A Quantum Approach for the Classification of Contaminant Ions in Water

Rudraksh Sharma

November 16, 2025

Abstract

This report details a Quantum Machine Learning (QML) solution for a 4-class classification problem: identifying contaminant ions (C_0, C_1, C_2, C_3) in water from photonic sensor data. We developed a hybrid quantum-classical neural network (HQ-NN) that integrates a classical feed-forward encoder, a variational quantum circuit, and a classical output head. To enhance robustness, we trained an ensemble of three model variants (using 6, 7, and 8 qubits) with a 3-fold stratified cross-validation scheme. The final solution employs a stacked ensemble, achieving a robust out-of-fold accuracy of **67.76%**. This approach demonstrates a viable QML pipeline for complex classification tasks on real-world experimental data.

1 Problem Understanding

The core task is to classify four distinct types of contaminant ions (C_0, C_1, C_2, C_3) dissolved in an aqueous solution. The classification must be performed using data from a photonic sensor, which provides three continuous physical parameters for each sample:

- **Wavelength (λ):** The wavelength of light used to probe the sensor.
- **Propagation Constant (β):** The effective wavenumber, reflecting how light propagates in the sensing region.
- **Electric Field Fraction:** The fraction of electric field energy interacting with the water sample, indicating sensor sensitivity.

The challenge lies in the nature of this data, which is non-trivial and may possess complex, non-linear correlations. This makes it a suitable candidate for Quantum Machine Learning, where high-dimensional Hilbert spaces can be leveraged to find effective decision boundaries. The goal is to build a robust model that generalizes well to unseen data, addressing the high-impact problem of water contamination detection.

2 Model Architecture

We designed a Hybrid Quantum-Classical Neural Network (HQ-NN) using the `PyTorch` and `Pennylane` libraries. The architecture is modular, consisting of three primary components:

1. **Classical Encoder:** A classical neural network that functions as a trainable feature extractor. It takes the 3 preprocessed input features and maps them to a latent space vector. This encoder is composed of:

```
Linear(in=3, out=H) -> ReLU -> Dropout(0.12) -> Linear(out=N_q)
```

Here, H is a hidden dimension (48 or 64) and N_q is the number of qubits, which is also the dimension of the latent space.

2. **Variational Quantum Circuit (VQC):** The core quantum component, implemented as a `qml.qnn.TorchLayer`. This VQC:
   - **Encodes** the N_q-dimensional latent vector from the encoder into the quantum state (see Section 3).
   - **Processes** the state using a trainable ansatz, `qml.StronglyEntanglingLayers`. This ansatz applies sequences of single-qubit rotations and entangling CNOT gates.
   - **Measures** the expectation value of the Pauli-Z operator, $\langle \sigma_Z \rangle$, for each of the N_q qubits, returning a classical vector of N_q expectation values.
3. **Classical Head:** A final classical neural network that receives the N_q measurement results from the VQC and performs the final 4-class classification:

   ```
   Linear(in=N_q, out=24) -> ReLU -> Dropout(0.12) -> Linear(out=4)
   ```

   This block outputs raw logits for the four ion classes.

To find an optimal configuration, we trained three variants of this architecture, systematically increasing the quantum resources:

- **v6_2:** $N_q = 6$ qubits, 2 layers of the `StronglyEntanglingLayers` ansatz.
- **v7_3:** $N_q = 7$ qubits, 3 layers of the ansatz.
- **v8_3:** $N_q = 8$ qubits, 3 layers of the ansatz.

3 Classical-to-Quantum Encoding Method

A naive, fixed encoding of 3 features onto a 6+ qubit system is suboptimal. We employed a **learnable data-driven encoding** strategy.

The 3 input features are first passed through the **Classical Encoder** (described in Sec. 2) to produce a latent vector $\mathbf{v} \in \mathbb{R}^{N_q}$. This step effectively "learns" the best way to represent the 3-dimensional input in the N_q-dimensional latent space.

This latent vector $\mathbf{v}$ is then transformed into rotation parameters $\boldsymbol{\theta}$ for the quantum circuit:

$$\boldsymbol{\theta} = \pi \cdot \tanh(\mathbf{v})$$

This use of the tanh function is a critical step to bound the angles to the range $(-\pi, \pi)$, preventing parameter saturation and ensuring stable gradients.

Finally, this parameter vector $\boldsymbol{\theta}$ is loaded into the quantum circuit using `qml.AngleEmbedding`. This method embeds the N_q parameters as R_Y rotation angles, one for each qubit. The rationale is to combine a powerful classical pre-processing network with a flexible quantum embedding, allowing the model to discover the most expressive feature representation for the quantum circuit.

4 Training Process and Optimization Details

The model was trained on a total of 48,000 samples (12,000 per class) using a 3-fold stratified cross-validation scheme. This ensures that each fold's validation set is representative of the overall class distribution and provides robust out-of-fold (OOF) predictions for ensembling.

4.1 Two-Stage Training

To stabilize the training of the complex hybrid model and avoid vanishing gradients (barren plateaus), we adopted a two-stage training strategy:

1. **Encoder Pretraining:** The classical encoder was first trained "classically" (with a simple linear head attached) for 8 epochs. This stage used a standard `CrossEntropyLoss` and an `Adam` optimizer (LR = 1e-3) to initialize the encoder weights to a sensible region of the parameter space.

2. **Full Hybrid Training:** The pretrained encoder weights were loaded into the full HQ-NN. The entire model (encoder, VQC, and head) was then trained for up to 20 epochs using an `Adam` optimizer with a lower learning rate (LR = 5e-4).

4.2 Loss Function and Optimization

For the main hybrid training, we replaced the standard cross-entropy loss with **Focal Loss** ($\gamma = 1.5$). This loss function was chosen to address class imbalance or difficulty by down-weighting the loss from easy-to-classify samples, forcing the model to focus on harder examples.

Furthermore, we applied manual class weights to the loss function:

$$\text{weights} = [1.0, 1.0, 1.0, 1.2]$$

This gives a 20% higher penalty for misclassifying ion C_3, which was identified as a particularly challenging class during initial tests.

To prevent overfitting, we employed an `EarlyStopping` patience of 6 epochs and a `ReduceLROnPlateau` learning rate scheduler, which monitored the validation accuracy.

4.3 Ensemble Strategy

After training all three variants (v6_2, v7_3, v8_3) across all 3 folds, we combined their OOF predictions.

- **Soft-Average:** A simple average of the 3 models' OOF probability distributions.
- **Stacking:** A `LogisticRegression` meta-model was trained on the concatenated OOF probabilities from all 3 models (creating a 12×1 feature vector per sample).

The stacked ensemble produced the best results.

5 Key Results

The OOF accuracy for the individual models and the two ensemble methods are as follows:

The stacked ensemble, which leverages a `LogisticRegression` model to find the optimal weights for combining the outputs of the three quantum models, was selected as the final model due to its superior performance.

6 Performance Metrics (Stacked Ensemble)

The final stacked model's performance is detailed below. The overall accuracy is **67.76%**.

6.1 Classification Report

The per-class metrics reveal a significant performance disparity between classes.

Model / Method	Out-of-Fold (OOF) Accuracy
Variant v6_2 (6Q, 2L)	64.88%
Variant v7_3 (7Q, 3L)	66.49%
Variant v8_3 (8Q, 3L)	65.95%
Soft-Average Ensemble	66.44%
Stacked Ensemble (Logistic)	**67.76%**

Table 1: Overall accuracy for individual model variants and ensemble methods. The stacked ensemble, which learns to weigh the predictions from the three variants, achieved the highest performance.

Class	Precision	Recall	F1-Score	Support
Ion C_0	0.5967	0.4274	0.4981	12000
Ion C_1	0.6039	0.9182	0.7286	12000
Ion C_2	0.9770	0.9823	0.9797	12000
Ion C_3	0.5048	0.3826	0.4353	12000
Weighted Avg	**0.6706**	**0.6776**	**0.6604**	**48000**

Table 2: Classification report for the final stacked ensemble model (OOF predictions).

6.2 Confusion Matrix

The confusion matrix provides a clear view of the model's strengths and weaknesses.

	Predicted C_0	**Predicted C_1**	**Predicted C_2**	**Predicted C_3**
True C_0	**5129**	3126	67	3678
True C_1	0	**11018**	157	825
True C_2	0	212	**11788**	0
True C_3	3466	3890	53	**4591**

Table 3: Confusion matrix for the final stacked ensemble model (OOF predictions).

6.3 Interpretation

The metrics clearly show that:

- The model is **exceptionally proficient** at identifying Ion C_2, with an F1-score of ∼98%.
- The model is also **strong** at identifying Ion C_1 (F1-score of ∼73%), primarily due to its very high recall (92%).
- The model's primary weakness is its **inability to reliably distinguish between C_0 and C_3**. A large number of true C_0 samples are misclassified as C_3 (and vice-versa), and both are frequently misclassified as C_1. This suggests that ions C_0 and C_3 may produce highly similar optical responses in the sensor.

7 Limitations and Possible Improvements

While this HQ-NN ensemble provides a robust baseline, several limitations and avenues for improvement exist.

7.1 Limitations

1. **Class Confusion:** The most significant limitation is the poor separability of classes C_0 and C_3. This may be a fundamental limit of the input features, suggesting the sensor data itself makes these ions difficult to distinguish.
2. **Ansatz Expressivity vs. Trainability:** We used a generic `StronglyEntanglingLayers` ansatz. While highly expressive, such ansatze can be difficult to train and are susceptible to barren plateaus, especially as qubit counts and layers increase. Our pretraining strategy was designed to mitigate this, but it remains a concern.
3. **Training Time:** Training VQCs is computationally intensive. The 3-fold CV on 3 model variants, including pretraining, represents a significant computational cost.

7.2 Possible Improvements

1. **Quantum Kernel Methods:** For a low-dimensional input (3 features), a Quantum Kernel (QK) approach could be highly effective. One could use the VQC architecture as a trainable kernel and pair it with a classical Support Vector Machine (SVM). This might yield better decision boundaries than the neural network head.
2. **Data Re-uploading:** Instead of a single encoding layer, the 3 input features could be re-uploaded multiple times between layers of the ansatz. This can significantly increase the model's expressivity without increasing the qubit count.
3. **Advanced Feature Engineering:** To address the C_0/C_3 confusion, new classical features could be engineered from the original three (e.g., polynomial terms, ratios) and fed into the classical encoder to help it find a more separable latent representation.
4. **Ansatz Design:** Future work could explore more problem-specific or hardware-efficient ansatze that may be easier to train and offer better performance for this specific data structure.