



THE UNIVERSITY *of* EDINBURGH
School of Physics
and Astronomy

Senior Honours Project

Quantum Machine Learning as a Higgs Classifier

Jordi Junyao Zhang Yu
March 2025

Abstract

In this report, we present an investigation of Quantum Machine Learning methods in the context of Higgs classifiers. Specifically, we look to classify di-photon events with $H \rightarrow \gamma\gamma$ as the signal process. We have trained a classical neural network and showed a common problem with machine learning as a whole in this context, namely features correlated with the invariant mass of the 2 photons. New features were engineered and subsequently used to train a classical neural network as a benchmark and a variational quantum classifier to represent QML methods. Our results do not agree with previous works done in similar contexts, however, we note that our analysis uses a simplified dataset not fully representative of those used at the LHC. QML is overall a promising field to apply on HEP analysis, however, there are still some serious shortcomings that still require a lot of work before QML can be consistently applied in this context.

Declaration

I declare that this project and report is my own work.

Supervisor: Dr. Liza Mijovic

10 Weeks

Contents

1	Introduction	2
1.1	Neural Networks	3
1.2	Quantum Computing Basics	4
1.3	Variational Quantum Classifiers	5
2	Monte-Carlo Data	5
3	Exploratory Model	6
3.1	Problems with Model 1	6
4	Feature Engineering	8
5	Classical NN Approach	9
5.1	Feature Ranking	9
5.2	Results and Discussion	10
6	Quantum VQC Approach	11
6.1	Discussion	12
7	Limitations of Quantum Models	13
7.1	Debugging and Software	14
7.2	Balancing Accuracy and Training Time	14
7.3	Circuit Complexity and Decoherence	14
8	Conclusion	15
A	Commonly Used Quantum Gates	18
B	Model 1 and 2	19
C	Rest of Feature Ranking	20
D	Model 3, 4 and 5	20
E	VQC Circuit and Code	22

1 Introduction

Since the discovery of the Higgs boson at the Large Hadron Collider (LHC), experiments to measure its properties have become increasingly of interest. Particle beams are accelerated to relativistic speeds and collided together to produce heavier particles, one of which is the Higgs boson. These particles have very short lifetimes and almost immediately decay into other particles, usually a mix of leptons, photons and hadron jets. Due to the short lifetimes, these heavy particles are almost impossible to detect directly, so instead we detect the decay products. These detected decay products are referred to as a final state.

In experiments, we label final states coming from a Higgs decay as signal and those from other decays as background. A typical experiment might have results similar to Fig. 1. The smoothly falling curve corresponds to background decays and the small bump to signal ones. The final states of signal and background events are both composed of a mix of leptons, photons and hadron jets, making it hard to distinguish between them, and with such a scarcity of signal events, classifying accuracy becomes even more important. Furthermore, the high complexity of the final state means there is a large number of variables to consider. Traditional cut-based methods aren't appropriate in situations with such a high dimensionality and dominant background, thus we turn to machine learning methods and its generalization properties of being able to accurately classify unseen data.

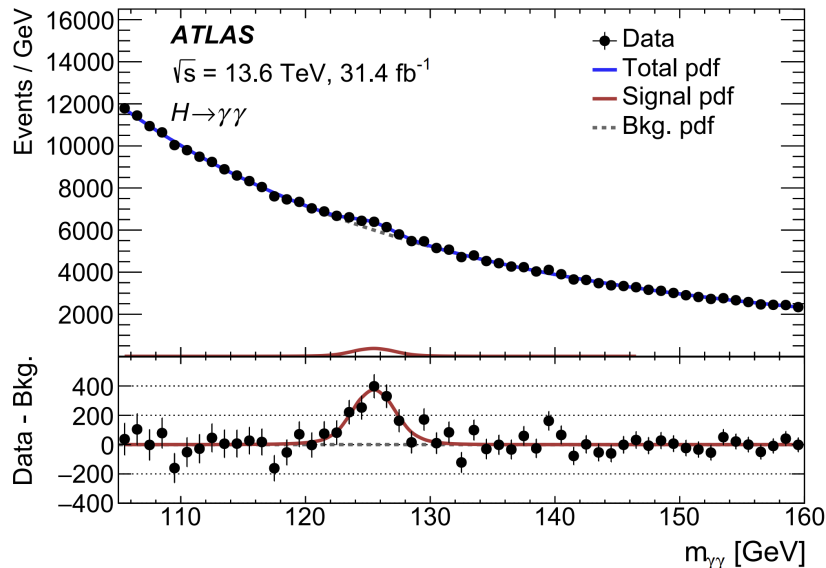


Figure 1: Di-photon invariant mass spectrum as observed in data. The signal, background and total probability density functions (pdf) are derived from the fit to data as detailed in [1]. The bottom curve is calculated by subtraction of the background from the total pdf. Source: [1]

Current HEP analyses are typically done using methods like Boosted Decision Trees and Neural Networks for the classification of data [2]. Recent works with Quantum Machine Learning (QML) methods have shown that these can perform similarly or better than their classical counterparts and serves as a proof of concept [3, 4]. However, they fail

to address the advantages and disadvantages QML methods may have. In this project, we will be looking to discriminate di-photon events with $H \rightarrow \gamma\gamma$ as our signal process, first using a classical ML method as a benchmark and after with a QML one. Ultimately, our goal is to expand on the topic and investigate the shortcomings of QML.

To do so, we chose to use a Neural Network (NN) as our classical model given its large popularity and ease of use. For similar reasons, we chose a Variational Quantum Classifier (VQC) as a representative of quantum methods.

1.1 Neural Networks

Neural networks are machine learning models designed to mimic the functioning and learning capabilities of the human brain. In our case, we will use them as classifiers. They are composed of units that process data called neurons that are organized into layers as shown in Fig. 2. Each neuron in a layer receives multiple inputs and generates an output z according to:

$$z = \sum_i (w_i x_i) + b$$

where w_i , x_i are the weights and inputs respectively, and b is the bias. These outputs are then passed through a user chosen activation function which introduces non-linearity into the system and is what allows the network to learn patterns. Some common choices include ReLU, sigmoid and tanh. This process of feeding data and generating outputs is called a forward pass.

Using these outputs, we can compute the loss function, which is a measure of performance for the model. Then we compute the gradients of the loss function with respect to the weights and biases. These are essentially the error in the weights and biases, and using an optimization algorithm we can update the weights and biases so as to minimize the loss. This process to update the trainable parameters is referred to as a backward pass. A full run of forward and backward passes constitutes an iteration. Usually in training, we pass a subset of the training data in each iteration. An epoch then consists of as many iterations as it takes to pass the entire dataset, such that $1 \text{ Epoch} = \frac{\text{Training Samples}}{\text{Batch Size}} \text{ Iterations}$ rounded upwards.

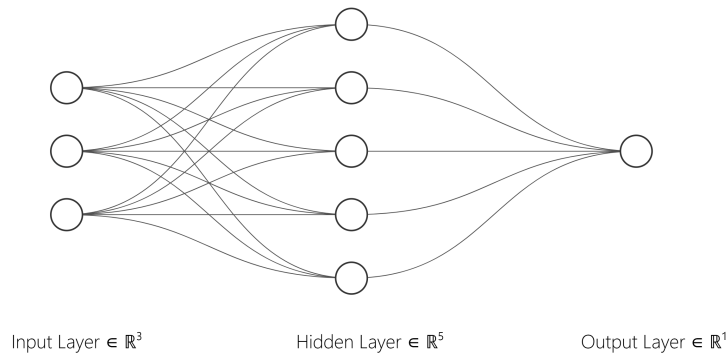


Figure 2: A 2-layer neural network with fully connected dense layers. Input layer receives raw data and passes it to the hidden layer where it is processed using trainable weights and biases. After data is processed, it comes out through the output layer.

1.2 Quantum Computing Basics

Before going into VQCs, we need to first explain some quantum computing basics to properly understand them. Much of the content in this section is taken from Ref. [5] Unlike classical computers, quantum computers utilize qubits instead of bits. Qubits are a 2-state quantum system whose state is a 2-component complex vector: $|\psi\rangle \in \mathbb{C}^2$. We can label its eigenstates as $|0\rangle$ and $|1\rangle$ which forms a "computational basis".¹ Now any state of the qubit may be written as a superposition of the computational basis: $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, where α and β are arbitrary complex numbers that satisfy $|\alpha|^2 + |\beta|^2 = 1$ for normalisation.

Multiple qubits compose a quantum register whose state is a vector in the tensor product of the qubits' vector spaces. So for 3 qubits we have: $|\psi\rangle \in \{\mathbb{C}^2 \otimes \mathbb{C}^2 \otimes \mathbb{C}^2\} = \mathbb{C}^8$. To obtain a computational basis for a register, we take the tensor product of single qubit basis states as so:

$$\begin{aligned} |0\rangle_2 \otimes |0\rangle_1 \otimes |0\rangle_0 &= |000\rangle = |0\rangle \\ |0\rangle_2 \otimes |0\rangle_1 \otimes |1\rangle_0 &= |001\rangle = |1\rangle \\ &\vdots \\ |1\rangle_2 \otimes |1\rangle_1 \otimes |1\rangle_0 &= |111\rangle = |7\rangle \end{aligned}$$

It is now easy to label these new states with binary as our new computational basis. Thus a general state is a linear combination of these computational basis:

$$|\psi\rangle = \sum_j \alpha_j |j\rangle, \quad \text{where } \alpha_j \in \mathbb{C} \text{ and } j = 0, 1, \dots, 2^{N_{\text{qubits}}} - 1$$

This treatment can be generalized to any number of qubits.

In classical computers, we use logical gates to do operations. Similarly, quantum computers utilize quantum gates acting on the register to do so. Quantum gates are devices that apply a fixed unitary operator on a selected qubit(s) and can be represented as matrices. See Appendix A for some commonly used gates. We can now build circuits by applying gates sequentially (left to right) to a register of qubits. Fig. 3 shows an example circuit.

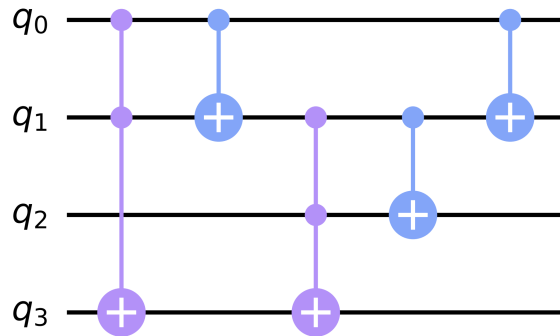


Figure 3: Quantum adder circuit using Toffoli and CX gates. Source: [6].

¹Quantum equivalent of 0s and 1s in binary.

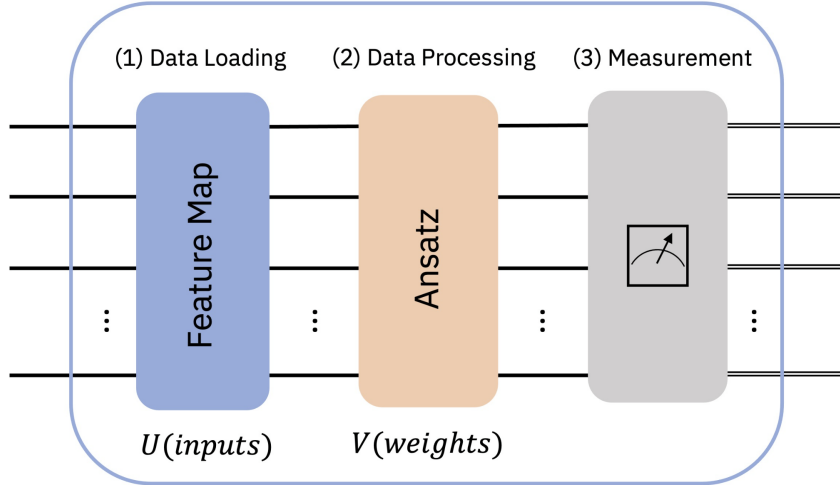


Figure 4: Generic quantum circuit for a variational quantum classifier. Source: [7]

1.3 Variational Quantum Classifiers

In a Variational Quantum Classifier (VQC), we construct a circuit in layers similar to the neural network[4]. It is composed of 3 steps, see Fig. 4:

- **Feature Map:** Quantum circuit responsible for encoding classical input data into a quantum state, similar to the input layer in a neural network. Unlike a neural network though, the feature map can be repeated to further encode data as quantum states.
- **Variational Ansatz:** This is a quantum circuit with tunable parameters that transforms quantum states and introduces non-linearity in the form of entanglement. This is analogous to the hidden layers of a neural network. The ansatz circuit can be repeated, which is similar to adding more hidden layers.
- **Measurement:** After applying a feature map and the ansatz, we measure the quantum state and assign it a label/class based on the probability distribution. Output layer equivalent.

We can think of the above 3 steps as a forward pass. Then for the backward pass, we can compute a loss function for the measurements and use it along with a classical optimizer. The optimizer will minimize the loss and adjust the ansatz parameters accordingly, thus enabling the circuit to learn.

2 Monte-Carlo Data

In this project, we utilize data with di-photon events from the signal $H \rightarrow \gamma\gamma$ events and background events with no Higgs boson. The signal processes were generated using Powheg Box v2 [8, 9, 10] and interfaced with Pythia 8.2 [11]. Background simulated samples were generated using MadGraph5_aMC@NLO [12] and interfaced with Pythia 8.2. ATLAS detector response for both signal and background events was simulated using Geant4 [13]. The simulated data was produced by the ATLAS experiment and provided by the project supervisor Liza.

3 Exploratory Model

Before further discussion, we want to illustrate a common problem with any type of machine learning algorithm in this sort of application. To do this, we first investigated a simple dataset consisting of p_T , η , ϕ and E of the 2 leading photons only, with a total of 8 features and 64085 samples. Out of our samples, we had 28089 background events and 35996 signal ones. We have a much larger proportion of signal to background events than a real experiment, however that is by design. It is impossible to train a high performance ML model with a huge imbalance of classes, hence we utilize simulated data that is more balanced class-wise.

Looking at Fig. 5, we observe that most signal events lie between 120000 and 130000 MeV with a small fraction of background events. Higgs boson mass is approximately 125 GeV and most signal events lie close to it, so this matches our expectations. Based on this we can make a rough estimate for expected performance. Just taking the ratio of signal to background events within the region we might expect an accuracy close to 90%.

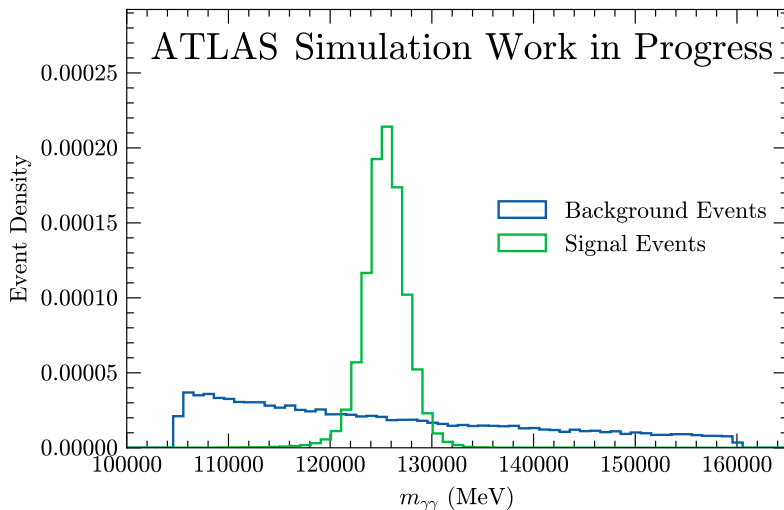


Figure 5: Histogram of $m_{\gamma\gamma}$ for signal vs background events

The p_T 's and E 's are much larger than the η 's and ϕ 's of the 2 photons, so to avoid any feature dominating over the rest, each feature was min-max scaled. Data was split into 80% training, 10% validation and 10% testing. Using this, we trained a neural network with fully connected layers, using the ADAM optimizer with a learning rate of 0.001 and a batch size of 128. We refer to this as Model 1, and it resulted in an accuracy of 81%. The model performs similar to the earlier estimate of performance and has overall a good accuracy. However, it also has issues which are not immediately obvious.

3.1 Problems with Model 1

Fig. 6a shows the classification predictions of the model using only background events as input. In the process of training the model, we have lost information of the smoothly falling background shape. This essentially makes this model useless, as even in the absence of signal events, it will predict a signal-like peak centred around the Higgs mass.

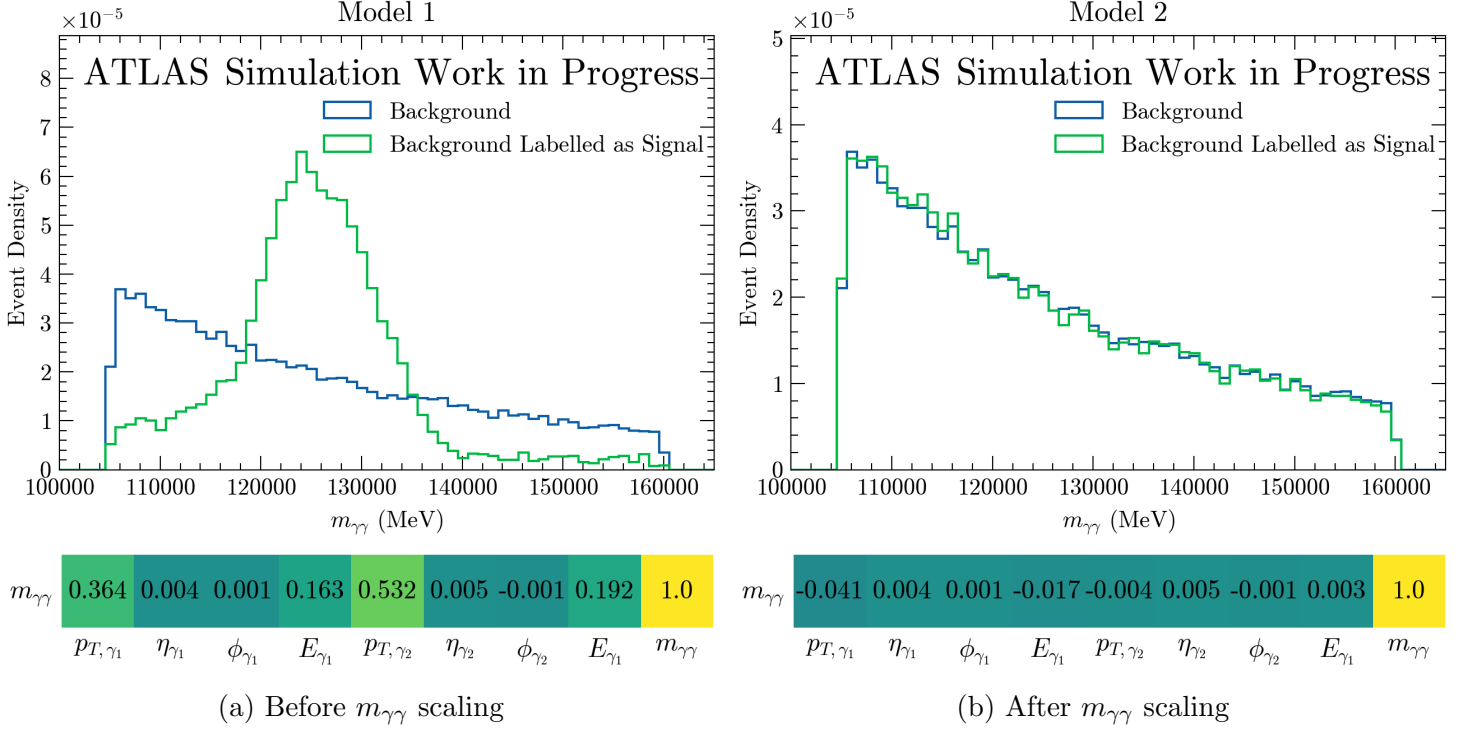


Figure 6: Before and after dividing the p_T 's and E 's by $m_{\gamma\gamma}$. Top: Background events incorrectly labelled as signal by Model 1 and Model 2, which were trained before and after $m_{\gamma\gamma}$ scaling respectively. Bottom: Background correlations of the features before and after $m_{\gamma\gamma}$ scaling.

The cause of this problem is due to a high correlation of the photon transverse momenta and energies with $m_{\gamma\gamma}$ as can be seen in Fig. 6a. Even though $m_{\gamma\gamma}$ is not one of the features used to train the model, because of the high correlation of other features with $m_{\gamma\gamma}$, the model learns about it and uses the peak of signal events. To avoid this, we can divide the transverse momenta and energies of the photons by $m_{\gamma\gamma}$. This step essentially breaks the correlation and gives us a set of features with little to no correlation with $m_{\gamma\gamma}$, as shown in Fig. 6b. We refer to this as $m_{\gamma\gamma}$ scaling.

Training a new model using the $m_{\gamma\gamma}$ scaled features, with the same optimizer and batch size as Model 1, we get Model 2 with a much worse performance of 61%. Since we have very few features and thus very little information of each event, this performance isn't too surprising. By taking away information of the $m_{\gamma\gamma}$ distribution (via $m_{\gamma\gamma}$ scaling), there is little information for the model to learn from. Again, plotting the classification predictions of background events as input data in Fig. 6b. Although we have worse performance, we retain information of the background shape and thus have trained a useful model.

We have thus shown the importance of avoiding any features that are correlated with $m_{\gamma\gamma}$ to avoid disturbing the background shape. In state of the art analyses, it is usual to discard any features with a linear correlation coefficient² higher than 5% [2].

²Calculated separately for background and signal. Feature is discarded if either is higher than 5%.

4 Feature Engineering

Moving onto a more realistic case, we use data that consists of p_T , η , ϕ and E of the 2 leading photons and the 4 leading jets. The dataset also includes the number of jets for each given event, so in total we have 25 features and 250000 samples with an equal number of signal and background events. Events with less than 4 jets have unphysical values for some of the features, for example when there are only 3 jets, the features corresponding to jet 4 kinematics have invalid values. To rectify this, we have set any of these values to the average³ for that feature so as to not disturb the distribution of the feature.

Instead of just taking these 25 features and using these for training, we have engineered new features based on Ref. [2]. All 49 features considered are shown in Tab. 1. Models 1 and 2 showed the importance of avoiding $m_{\gamma\gamma}$ correlations, so we discarded any feature with a linear correlation higher than 5%, leaving us with 41 features.

Included Features	$p_{T,\gamma_1}, p_{T,\gamma_2}, \eta_{\gamma_1}, \eta_{\gamma_2}, \phi_{\gamma_1}, \phi_{\gamma_2},$
	p_T, η, ϕ of the 4 leading jets,
	$p_{T,jj}^\dagger, m_{jj}$, and $\Delta y, \Delta\phi, \Delta\eta$ between j_1 and j_2 ,
	$p_{T,\gamma\gamma j_1}, p_{T,\gamma\gamma jj}^\dagger,$
	$\Delta y, \Delta\phi, \Delta R$ between the $\gamma\gamma$ and jj systems,
	Invariant mass of system comprising of the 4 leading jets,
	Number of Jets, Number of Central Jets ($ \eta < 2.5$),
	p_T of highest- p_T jet out of the 4 leading jets,
	Scalar sum of the p_T of the 4 leading jets,
	$p_{T,\gamma\gamma}$ projected to the thrust axis of the $\gamma\gamma$ system ($p_{Tt,\gamma\gamma}$),
Discarded Features	$\Delta\eta$ between the 2 photons, $\eta^{Zep} = \frac{\eta_{\gamma\gamma} - \eta_{jj}}{2}$,
	$\phi_{\gamma\gamma}^* = \tan\left(\frac{\pi - \Delta\phi_{\gamma\gamma} }{2}\right) \sqrt{1 - \tanh^2\left(\frac{\Delta\eta_{\gamma\gamma}}{2}\right)},$
	$\frac{p_{T,\gamma\gamma}}{m_{\gamma\gamma}}, \eta_{\gamma\gamma}, p_{T,j_F}, \eta_{j_F},$
	$\Delta\theta_{\gamma\gamma j_F}$ between the $\gamma\gamma$ system and j_F
	$p_{T,\gamma\gamma}, m_{\gamma\gamma j_1}, m_{\gamma\gamma jj}, m_{\gamma\gamma j_F}, p_{T,\gamma_1}, p_{T,\gamma_2},$
	$\cos\theta_{\gamma\gamma}^* = \left \frac{(E_{\gamma_1} + p_{z,\gamma_1})(E_{\gamma_2} - p_{z,\gamma_2}) - (E_{\gamma_1} - p_{z,\gamma_1})(E_{\gamma_2} + p_{z,\gamma_2})}{m_{\gamma\gamma} + \sqrt{m_{\gamma\gamma}^2 + p_{T,\gamma\gamma}^2}} \right $

Table 1: Engineered features based on Ref. [2]. Included features are those that are acceptable for training. Discarded features are those with a correlation with $m_{\gamma\gamma}$ higher than 5% and not appropriate to use. Features with \dagger have two versions with different jet p_T requirements. One version using jets with $p_T > 25$ GeV and another using jets with $p_T > 30$ GeV. Both versions are considered separate features. The $\gamma\gamma$ and jj notations refer to the systems composed of the two leading photons and jets respectively. The two leading photons are labelled γ_1 and γ_2 , the two leading jets as j_1 and j_2 , and the most forward jet out of the 4 leading jets is denoted as j_F .

³Average is calculated by excluding unphysical values.

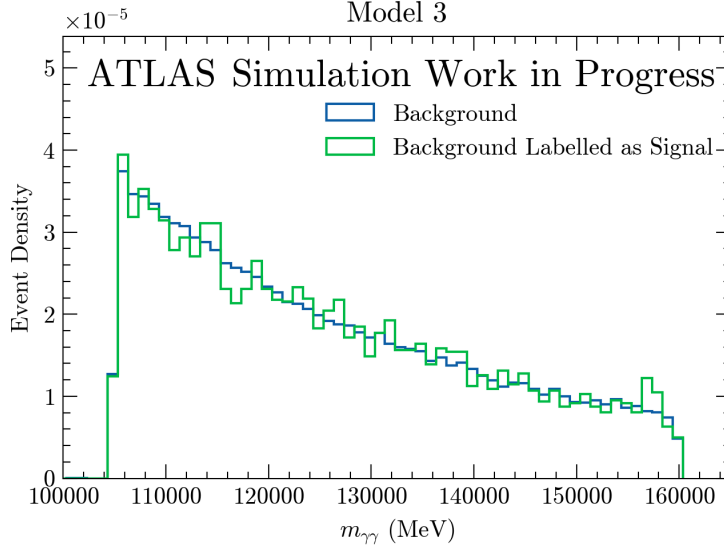


Figure 7: Background events mistaken as signal by Model 3, which was trained using the 41 engineered features.

5 Classical NN Approach

In this section we discuss 3 models we trained using different number of features. All 3 models were trained using the ADAM optimizer with a learning rate of 0.001 and a batch size of 1000. To avoid over-fitting and to reduce the training times, we used an earlystopping callback function that monitors the validation accuracy and stops training once performance stops improving. All classical models were implemented using the Tensorflow package, specifically the Keras module.

After applying min-max scaling on the 41 engineered features, we split the dataset into 80% training, 10% validation and 10% testing. We then used this to train another fully connected neural network, with an excellent testing accuracy of 95%. We refer to this as Model 3. Plotting the predictions of Model 3 on the background events in Fig. 7, we find that indeed background shape has not been disturbed as required.

5.1 Feature Ranking

In preparation for the quantum classifiers, we ranked the features based on importance. The goal here is to be able to reduce the number of features while retaining as much information as possible. To do this we used the permutation feature ranking algorithm based on Ref. [14]. It goes as follows:

Input: Trained model, testing data \mathbf{X} and true outcome \mathbf{Y} .

1. Calculate \mathbf{L}_{orig} , the loss of the model based on the predictions on the testing data. In our case the binary cross-entropy.
2. For each feature \mathbf{f} :
 - Generate a new training data set by permuting or shuffling feature \mathbf{f} in \mathbf{X} . This breaks any association between feature \mathbf{f} and the true outcome y .

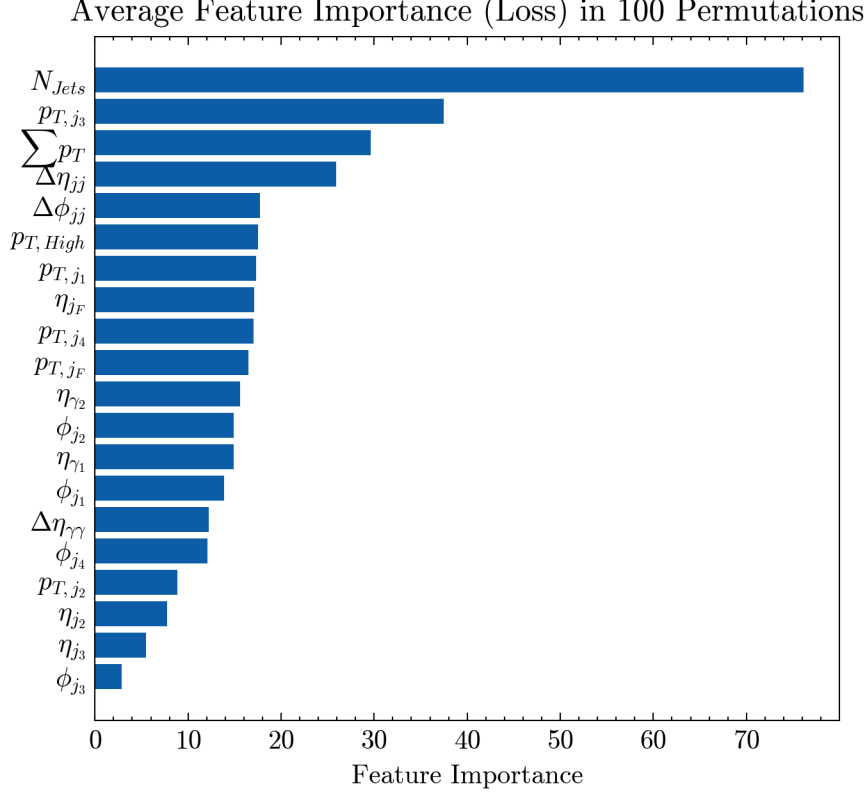


Figure 8: Top 20 features ranked by feature importance using the permutation feature ranking algorithm.

- Calculate L_{perm} , the loss of the model based on the predictions on the permuted data.
- Calculate permutation feature importance as the quotient $FI_f = \frac{L_{orig}}{L_{perm}}$

3. Sort features by descending FI .

Given the random nature of step 2 of the algorithm, results vary between different runs of it. To obtain a more reliable result, we instead ran step 2 100 times and ranked the features based on the average feature importance. Our results as shown in Fig. 8. We have only shown the top 20 features because the other features have very similar importances and are almost negligible, see Appendix C for the other features. It is therefore likely that the rest of the features are not going to affect model performance if excluded.

To test this, we trained 2 more dense neural networks with earlystopping, using differing number of features. The first one using the top 20 features from the feature ranking, and the second one using the top 6. We refer to these as Model 4 and Model 5, with testing accuracies of 94% and 86% respectively. Only looking at the accuracies tells us that the feature ranking is accurate and works well as a feature reduction method.

5.2 Results and Discussion

In Fig. 9 we compare the Receiver Operating Characteristic (ROC) curves of the models. This is a curve that illustrates the variation of signal efficiency and background efficiency

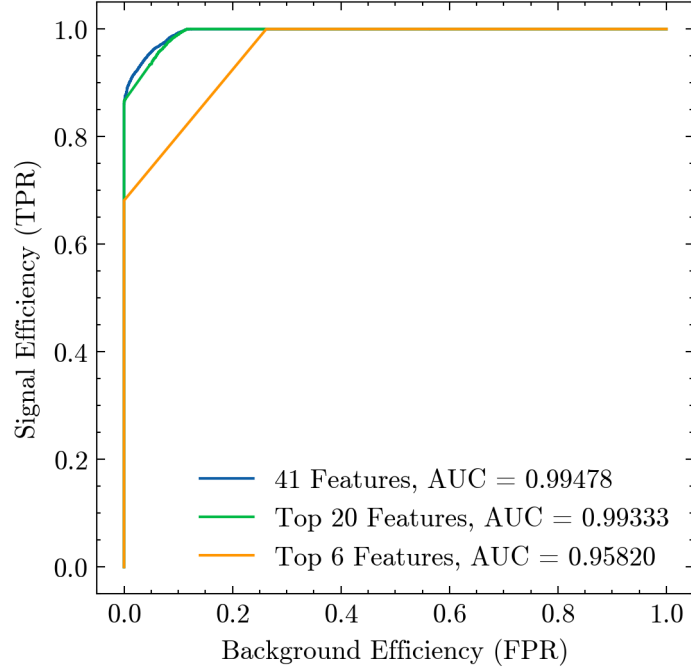


Figure 9: ROC comparison for Models 3, 4 and 5, which are trained using 41 features, top 20 and top 6 respectively.

as we vary the thresholds for each class. It is clear that there is very little difference between the full feature set and the top 20 features, confirming our previous suspicions. So we can safely discard the rest of the features and only consider the top 20 without significantly affecting performance. Model 5 with the top 6 features serves as a benchmark model to compare with the quantum models. Reducing the number of features, reduces the information fed into the model, so it isn't surprising that the AUC of Model 5 is much worse than the other 2 models. Overall, this shows that our chosen method of feature reduction is successful. However, to properly compare the permutation ranking method in a HEP context, future work should include training models using other methods such as autoencoders and principal component analysis to properly assess its effectiveness.

6 Quantum VQC Approach

For the VQC, we implemented it as a 6-qubit circuit and thus taking 6 features as input. The reason for choosing such few features is due to training time constraints as we will see in a later section. We implemented the circuit using the qiskit [15] packages and ran them using an ideal quantum simulator from the same packages, see Appendix for code.

We have 2 different versions of the VQC, both of them use a second-order Pauli-Z evolution circuit as the feature map, also known as a ZZ Feature Map. It consists of a sequence of alternating Phase and CX gates. We used only 1 repetition of the feature map. From our testing, increasing the number of repetitions only decreased performance. For the variational ansatz, one of the VQCs used the Real Amplitudes circuit, which is composed of alternating Y rotations and CX gates. The other VQC uses the EfficientSU2 circuit, which consists of Y and Z rotations and CX gates for entanglement. Unlike with

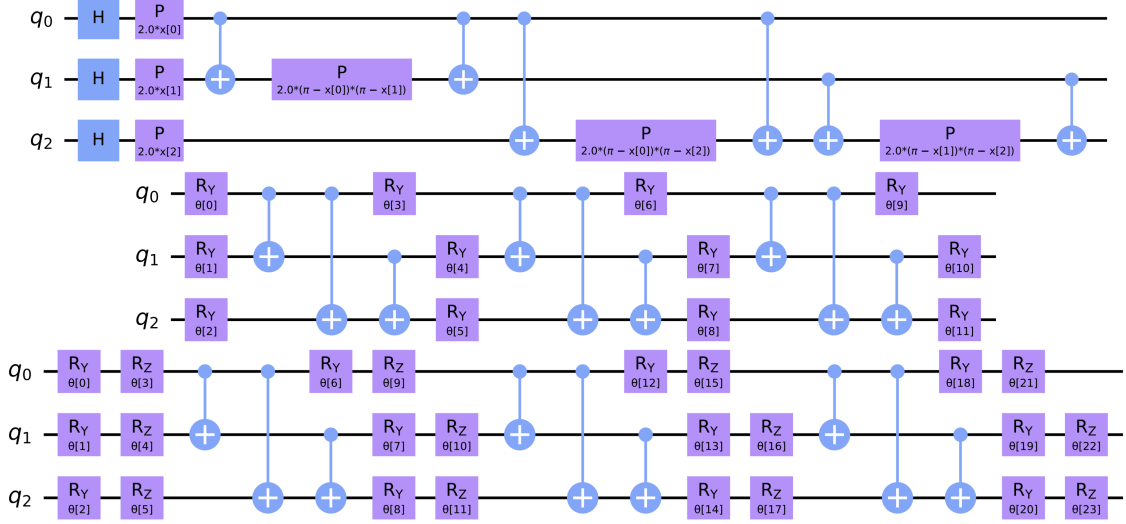


Figure 10: Top: ZZ Feature Map for 3 qubits, with a single repetition. Middle: Real Amplitudes ansatz for 3 qubits, with 3 repetitions. Bottom: EfficientSU2 ansatz for 3 qubits, with 3 repetitions.

the feature map, we found that having a few repetitions of the ansatz was beneficial, but too many would decrease performance, so we settled with 3 repetitions. We have shown both a 3-qubit version of the feature map and ansatz in Fig. 10, due to lack of space, the full 6-qubit circuits are in Appendix E.

Before training, we note that our feature map consists of rotation gates. Given this, instead of Min-Max scaling so that values are between 0 and 1, it is more appropriate to scale them between $-\pi$ and π . We scaled like so 1000 training samples and 5000 testing ones and used these for training, along with the COBYLA⁴ optimizer to train the VQC for 200 iterations. Most of the time taken was in training the models, so we only have 1000 training samples. For testing, we could afford spending more time on this so we used 5000 testing samples to obtain more reliable results. Other things to note are that we do not have a validation dataset in the quantum case. This is due to qiskit not supporting them natively, or at least not to our knowledge. For the same reason, we did not use an early stopping function. We trained both VQCs in the same way resulting in testing accuracies of approximately 70% for both.

6.1 Discussion

In Fig. 11 we show the ROC curve of the VQCs. Immediately we can tell that the performance is much worse than the classical models. Even our worst neural network using the top 6 features could achieve a signal efficiency of almost 70% with virtually no false positives. The VQCs are capable of achieving similar signal efficiency but with a much larger false positive rate of around 30%. Furthermore, we note that using the EfficientSU2 ansatz performs slightly better than the RealAmplitudes ansatz.

Our results differ largely from previous works done by Belis *et al.* and Wu *et al.* in Refs. [3, 4]. It is important to note, however that there are some significant differences

⁴Stands for Constrained Optimization BY Linear Approximation.

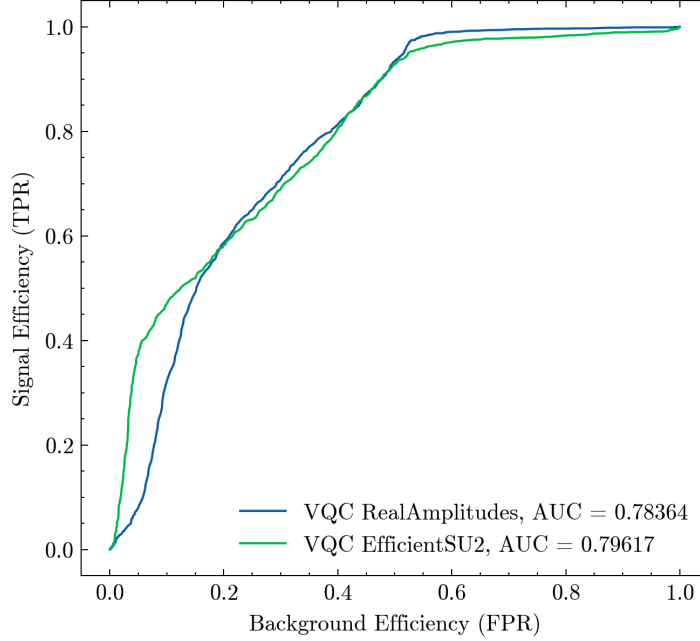


Figure 11: ROC curve for the VQC, trained using the top 6 features.

between our models and theirs. Specifically, our VQCs are optimized using the non-gradient-based optimizer COBYLA. Belis *et al.* used the ADAM optimizer, however we were unable to get this working. Since QML is still a very new field, there is a lack of resources and documentation, due to this we couldn't find a working test case using the ADAM optimizer. We also tried training a VQC with the ADAM optimizer on the Iris dataset from scikit-learn [16] to make our own test case, however this also proved unsuccessful. It seems that the optimizer would not run a single iteration, we suspect there might be a bug in the source code used for the optimizer or with the way the VQC class from qiskit handles it. Wu *et al.* instead uses a simultaneous perturbation stochastic approximation algorithm or SPSA. For the same reason, we were unable to use the SPSA optimizer.

Models 4 and 5 showed that our method of feature reduction does work successfully, however when applied to the quantum case we can't say the same. It is clear that further work is needed here and other methods of feature reduction, such as autoencoders and principal component analysis, need to be applied and compared.

7 Limitations of Quantum Models

Having trained a variety of models using both classical ML and quantum ML methods, we want to discuss some of the key limitations we encountered during the development process. In particular, quantum computing is still a very new field, and so operating quantum computers is both expensive and time consuming. So we will look at these limitations in the context of training time and things that require improvement before QML can be consistently used.

7.1 Debugging and Software

Given the time constraints, it is not ideal to be testing programs on the quantum computers as this can be very expensive, both in time and money. Instead, we can use quantum simulators that run locally on a classical computer to test these programs before running them on the quantum computers. However, memory becomes a bottleneck for the type of circuits that can be simulated. When storing quantum states on classical hardware, this is done by storing them as an array of shape $(1, 2^{N_{\text{Qubits}}})$. Each element of the array being a complex number that is stored as 2 long float numbers for the real and imaginary parts. Each element thus requires 16 Bytes of memory, 8 Bytes for each part. Thus we get that the minimum memory required to simulate a program is:

$$16 \times 2^{N_{\text{Qubits}}} \text{ Bytes} \quad (1)$$

To put it into perspective, current supercomputers that have around a petabyte of RAM, would be able to store about 45 qubits, while an average computer with 16 GB of RAM could store about 25 qubits. This means that functionally, we are limited in the number of qubits we can easily test. This is especially relevant to cases like ours where we have a high number of variables to consider due to the nature of the experiments at the LHC.

Other than simulation, we found that QML has a lack of software support for many typical functions that are available to classical ML. To give some examples, early stopping and validation are not natively supported by quantum computing packages such as qiskit. Other things include optimizer support. As stated previously, we were unable to get gradient-based optimizers to work properly with the VQC and were only able to use some non-gradient-based ones such as COBYLA.

7.2 Balancing Accuracy and Training Time

Our classical model could handle 250000 training samples easily while running hundreds of epochs within a couple of minutes. Meanwhile, the VQC struggled to run even a couple of iterations. This goes to show that quantum methods are in general slow to run and thus it is important to understand what affects training time. In our experience, the 3 most important things are the number of training samples, optimizer iterations and the number of qubits. All 3 of these drastically increase training time but also allow the models to take in more information and potentially perform better. Then the main issue here is balancing the performance of the model with the amount of time we are willing to spend training the model.

7.3 Circuit Complexity and Decoherence

Our quantum models are run on ideal quantum simulators, however, real quantum computers are prone to making errors due to imperfect qubits and gates. Furthermore, qubits can only store a quantum state for a limited time, after which they experience decoherence and lose the information of the quantum state. These two issues heavily impact whether a program can be feasibly run on a quantum computer.

Qubits	Basis Gates	T1 (μ s)	T2 (μ s)	Readout Error	SX Error	RZ Error	CZ Error
156	CZ, ID, RX, RZ, RZZ, SX, X	206.15	119.37	9.277×10^{-3}	2.432×10^{-4}	0	3.268×10^{-3}

Table 2: Relevant specifications of the `ibm_marrakesh` quantum computer taken on 24/03/2025 at 16:05. Gate errors shown are the median values from the calibration process. RZ error is 0 because RZ gates are implemented purely in software and do not involve any operations on the qubits. Source: [17]

For example, let's consider running our VQC circuit on one of IBM's publicly available quantum computers. Our computer of choice is the `ibm_marrakesh`, its relevant specifications are shown in Tab. 2. Without accounting for decoherence, the probability of a circuit to execute successfully with no errors is simply the probability of all gates being successfully applied:

$$P(\text{Program success}) \approx P(\text{all gates succeed})$$

Our VQC is composed of 6 Hadamard, 21 Phase (equivalent to RZ gates), 75 CX and 24 RY gates. After transpiling⁵ the VQC circuit, we have 351 SX, 229 RZ and 202 CZ gates. So the probability of all gates executed without error is:

$$\begin{aligned}
P(\text{all gates succeed}) &= P(\text{SX success})^{351} P(\text{RZ success})^{229} P(\text{CZ success})^{202} \\
&= (1 - 2.432 \times 10^{-4})^{351} (1 - 0)^{229} (1 - 3.268 \times 10^{-3})^{202} \quad (2) \\
&\approx 47\%
\end{aligned}$$

A probability of 47% is already not promising. The probability of decoherence is given by exponential decays, which further lowers the chances of successfully running the circuit. This means that in practice, it may be necessary to make many reruns of a program to make sure there are no errors. If quantum circuits are to be pursued in detail, there will need to be advancements in improving quantum computing hardware.

8 Conclusion

This report examined the performance of both classical Neural Networks (NNs) and the Variational Quantum Classifier (VQC) in the classification of Higgs boson events. Our analysis highlighted the strengths and limitations of each approach.

The classical neural network demonstrated strong predictive capabilities and a superiority to their quantum counterpart. Due to their ability to handle large datasets without compromising the time taken to train them, they are a very reliable approach to Higgs classifiers.

Our feature engineering process played a crucial role feature reduction. By selecting the most relevant features, we can train models using less features while retaining as much information as possible. Using these we trained a couple of VQCs.

⁵Translates a circuit into an appropriate and optimized form in terms of the native gates available to `ibm_marrakesh`.

Contrasting with the NNs, the VQC, while still in its early stages, exhibited promising results that suggest quantum classification could become viable as quantum hardware advances. However, its scalability severely hinders its ability to perform closer to the classical neural networks. Additionally, the lack of a mature quantum software framework makes working with quantum classifiers challenging. Advancements in error correction or reducing decoherence could eventually allow VQCs to more easily compete against well established classical methods.

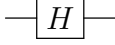
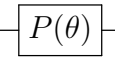
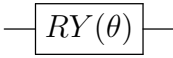
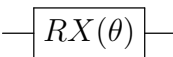
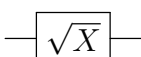

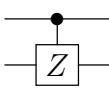
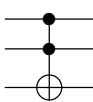
While classical NNs remain the dominant approach for Higgs classification, this study reinforces the importance of further research into quantum machine learning. As quantum technology matures, the integration of quantum classifiers into high-energy physics workflows could offer computational advantages that surpass classical methods.

References

- [1] ATLAS Collaboration. “Measurement of the $H \rightarrow \gamma\gamma$ and $H \rightarrow ZZ^* \rightarrow 4\ell$ cross-sections in pp collisions at $\sqrt{s} = 13.6$ TeV with the ATLAS detector”. *Eur. Phys. J. C* 84.1 (2024), 78. arXiv: [2306.11379 \[hep-ex\]](#).
- [2] ATLAS Collaboration. “Measurement of the properties of Higgs boson production at $\sqrt{s} = 13$ TeV in the $H \rightarrow \gamma\gamma$ channel using 139 fb⁻¹ of pp collision data with the ATLAS experiment”. *JHEP* 07 (2023), 088. arXiv: [2207.00348 \[hep-ex\]](#).
- [3] V. Belis et al. “Higgs analysis with quantum classifiers”. *EPJ Web Conf.* 251 (2021), 03070. arXiv: [2104.07692 \[quant-ph\]](#).
- [4] S. L. Wu et al. “Application of quantum machine learning using the quantum kernel algorithm on high energy physics analysis at the LHC”. *Phys. Rev. Res.* 3.3 (2021), 033221. arXiv: [2104.05059 \[quant-ph\]](#).
- [5] A. D. Kennedy. “Quantum computing project slides”. *Quantum Computing Project course at University of Edinburgh* (2024).
- [6] O. Daei et al. “Optimized Quantum Circuit Partitioning”. *Int. J. Theor. Phys.* 59 (2020), 3804–3820. arXiv: [2005.11614 \[quant-ph\]](#).
- [7] “Quantum neural networks”. Date Accessed: 25 March 2025. URL: https://qiskit-community.github.io/qiskit-machine-learning/tutorials/01_neural_networks.html.
- [8] P. Nason. “A New method for combining NLO QCD with shower Monte Carlo algorithms”. *JHEP* 11 (2004), 040. arXiv: [hep-ph/0409146](#).
- [9] S. Frixione et al. “Matching NLO QCD computations with Parton Shower simulations: the POWHEG method”. *JHEP* 11 (2007), 070. arXiv: [0709.2092 \[hep-ph\]](#).
- [10] S. Alioli et al. “A general framework for implementing NLO calculations in shower Monte Carlo programs: the POWHEG BOX”. *JHEP* 06 (2010), 043. arXiv: [1002.2581 \[hep-ph\]](#).
- [11] T. Sjöstrand et al. “An introduction to PYTHIA 8.2”. *Comput. Phys. Commun.* 191 (2015), 159–177. arXiv: [1410.3012 \[hep-ph\]](#).

- [12] J. Alwall et al. “The automated computation of tree-level and next-to-leading order differential cross sections, and their matching to parton shower simulations”. *JHEP* 07 (2014), 079. arXiv: [1405.0301 \[hep-ph\]](#).
- [13] GEANT Collaboration. “Geant4-a simulation toolkit”. *Nucl. Instrum. Methods Phys. Res., Sect. A* 506.3 (2003), 250–303.
- [14] A. Fisher et al. “All Models are Wrong, but Many are Useful: Learning a Variable’s Importance by Studying an Entire Class of Prediction Models Simultaneously”. *J. Mach. Learn. Res.* 20.177 (2019), 1–81. arXiv: [1801.01489 \[stat.ME\]](#).
- [15] A. Javadi-Abhari et al. “Quantum computing with Qiskit” (May 2024). arXiv: [2405.08810 \[quant-ph\]](#).
- [16] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. *J. Mach. Learn. Res.* 12 (2011), 2825–2830.
- [17] “*ibm_marrakesh specifications*”. Date Accessed: 24 March 2025. URL: https://quantum.ibm.com/services/resources?system=ibm_marrakesh.

A Commonly Used Quantum Gates

Gate/operator	Description	Circuit form	Matrix
Hadamard (H)	Applies a Hadamard transform, maps the basis states $ 0\rangle \rightarrow \frac{ 0\rangle+ 1\rangle}{\sqrt{2}}$ and $ 1\rangle \rightarrow \frac{ 0\rangle- 1\rangle}{\sqrt{2}}$.		$\frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$
Phase/RZ	Single-qubit rotation about the Z axis. Phase gate is equivalent to Z Rotation up to a phase factor: $P(\theta) = e^{i\theta/2} RZ(\theta)$.		$\begin{bmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{bmatrix}$
Rotation-Y (RY)	Single-qubit rotation about the Y axis..		$\begin{bmatrix} \cos(\theta/2) & -\sin(\theta/2) \\ \sin(\theta/2) & \cos(\theta/2) \end{bmatrix}$
Rotation-X (RX)	Single-qubit rotation about the X axis.		$\begin{bmatrix} \cos(\theta/2) & -i \sin(\theta/2) \\ i \sin(\theta/2) & \cos(\theta/2) \end{bmatrix}$
Square-Root (SX)	Applies a superposition based on the square root of Pauli-X gate, which is an RX with $\theta = \pi$.		$\frac{1}{2} \begin{bmatrix} 1+i & 1-i \\ 1-i & 1+i \end{bmatrix}$
Controlled-NOT (CNOT, CX)	2-qubit gate that flips target qubit if control qubit is in the $ 1\rangle$ state. Control is denoted with a dot and target with \oplus .		$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$
Controlled-Z (CZ)	2-qubit gate that flips target qubit phase if control qubit is in the $ 1\rangle$ state.		$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$
Toffoli (CCX)	3-qubit gate that flips target qubit if both control qubits are in the $ 1\rangle$ state.		$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$

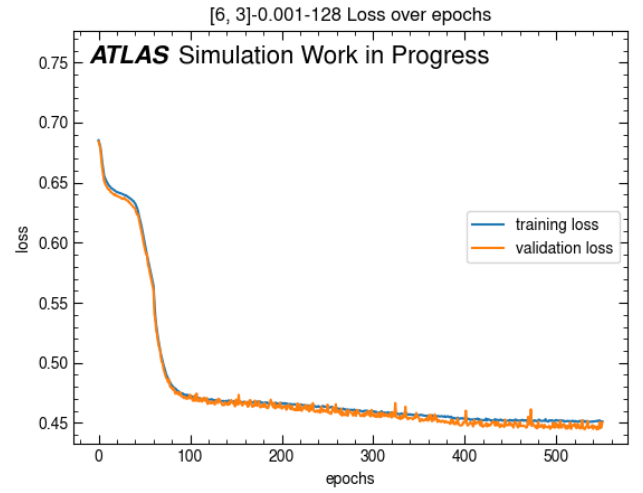
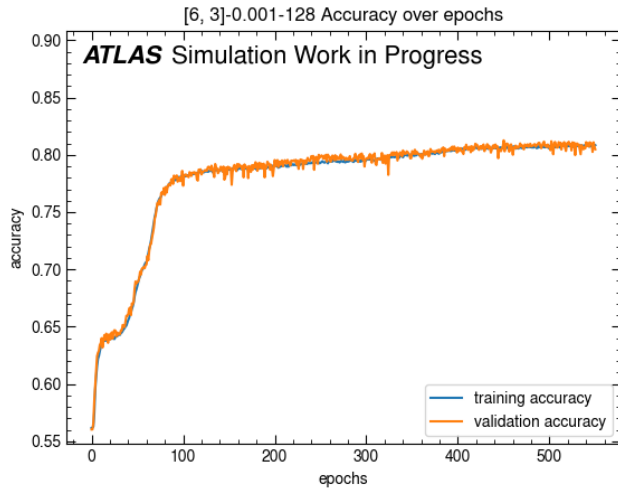
Descriptions, circuit symbols and matrix representations of some commonly used gates.

B Model 1 and 2

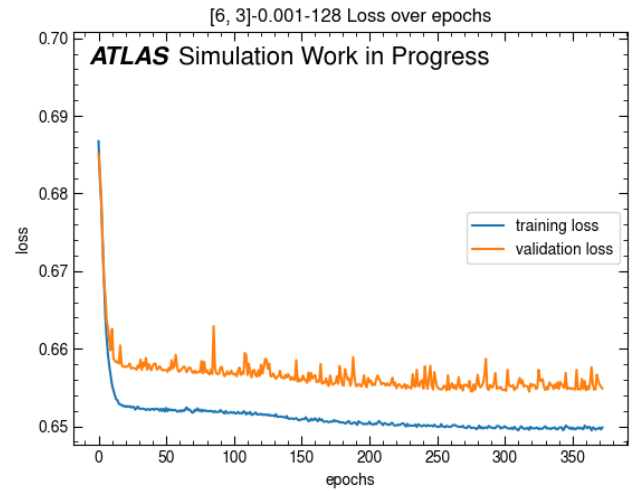
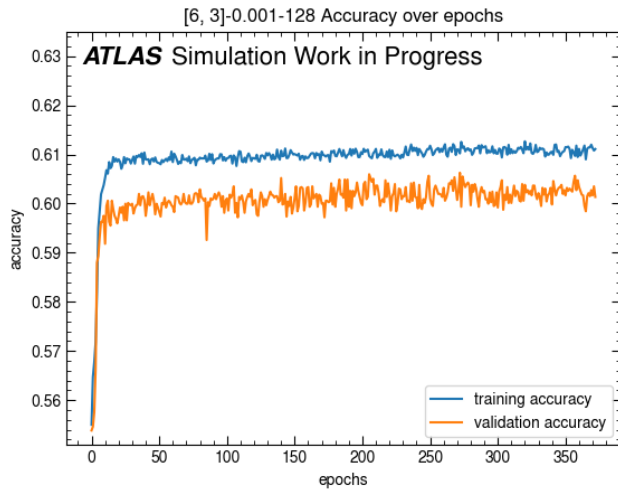
Both models use the same architecture with fully connected layers. Uses ADAM optimizer with a learning rate of 0.001 and batch size of 128. Data is Min-Max scaled between 0 and 1.

	Input Layer	Hidden Layer 1	Hidden Layer 2	Output Layer
Neurons	8	6	3	1
Activation Function	ReLU	ReLU	ReLU	Sigmoid

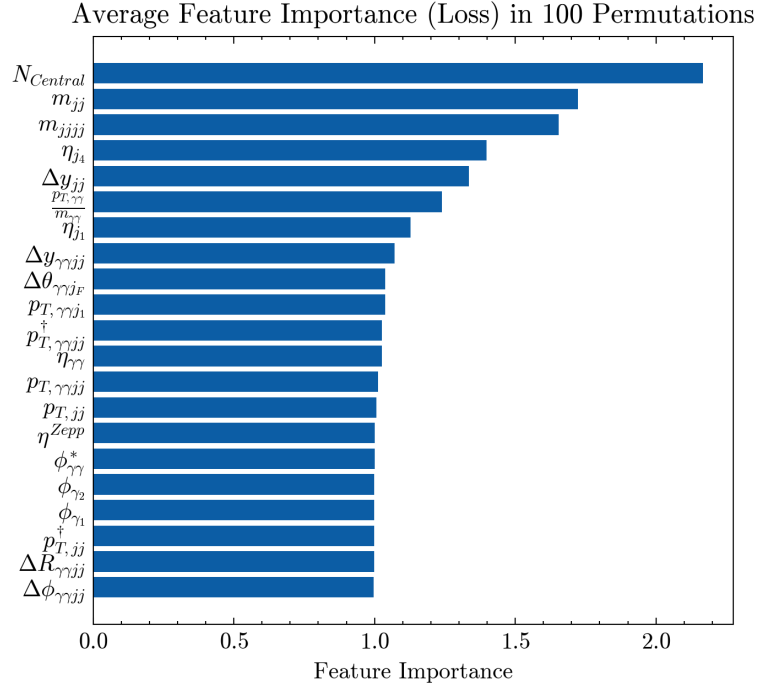
Model 1 Accuracy and Loss while training.



Model 2 Accuracy and Loss while training. Note that we do observe over-fitting in this model, however since this model is only made to showcase the importance of $m_{\gamma\gamma}$ independent features, we decided to not spend time on optimizing it since the model is only meant to be illustrative.



C Rest of Feature Ranking

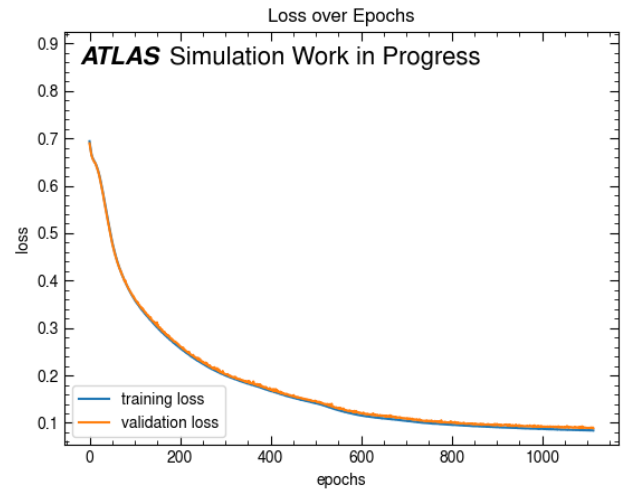
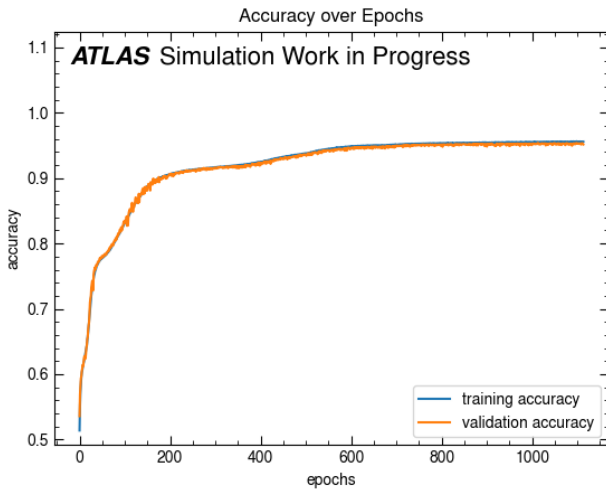


D Model 3, 4 and 5

Model 3 uses the following architecture with dense layers. Uses ADAM optimizer with 0.001 learning rate, batch size of 1000. Data is Min-Max scaled between 0 and 1.

	Input Layer	Hidden Layer	Output Layer
Neurons	41	20	1
Activation Function	ReLU	ReLU	Sigmoid

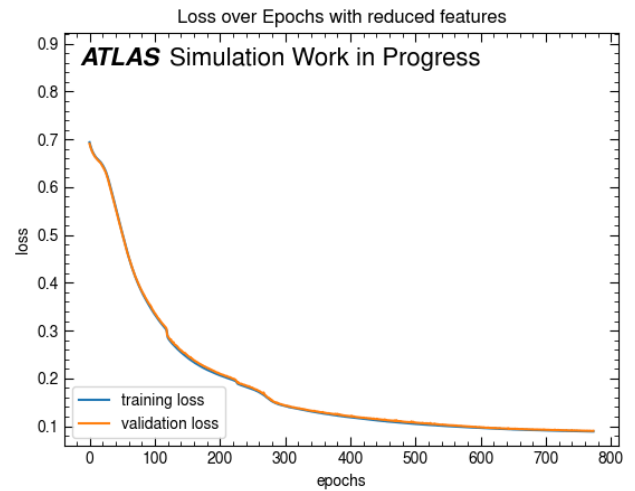
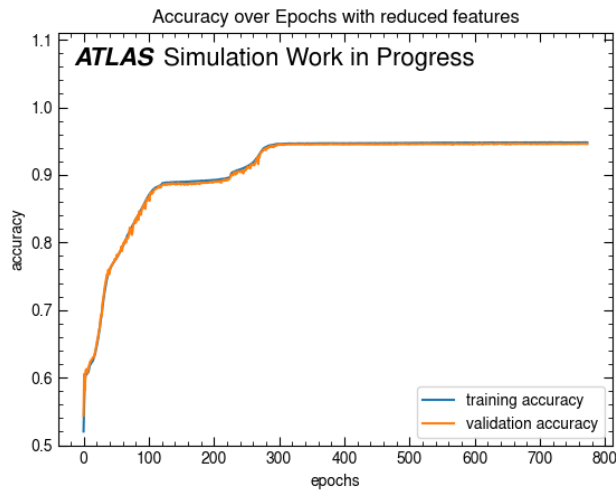
Model 3 Accuracy and Loss during training.



Model 4 is exactly the same as Model 3, except for the input and hidden layers.

	Input Layer	Hidden Layer	Output Layer
Neurons	20	10	1
Activation Function	ReLU	ReLU	Sigmoid

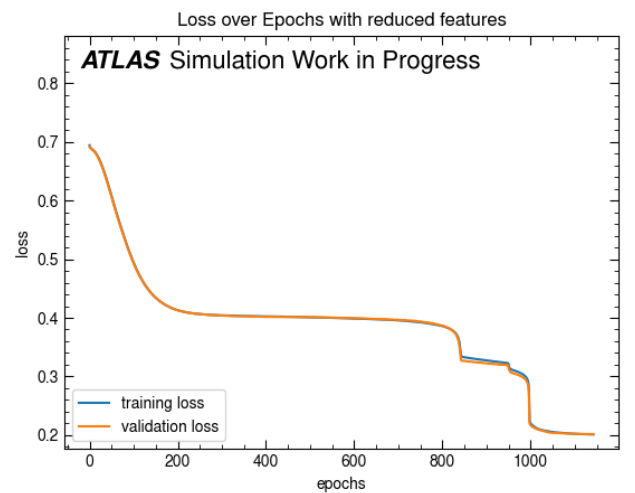
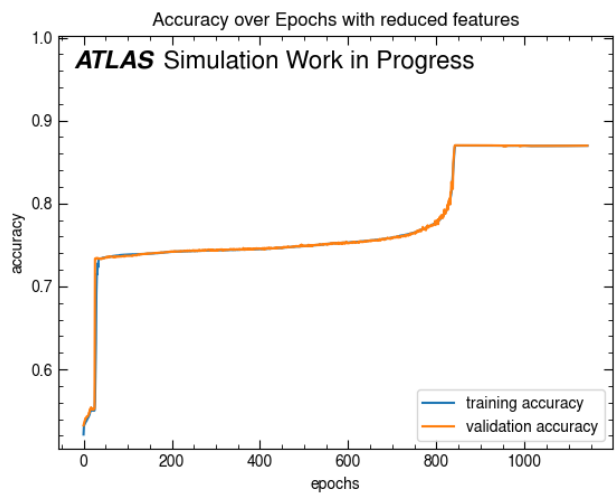
Model 4 Accuracy and Loss during training.



Model 5 is exactly the same as Model 3, except for the input and hidden layers.

	Input Layer	Hidden Layer	Output Layer
Neurons	6	3	1
Activation Function	ReLU	ReLU	Sigmoid

Model 5 Accuracy and Loss during training.

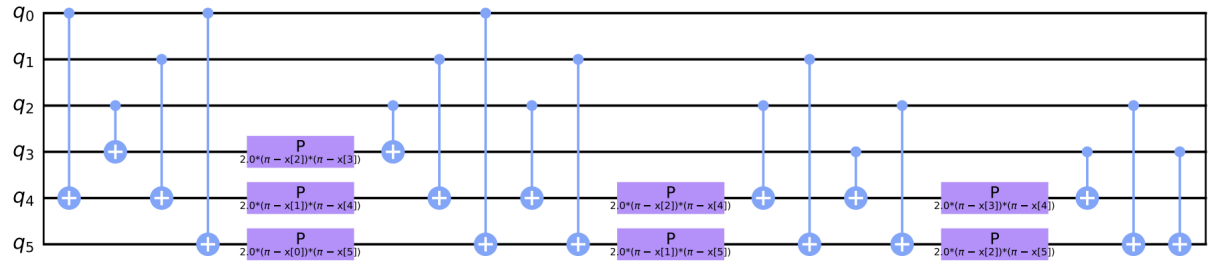
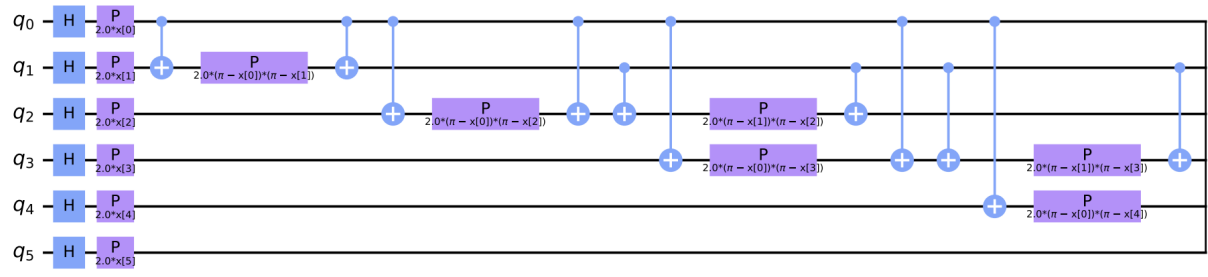


E VQC Circuit and Code

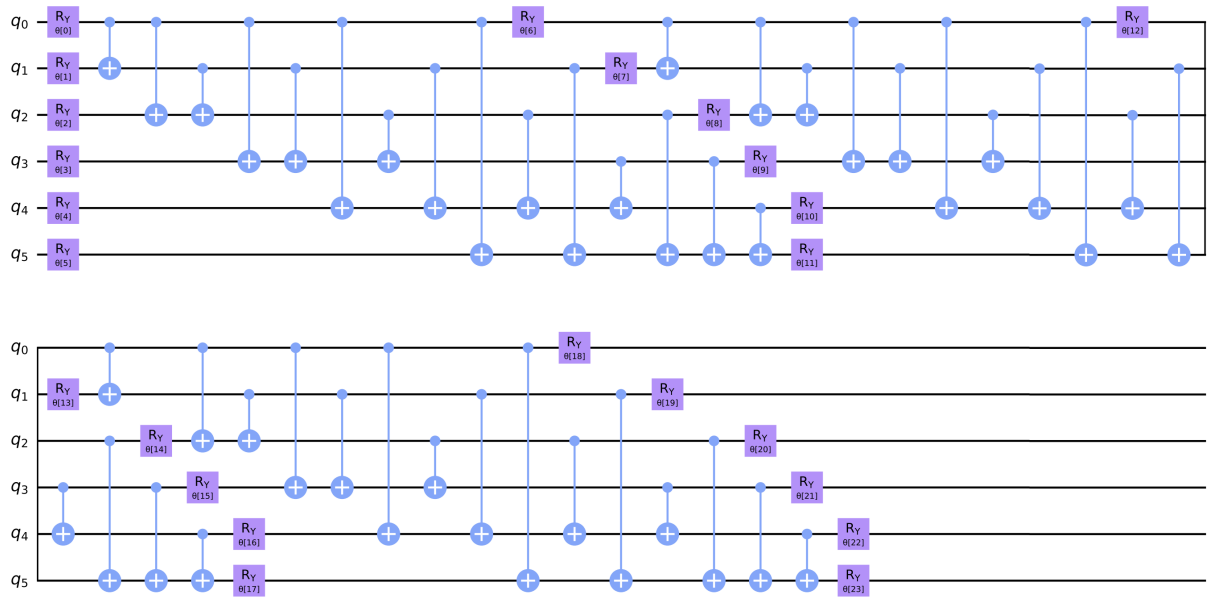
Relevant code for the VQC. We have 2 VQCs, the only difference between them is that the one using RealAmplitudes ansatz is trained on 200 iterations, while the EfficientSU2 ansatz is trained on 500 iterations.

```
1 from qiskit_machine_learning.algorithms.classifiers import VQC
2 from qiskit.circuit.library import ZZFeatureMap
3 from qiskit.circuit.library import RealAmplitudes, EfficientSU2
4 from qiskit_machine_learning.optimizers import COBYLA
5 from qiskit.primitives import StatevectorSampler as Sampler
6
7 num_features = 6
8
9 feature_map = ZZFeatureMap(feature_dimension=num_features, reps=1)
10 ansatz = EfficientSU2(num_qubits=num_features, entanglement='full',
11                      reps=3)
12 optimizer = COBYLA(maxiter=500)
13 sampler = Sampler()
14
15 objective_func_vals = []
16 plt.rcParams["figure.figsize"] = (12, 6)
17
18 def callback_graph(weights, obj_func_eval):
19     clear_output(wait=True)
20     objective_func_vals.append(obj_func_eval)
21     print("Optimizer Iteration:", len(objective_func_vals))
22     print(time.time()-start)
23
24 vqc = VQC(sampler=sampler,
25          feature_map=feature_map,
26          ansatz=ansatz,
27          loss="cross_entropy",
28          optimizer=optimizer,
29          callback=callback_graph,
30          warm_start=False)
```

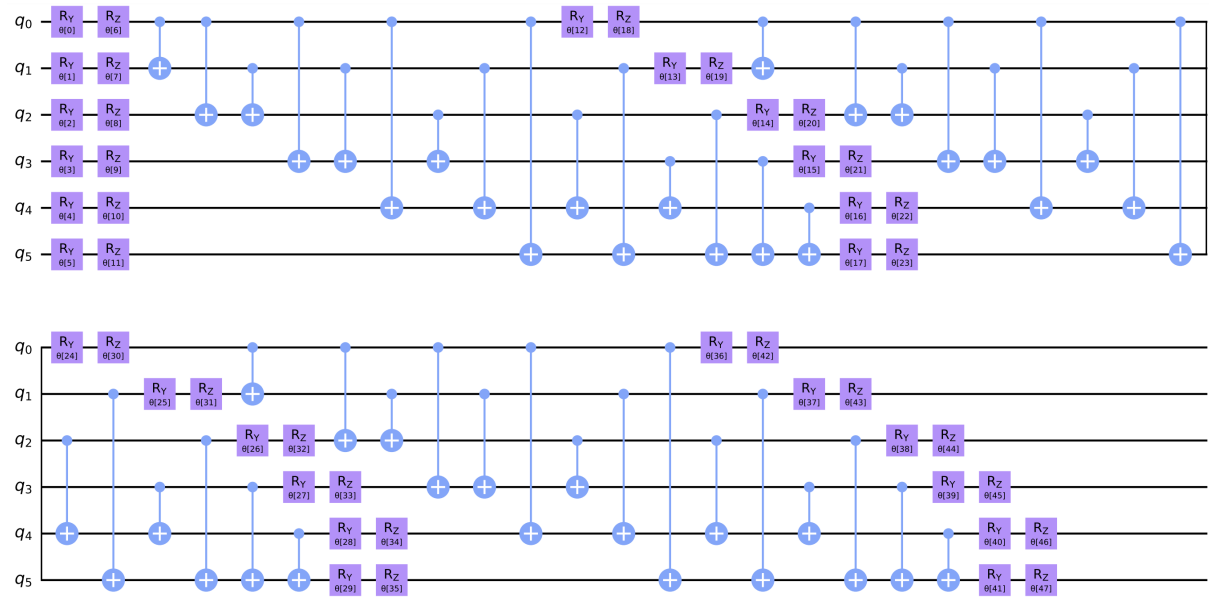
Circuit for ZZ Feature Map.



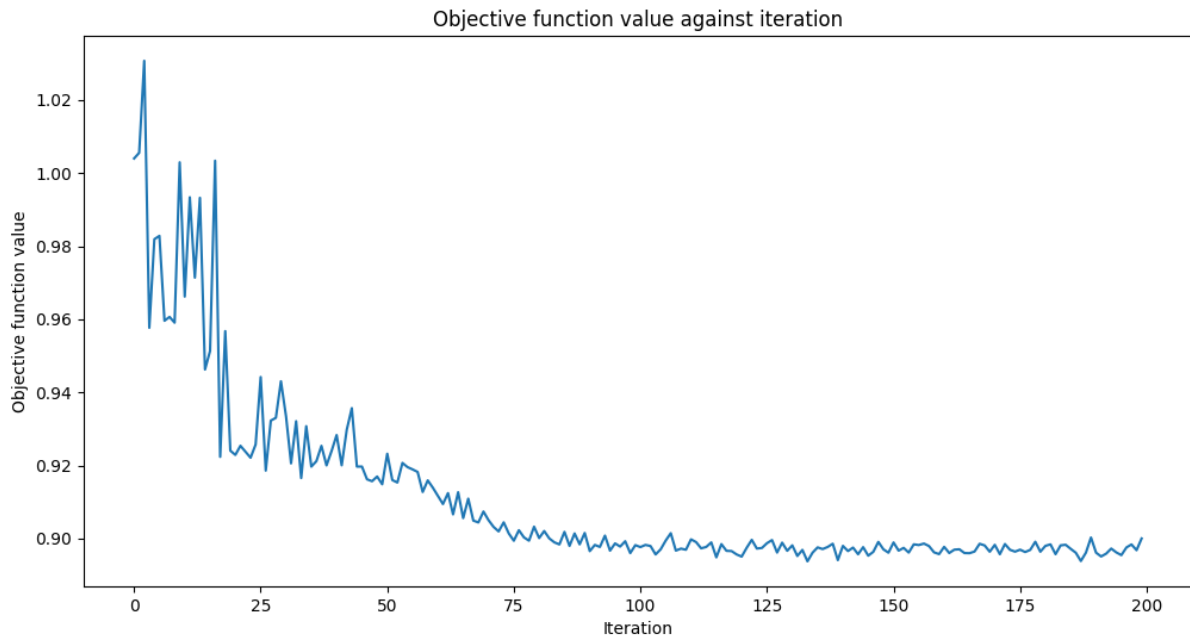
Circuit for RealAmplitudes ansatz.



Circuit for EfficientSU2 ansatz.



ZZ Feature Map + RealAmplitudes loss.



ZZ Feature Map + EfficientSU2 loss.

