### Written report

*The target audience of this report is the professor.* It should contain an in-depth discussion of the **theory** and **derivation of key results**, and the **tables, plots, and other relevant results generated by your software implementation** that reproduce the original work. It should follow loosely the **same structure** of the original paper, but be expressed in your own words.

It should also contain:

* rationale for choice of programming framework
* a discussion of any issues encountered that affected reproducibility

~~Further to this second point, reproducing the work of others is extremely challenging. It is common for papers to, e.g., omit simulation parameters, make unstated assumptions, or even include outright errors. As such, you will not be graded on how close your results are, but rather, how close your results are to what the paper~~ *~~intended~~* ~~to show, as a function of the information that was available to you~~. **A critical analysis of how reproducible the work is, and what your group's thought process was for handling ambiguitie**s, is therefore an important component of the report. ~~My hope is that this experience will allow you to think more carefully about how you present your own work in the future, and to promote open and reproducible scientific practices.~~

# 1 Introduction (Bobby)

Copy stuff from slides

# 2 Theory

## 2.1 Significance of Learning Temporal Data (Bobby)

For time-series data, it is very important if we can predict the output using past information.

## 2.2 What is Recurrent Neural Network (RNN) (Bobby)

F

## 2.3 Using Quantum Recurrent Neural Network (Bobby)

Talk about benefit of using QRNN,

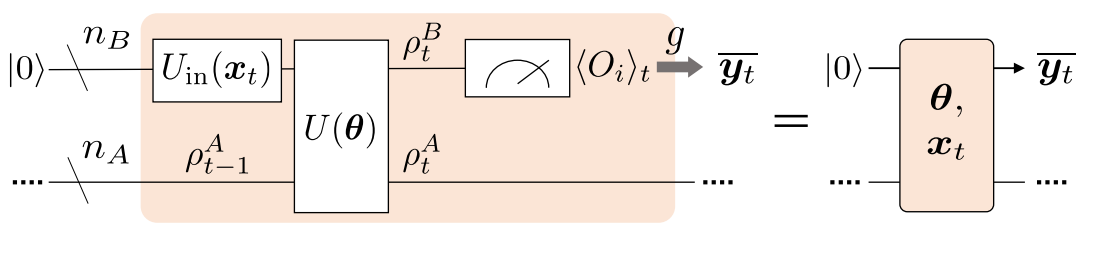
Tell reader, Why should use QRNN?

Finally, QRC, which stands for Quantum Reservoir Computing, has a similar structure as

QRNN, and it has shown promising quantum advantage

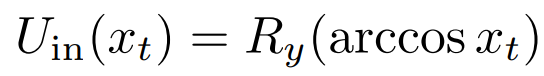
# 3 Derivation of Key Results in Software

The circuit for each iteration of the neural network is composed of two parts: the encoding step and the evolution step (Figure 3-1 below). The qubits are split into two groups. Group B will go through the encoding step and it will be reset to zero after each iteration. Then all qubits will be fed into the evolution step.



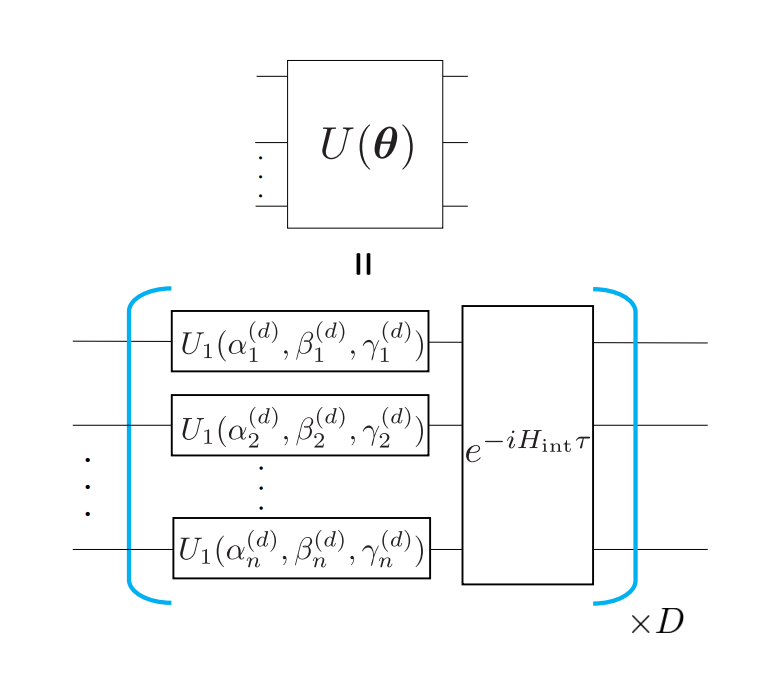
*Figure 3-1: Structure of each iteration.*

The encoding step *Uin* is just an RY rotation of the arccosine of the iteration’s input(*xt*), applied to the Group B qubits (Figure 3-2 below).



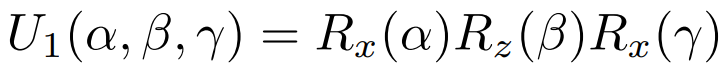
*Figure 3-2: RY Rotation used in encoding step.*

The evolution step *U(θ)* consists of a process that is repeated *D* times, which is specified as having the value of *D=3* in the paper (Figure 3-3 below).



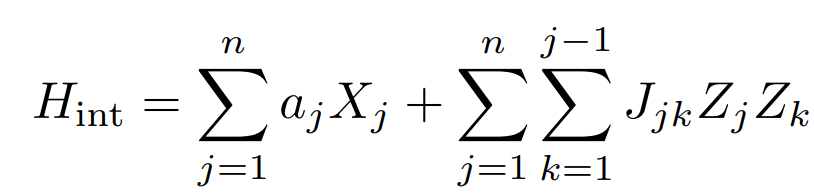
*Figure 3-3 Structure of the circuit used for each Evolution Step.*

Within this process, the training parameters α, β, γ are first used in *U1*,which is a combination of RX and RZ rotations (Figure 3-4 below).



*Figure 3-4: Series of RX and RZ rotations that have training parameters passed into them (start of evolution step).*

Then, a Hamiltonian *Hint* is constructed as a combination of PauliX’s and PauliZ’s (Figure 3-5 below). Coefficients *aj* and *Jjk* are random values from a uniform distribution on [-1,1] that are fixed during training, and *n* is the total number of qubits(*nA + nB = n*).



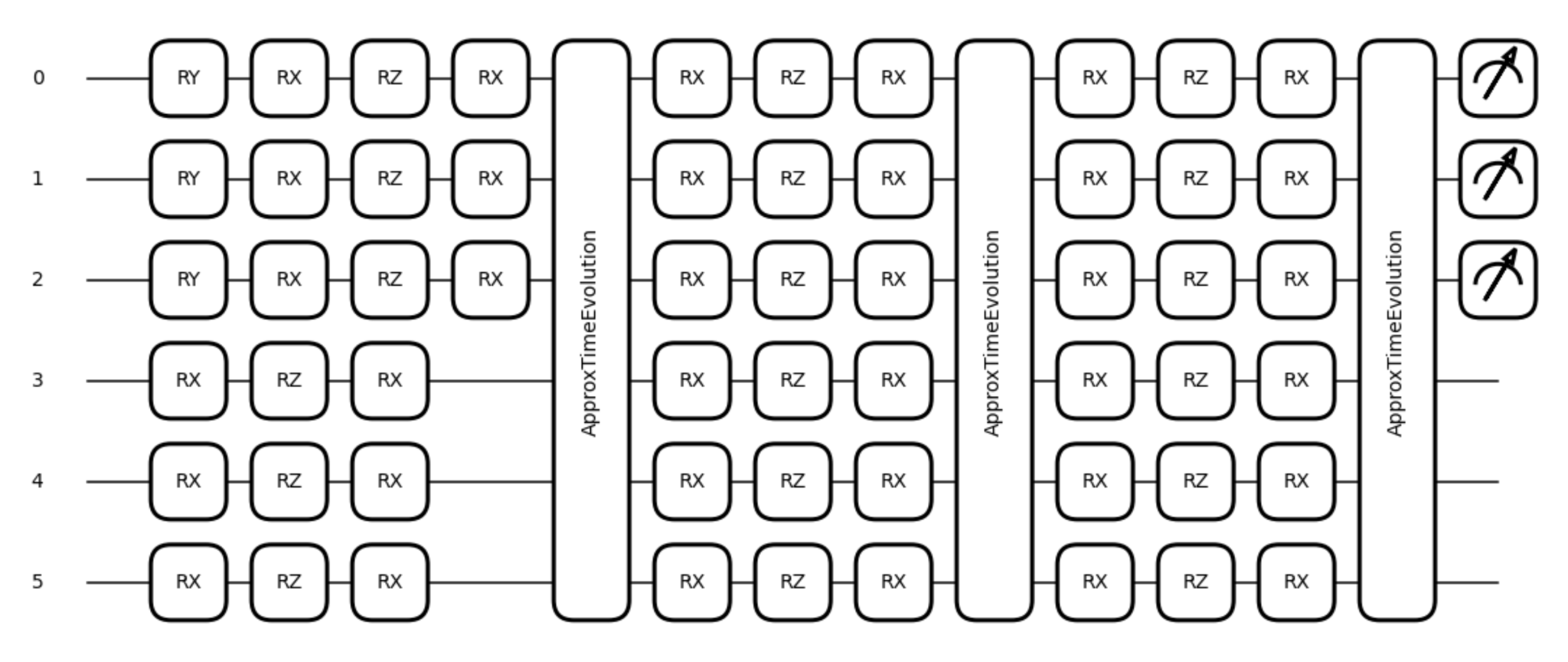
*Figure 3-5: Equation for Hamiltonian used in evolution step.*

The Trotterized time-evolution operator for this Hamiltonian is then applied to all qubits (*qml.ApproxTimeEvolution()* abstracts away the calculation of the Hamiltonian being multiplied by negative *i* and evolution time tau(τ), and then raised to the exponential as shown in Figure 3-3).

Finally, the Z-expectation values of all Group B qubits are measured (Figure 3-6 below), which is then averaged and multiplied by training parameter *c* to get the final prediction value of the current iteration.

As detailed in the paper, all training parameters are initialized to zero (except for *c*, which is set to 1) at the beginning. Also, tau is set to 0.2 because the paper suggests that only a range of tau provides an accurate prediction. A smaller value of tau will lead to less past information being passed into the next iteration, while a bigger value of tau will make the Hamiltonian dynamics too complex for rotations to extract information. The paper also did experiments on how the prediction’s mean squared error is affected by the value of tau, and found that a value of 0.2 gives the smallest error.

The resulting Group A qubits will be maintained and used together to train the neural network in the next iteration. In order to achieve that, we create an extra device with the same parameters and the same architecture to return the density matrix of the Group A qubits (instead of measuring expectation values). Therefore, we could use this density matrix to initialize the Group A qubits in the other device using *qml.QubitDensityMatrix()* so that we can train the QRNN with past information.

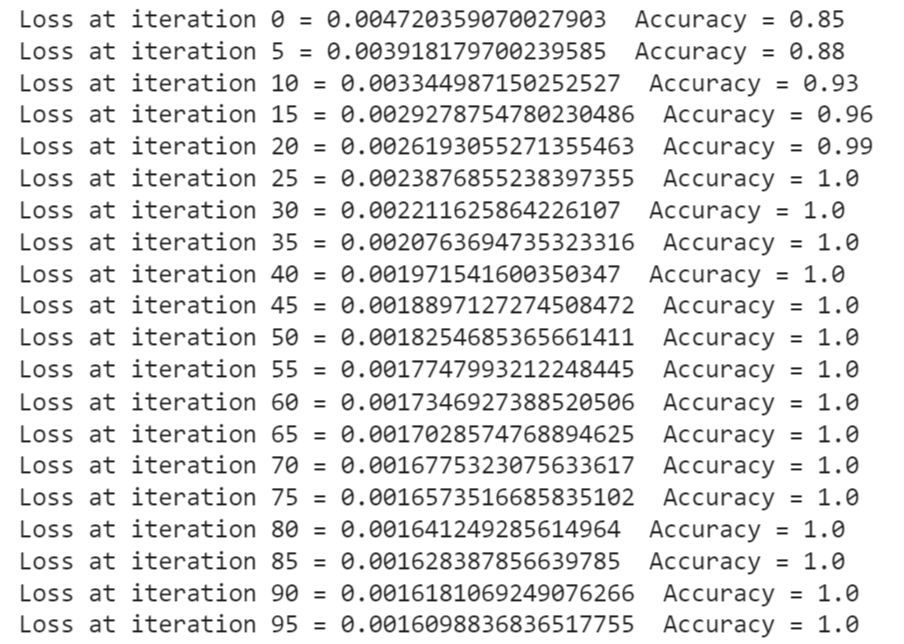


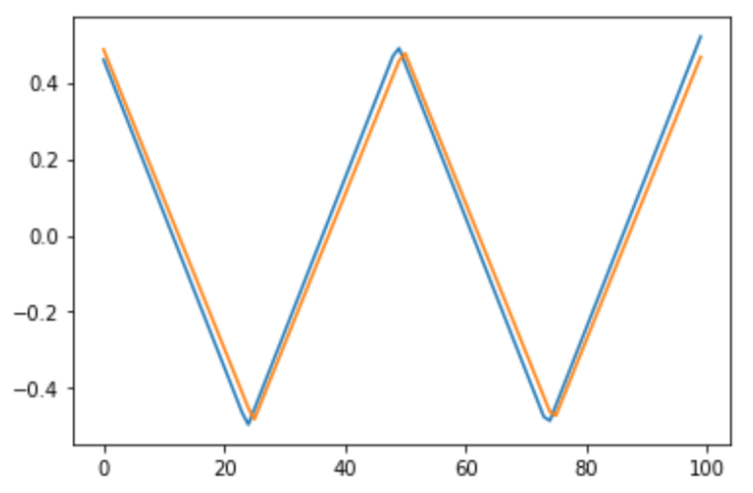
*Figure 3-6: Overall diagram of our software architecture (one iteration).*

# 4 Results of Software Implementation

We compute the accuracy by checking if the difference between the prediction and the true label is less than 0.1. An accuracy of 1.0 indicates that for all data points, the difference is less than 0.1.

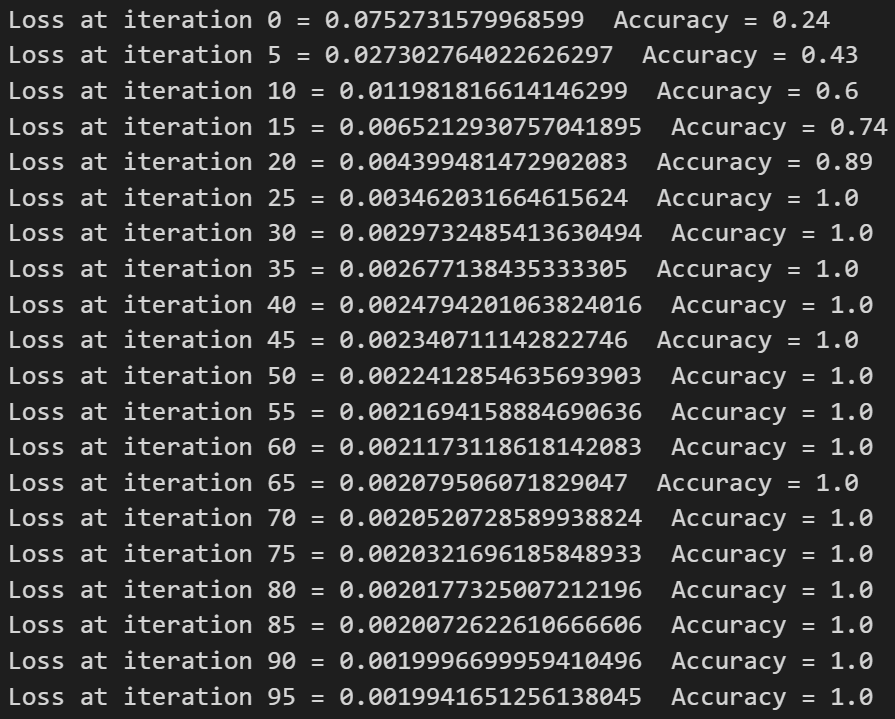
After training for 100 iterations on a triangular wave (blue line in Figure 4-1), we get a loss of 0.001609 and an accuracy of 1.0.

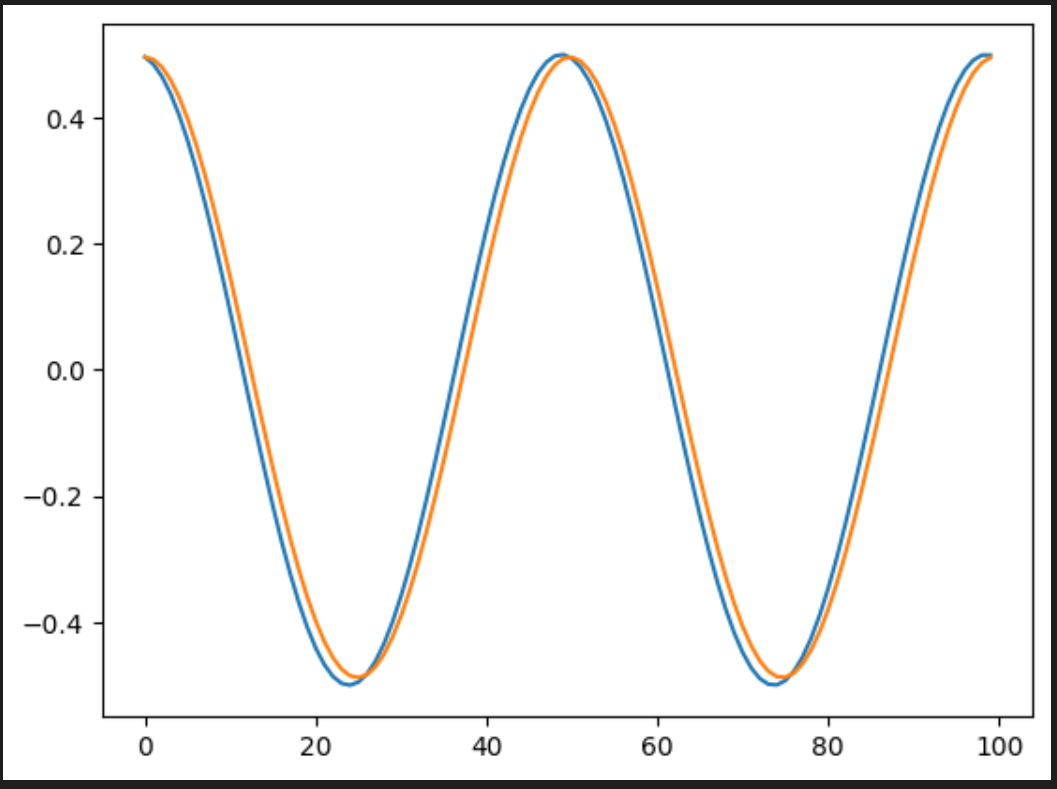




*Figure 4-1: Training loss and results after running 100 iterations on the triangular wave.*

After training for 100 iterations on a cosine wave (blue line in Figure 4-2), we get a loss of 0.001994 and an accuracy of 1.0.





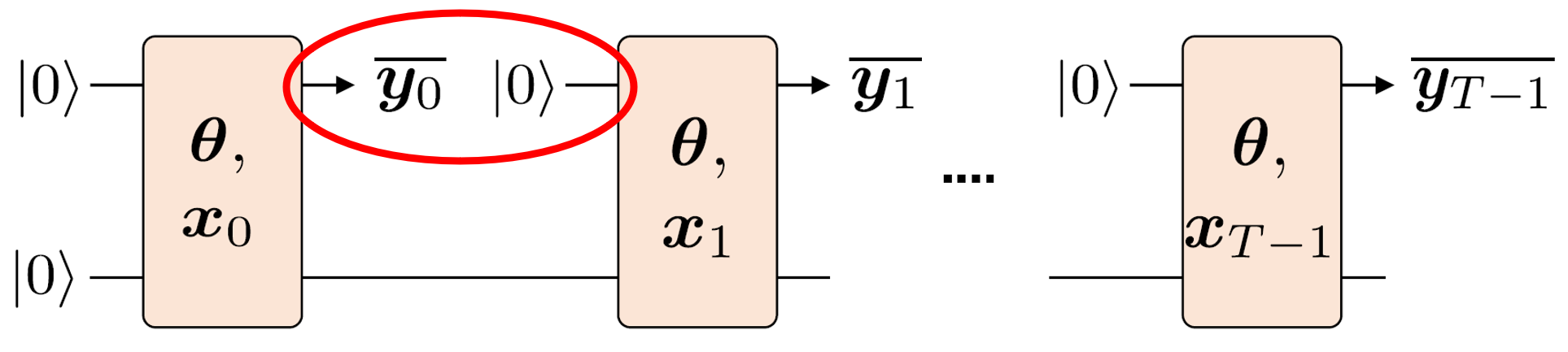
*Figure 4-2: Training loss and results after running 100 iterations on the cosine wave.*

# 5 Challenges affecting reproducibility

Ultimately, the results of the research paper were completely reproducible. During development, a few challenges affecting reproducibility appeared.

## 5.1 Challenge #1: Measure and Reset

After each iteration of the neural network, all the qubits in Group B (not the ones that save past information) must be reset to |0> (Figure 5-1 below).



*Figure 5-1: Measure and Reset of Group B qubits after each iteration.*

However, the rest of the qubits, which are part of Group A, must maintain their values and input them into the next iteration. The problem is that past information cannot be stored, since in each iteration, Pennylane will automatically reset all qubits to zeros.

Our initial idea was to create an extra set of 3 qubits for each iteration. Each of these sets would have their qubits just set to |0> before they are used in their corresponding iteration. This idea was discarded due to its heavy computational cost, because reproducing 100 iterations of training as detailed in the research paper would require (3)(100) = 300 extra qubits!

As recommended from Piazza discussions, we looked into using the “default.mixed” device with the function *qml.density\_matrix()* for returning the reduced density matrix that traces the Group A qubits only (and then using *qml.QubitDensityMatrix()* to maintain the state of the Group A qubits in the next iteration).

We settled on instantiating a total of 2 devices. Both devices are running the same circuit and functionality, but one device only measures the result of each iteration, whereas the other device only returns the density matrix of the Group A qubits, in order to instantiate the next iteration correctly for both devices.

## 5.2 Challenge #2: Computing Accuracy

Similar to the demo from Lecture 8, we have a helper function that computes accuracy for using *qml.GradientDescentOptimizer()*. However, our predictions find numerical values (not categorical). This led to the realization that predictions should not be penalized heavily if they are close to true values, but do not exactly match.

Therefore, we modified our accuracy computation to instead use the absolute difference between predicted values and true values.

## 5.3 Challenge #3: Using a Hamiltonian

When the code was being developed, we had not yet learned about Hamiltonians in class. After discussions on Piazza, we learned to use *qml.Hamiltonian*() to form Hamiltonian and pass it to *qml.ApproxTimeEvolution()* to implement the evolution step of each iteration.

# 6 Choice of Programming Framework

We chose Pennylane due to all team members being most familiar with this framework, and that we successfully implemented the measure and reset functionality within Pennylane, without having to switch to another framework.

# 7 Conclusion (Bobby)

Summarize what was mentioned in this report

* Remember to mention QRC again like the paper

Copy content from slides

* Limitations and Open Questions
* Future directions of exploration