rates were from previous trials. Surrogate model is a fast and cheap model that predicts the cost of hyperparameter configuration by not having to retrain a full model in every trial, allowing for less computation. Surrogates in HyperBand are used to predict the performance of a new hyperparameter configuration given information from previous configurations.

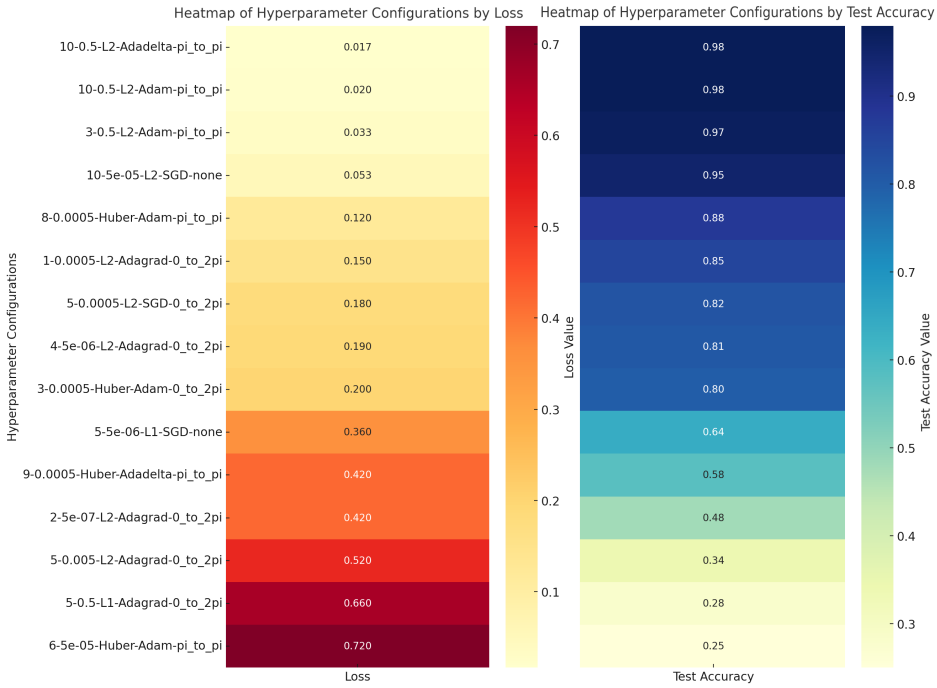


Figure 23: HyperBand optimisation results

HyperBand recovers trends consistent with our Bayesian search but under a different search space: unlike the Bayesian runs, HyperBand also tuned input normalization. Consequently, results are not strictly comparable across the two methods. In our QRU setting, HyperBand frequently selected large learning rates (e.g., 0.5) together with adaptive optimizers (Adadelta/Adagrad/Adam), and often favored deeper circuits (see Fig. 23). An fANOVA analysis suggests that, within the explored ranges, most of the performance variance is explained by the learning rate and the optimizer, whereas depth, loss function, and normalization contributed less—without implying they are negligible in general. Practically, HyperBand identified high-performing configurations faster than our Bayesian procedure.

4.3 Comparison of QRU, VQC, and MLP

We compare three models under matched parameter budgets and an identical training protocol: a single-qubit QRU, a three-qubit Variational Quantum Circuit (VQC), and a classical MLP ($3\to h\to 1$) whose trainable head is included in the parameter count. All models output a single scalar $y \in [-1, 1]$ trained with L2 loss against targets $\{-1, 0, +1\}$; classification is obtained by thresholding at $\{-0.33, +0.33\}$. We use Adam with a fixed learning rate 5×10^{-5} and 10 epochs in this run. Figure 24 reports per-epoch means of loss, train accuracy, and test accuracy.

QRU (1 qubit): Each re-uploading layer $i = 1..L$ applies, for each feature x_j ($j = 1..D$), the sequence $\mathrm{RX}(\theta^{(i)}_{j,0})$, $\mathrm{RY}(\theta^{(i)}_{j,1}\, x_j)$, $\mathrm{RX}(\theta^{(i)}_{j,2})$ on the single wire; readout is $\langle Z\rangle$. Trainable body parameters: $P_{\mathrm{QRU}} = 3DL$.

VQC (3 qubits): One-shot encoding $\mathrm{RY}(x_0)$ on q_0, $\mathrm{RY}(x_1)$ on q_1, $\mathrm{RY}(x_2)$ on q_2; then L variational layers of local trainable rotations $\mathrm{RX}(\alpha_{i,q})$, $\mathrm{RY}(\beta_{i,q})$, $\mathrm{RZ}(\gamma_{i,q})$ on $q \in \{0, 1, 2\}$, followed by ring entanglement ($\mathrm{CNOT}_{0\to1}$, $\mathrm{CNOT}_{1\to2}$, $\mathrm{CNOT}_{2\to0}$). We aggregate onto q_0 via $\mathrm{CNOT}_{1\to0}$ and $\mathrm{CNOT}_{2\to0}$ and measure $\langle Z_0\rangle$. Trainable body parameters: $P_{\mathrm{VQC}} = 3QL$ with Q=3.

MLP ($3\to h\to 1$): Two fully-connected layers with ReLU and a tanh output, `fc1` : $\mathbb{R}^3 \to \mathbb{R}^h$, `fc2` : $\mathbb{R}^h \to \mathbb{R}$, then tanh. We include the head: $P_{\mathrm{MLP}} = (3{+}1)h + (h{+}1)\cdot 1 = 5h{+}1$.

To match the body capacity of the two quantum models we set a common depth L. With D=3 and Q=3, this yields $P_{\mathrm{QRU}}{=}3DL{=}9L$ and $P_{\mathrm{VQC}}{=}3QL{=}9L$. We choose h such that $5h{+}1 \approx P_{\mathrm{QRU}}$. Table 4 summarizes the counts used in our run (L=10).

Model	Formula	Hyperparams	P	Output
QRU (1q)	$3DL$	D=3, L=10	$3\times3\times10 = 90$	$\langle Z\rangle$
VQC (3q)	$3QL$	Q=3, L=10	$3\times3\times10 = 90$	$\langle Z_0\rangle$
MLP ($3\to h\to 1$)	$5h{+}1$	$h{=}\lfloor(90{-}1)/5\rfloor{=}17$	$5\times17{+}1 = 86$	tanh

Table 4: Parameter-matched setup (D=3, Q=3, L=10). The MLP includes its head in P_{MLP}.

Under this protocol (LR = 5×10^{-5}, 10 epochs), we obtain results shown with Table 5:

A coarse trainability proxy (area under the loss curve across epochs) ranks MLP $<$ QRU $\lesssim$ VQC, i.e., the MLP optimizes fastest (lowest loss area), while the QRU achieves the best generalization in this run.

QRU attains the highest test accuracy (0.987), followed by MLP (0.970). VQC lags (0.927) and shows a negative generalization gap (test $<$ train, -0.029), unlike QRU and MLP where test is slightly higher than train.

Model	Loss	Train Acc	Test Acc
QRU (1q)	**0.038**	**0.977**	**0.987**
VQC (3q)	0.084	0.956	0.927
MLP (h=17)	0.044	0.948	0.970

Table 5: Final per-epoch means under matched parameter budgets.

MLP reaches the lowest loss and the smallest area-under-loss, consistent with faster optimization on this scalar-regression-with-thresholds objective, however, low loss does not translate into the best test accuracy here.

With matched body budgets (P=90) and identical logging, re-uploading on a single qubit (QRU) proves highly competitive, slightly outperforming the 3-qubit VQC with ring entanglement under L2+threshold training.

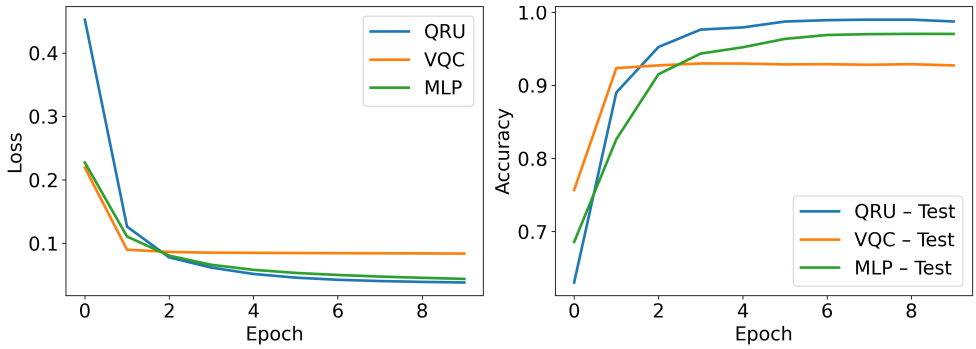


Figure 24: Learning curves under matched parameter budgets. Left: loss vs. epoch. Right: accuracy vs. epoch (solid = train, dashed = test.

5 Conclusion

Our study demonstrates that the QRU model is well suited for classification tasks in the context of current NISQ devices, due to its minimal ressource requirement and strong classification performances. By systematically exploring and analyzing both model-specific hyperparameters (such as circuit depth, rotation gates and input normalization) and training hyperparameters (including learning rate, batch size, optimizer choice and loss functions), we identified optimal configurations that balance accuracy and computational efficiency, thus making the QRU highly suitable for real-world quantum machine learning applications.

Through the application of global optimization methods, we uncovered significant correlations and interactions among hyperparameters. Moreover, our mathematical analysis established the QRU's superior expressivity compared to the widely-used VQC. Indeed, we highlighted that VQC's limitations are not only in terms of qubit

counts and entanglement gate noise but also in the expressivity boundaries in Fourier frequency domains. These insights justify the QRU as the preferred choice for quantum machine learning tasks.

This work thus provides a solid framework for optimizing QRU performance within the constraints of current quantum hardware, representing a significant advancement toward their practical application in diverse scientific fields such as including high-energy physics.

Statements and Declarations

All code and experimental data supporting the results of this paper are publicly available at: https://github.com/LeaCasse/QRU_Calorimetry_Optimization

References

[1] Biamonte, J., Wittek, P., Pancotti, N., Rebentrost, P., Wiebe, N., Lloyd, S.: Quantum machine learning. Nature **549**(7671), 195–202 (2017) https://doi.org/10.1038/nature23474

[2] Hitran, Ellynn: Simulate complex physics with graph networks. Accessed: 2024-09-23 (2023)

[3] Team, G.: Graph Network Simulator (GNS). Accessed: 2024-09-23 (2023). https://github.com/geoelements/gns

[4] Zhou, Z.H., Liu, S.: Machine learning (2021)

[5] Steane, A.: Quantum computing. Reports on Progress in Physics **61**(2), 117 (1998)

[6] Preskill, J.: Quantum computing in the nisq era and beyond. Quantum **2**, 79 (2018) https://doi.org/10.22331/q-2018-08-06-79

[7] Bharti, K., Cervera-Lierta, A., Kyaw, T.H., Haug, T., Alperin-Lea, S., Anand, A., Degroote, M., Heimonen, H., Kottmann, J.S., Menke, T., *et al.*: Noisy intermediate-scale quantum algorithms. Reviews of Modern Physics **94**(1), 015004 (2022)

[8] Pérez-Salinas, A., Cervera-Lierta, A., Gil-Fuster, E., Latorre, J.I.: Data re-uploading for a universal quantum classifier. Quantum **3**, 139 (2019)

[9] Ezhov, A.A., Ventura, D.: Quantum neural networks. Future Directions for Intelligent Systems and Information Sciences: The Future of Speech and Image Technologies, Brain Computers, WWW, and Bioinformatics, 213–235 (2000)

[10] Schuld, M.: Supervised quantum machine learning models are kernel methods. arXiv preprint arXiv:2101.11020 (2021)

[11] Glendinning, I.: The bloch sphere. In: QIA Meeting, pp. 3–18 (2005)

[12] Team, P.O.: HGCal-like simulations. Accessed: 2024-09-23 (2021). https://llrogcid.in2p3.fr/hgcal-like-simulations/

[13] Kingma, D.P.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)

[14] Cerezo, M., Arrasmith, A., Babbush, R., Benjamin, S.C., Endo, S., Fujii, K., McClean, J.R., Mitarai, K., Yuan, X., Cincio, L., *et al.*: Variational quantum algorithms. Nature Reviews Physics **3**(9), 625–644 (2021)

[15] Becheva, E., Beaudette, F., Sauvan, J.-B., Melennec, M., Ghosh, S., Mellin, M., Magniette, F.: High granularity calorimetry D2 dataset. https://zenodo.org/records/14260279 (2024). https://doi.org/10.5281/zenodo.14260279

[16] Collaboration, C.: The phase-2 upgrade of the cms endcap calorimeter. Technical report, CERN, Geneva (2017). https://doi.org/10.17181/CERN.IV8M.1JY2 . https://cds.cern.ch/record/2293646

[17] Briscoe, E., Feldman, J.: Conceptual complexity and the bias/variance tradeoff. Cognition **118**(1), 2–16 (2011)

[18] Barthe, A., Pérez-Salinas, A.: arxiv: Gradients and frequency profiles of quantum re-uploading models. Technical report (2023)

[19] Hubregtsen, T., Pichlmeier, J., Stecher, P., Bertels, K.: Evaluation of parameterized quantum circuits: on the relation between classification accuracy, expressibility, and entangling capability. Quantum Machine Intelligence **3**, 1–19 (2021)

[20] Tüysüz, C., Clemente, G., Crippa, A., Hartung, T., Kühn, S., Jansen, K.: Classical splitting of parametrized quantum circuits (2022). arXiv preprint arXiv:2206.09641

[21] McClean, J.R., Boixo, S., Smelyanskiy, V.N., Babbush, R., Neven, H.: Barren plateaus in quantum neural network training landscapes. Nature communications **9**(1), 4812 (2018)

[22] Aktar, S., Bärtschi, A., Oyen, D., Eidenbenz, S., Badawy, A.-H.A.: Graph neural networks for parameterized quantum circuits expressibility estimation. arXiv preprint arXiv:2405.08100 (2024)

[23] Alaa, A.M., Schaar, M.: Hyperparameter optimization for machine learning models based on bayesian optimization. ResearchGate (2019)

[24] Moussa, C., Rijn, J.N., Bäck, T., Dunjko, V.: Hyperparameter importance of quantum neural networks across small datasets, 32–46 (2022) https://doi.org/10.

1007/978-3-031-18840-4_3