Quantum Computing based Generative Adversarial Network for Time-Series forecasting

Aadittya Tiwari[1]
[1]Lynbrook High School Math & Computer Science Department

Link to the paper

Project Motivation

- Generative ML models were relatively new (Project started January, ChatGPT launched November 30)

- GANs produce synthetic data, which could be used for predicting data, on the idea that future data is modeled after historical data and synthetic data is also modeled after historical data

- Stock price prediction is a lucrative field, and is a great example of time-series forecasting
    - It’s easy to objectively measure model performance with numerical outputs instead of image outputs, where you would need subjective criteria to judge performance

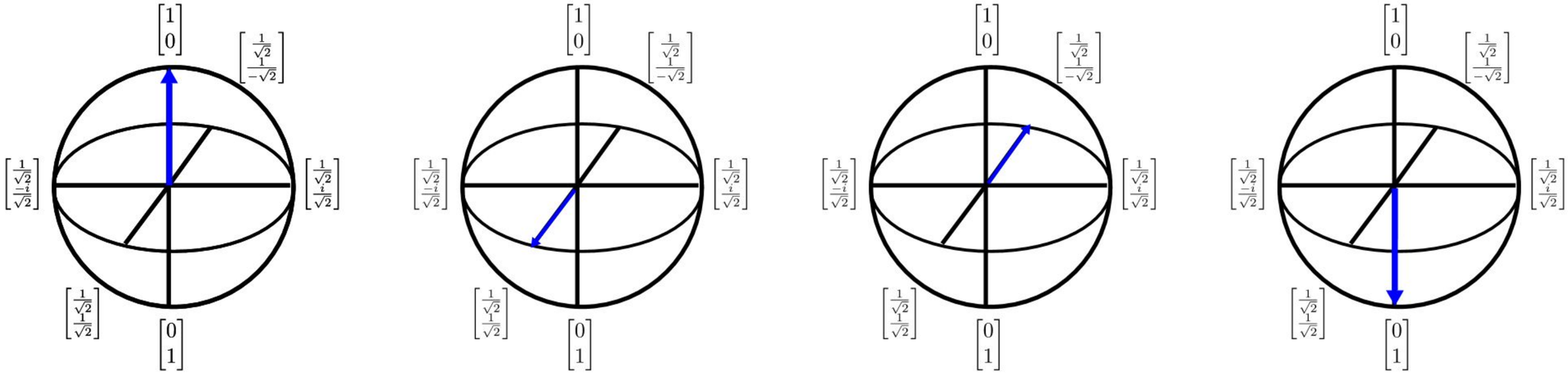
- Led to me being interested in researching the effects of incorporating Quantum Computing algorithms in GANs

Proposed project:

A Generative Adversarial Network replacing the generator with a Quantum Machine Learning model for time-series forecasting.

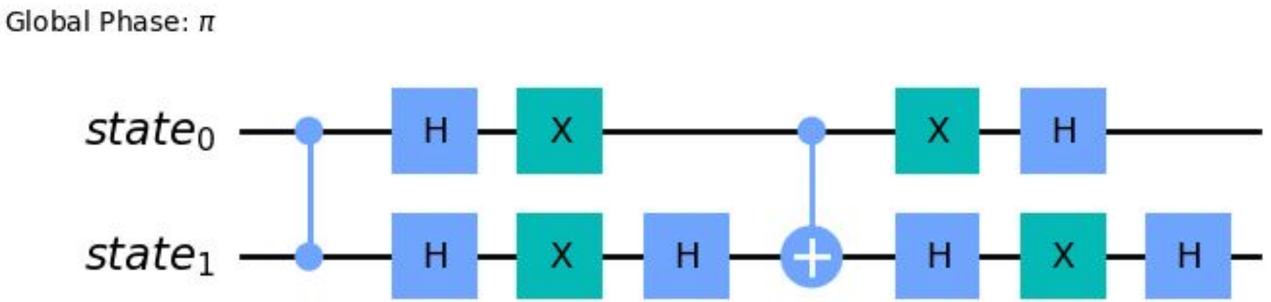
Quantum Computing Prerequisites

- Classical computing uses bits, which can be either 0’s or 1’s
- Quantum computing uses Qubits, which can be in a superposition, or in multiple, states at once - not definitely 0 or 1 entirely until measured, where it collapses to one state
  - State of a qubit is mathematically represented through the equation $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, where $|\alpha|^2$ & $|\beta|^2$ represent the probability of the Qubit being measured as 0 or 1, respectively



Quantum Computing Prerequisites

- Quantum Gates are the counterparts to classical Logic gates, both are used to perform operations on bits/qubits
  - Quantum gates change the probability of a qubit (or multiple qubits affected by a gate) to collapse to either 0 or 1
- A conjunction of Quantum logic gates lead to Quantum algorithms, which is what we will be using to write QML models

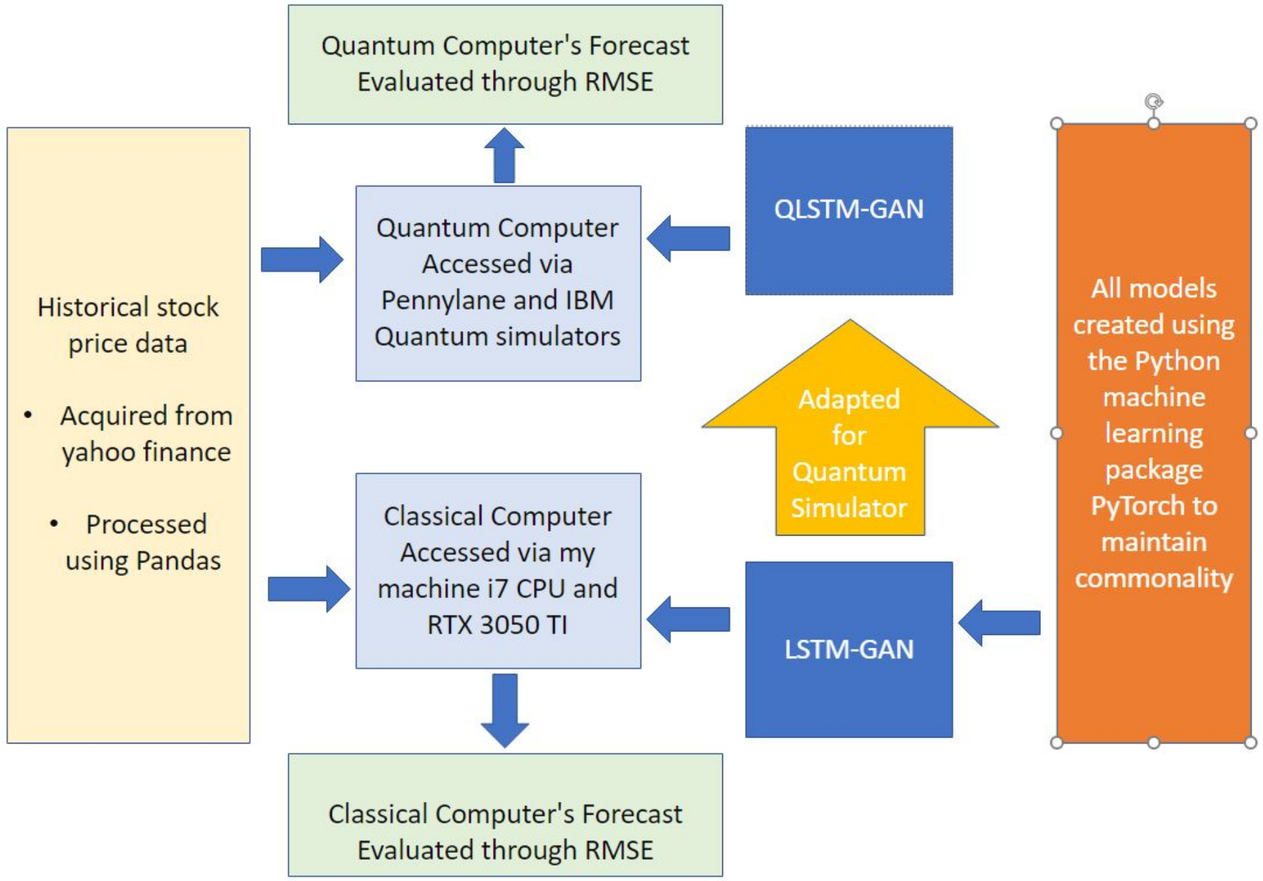


Visualization of a phase-flip oracle for Grover's algorithm. Composed of Hadarmard gates (H), Control Not (+ and line), and Control X gates (X). The oracle is later used in the full algorithm multiply "good states" that you are searching for by -1

Quantum Computing Prerequisites: Pennylane

- Framework for Quantum Machine Learning
- Easy to “combine” with Pytorch and Tensorflow, for hybrid Quantum-Classical Machine learning
- You write models using

Outline of the project

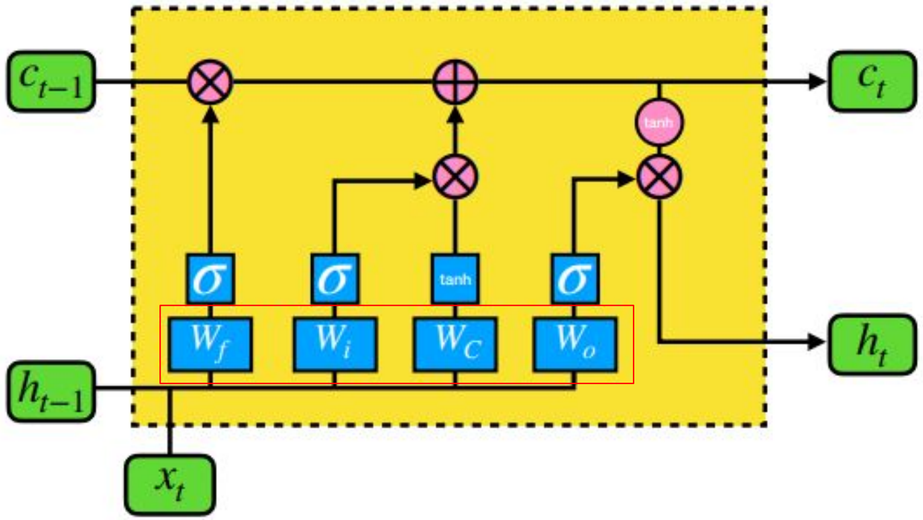


- Determine and develop a Quantum ML model to be used a generator for the GAN
- Use an existing GAN for time-series forecasting and replace the generator with the aforementioned model
- Select 3 stocks with differing trends to use as control metrics
- Train the Quantum and Classical GANs on the stocks and compare their performances through RMSE of predictions, Epochs to converge, and parameter count

LSTM basics

- Quantum LSTM, or QLSTM, is the neural network model that we will be using later on to make our GAN a Quantum GAN

- Classical LSTM has the layers input, update, forget, output
  - These four layers then comprise the circuit for the model to retain important information and forget useless information whilst training



Information flow in an LSTM circuit visualized

The information flow in a classical LSTM cell (Figure 2) is

$$f_t = \sigma\left(W_f \cdot v_t + b_f\right), \tag{1a}$$

$$i_t = \sigma\left(W_i \cdot v_t + b_i\right), \tag{1b}$$

$$\tilde{C}_t = \tanh\left(W_C \cdot v_t + b_C\right), \tag{1c}$$

$$c_t = f_t * c_{t-1} + i_t * \tilde{C}_t, \tag{1d}$$

$$o_t = \sigma\left(W_o \cdot v_t + b_o\right), \tag{1e}$$

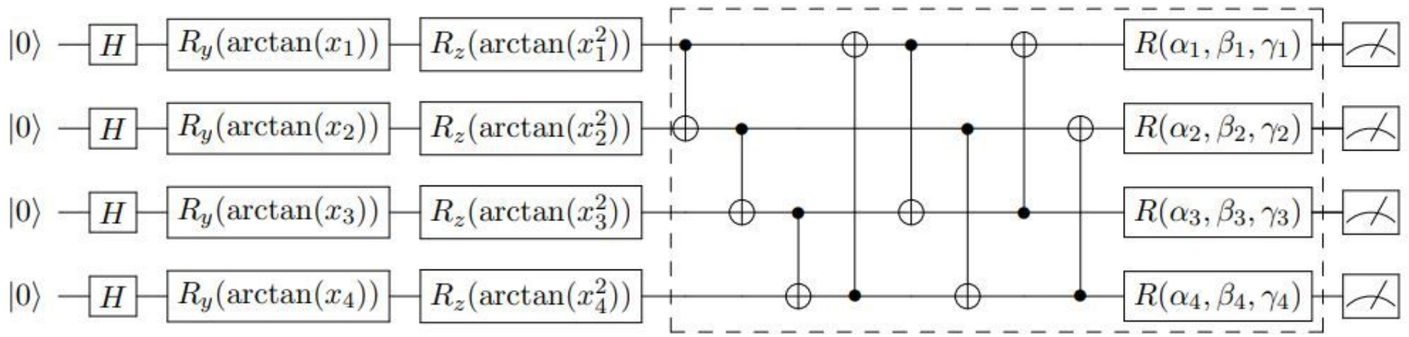
$$h_t = o_t * \tanh\left(c_t\right), \tag{1f}$$

The math behind it, using sigmoid and tanh gates

Quantum LSTM

- By replacing these layers with Variational Quantum Circuits, we have successfully made a QLSTM
  - Variational Quantum Circuits: Classical Neural Networks parallels, built with Parameterized Quantum Circuits, which are built with Ansatzes, which are parallels to neural network layers

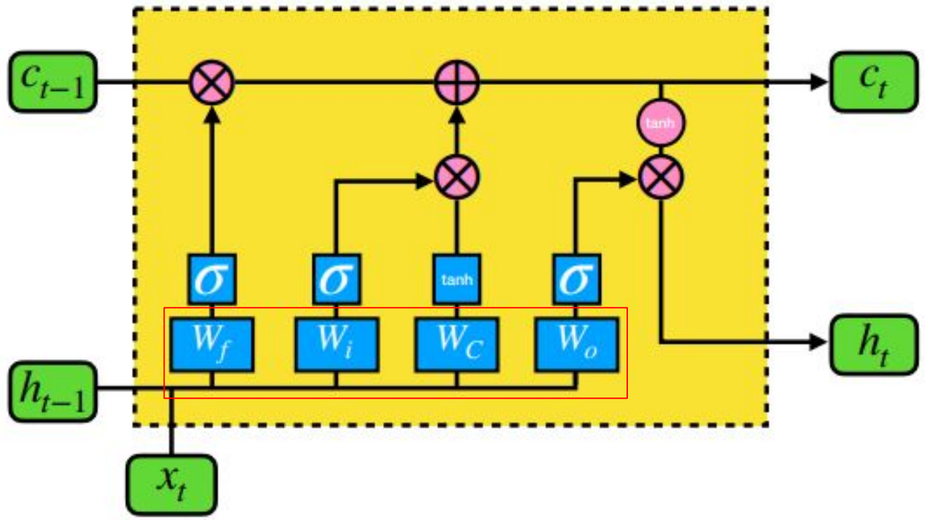
- The advantage this QLSTM now has over the classical LSTM is documented in this paper, basically researchers found that QLSTM converges much quicker than LSTMs, require less parameters, but LSTMs can perform just as well as QLSTM's at forecasting tasks if both are trained for sufficiently large epochs

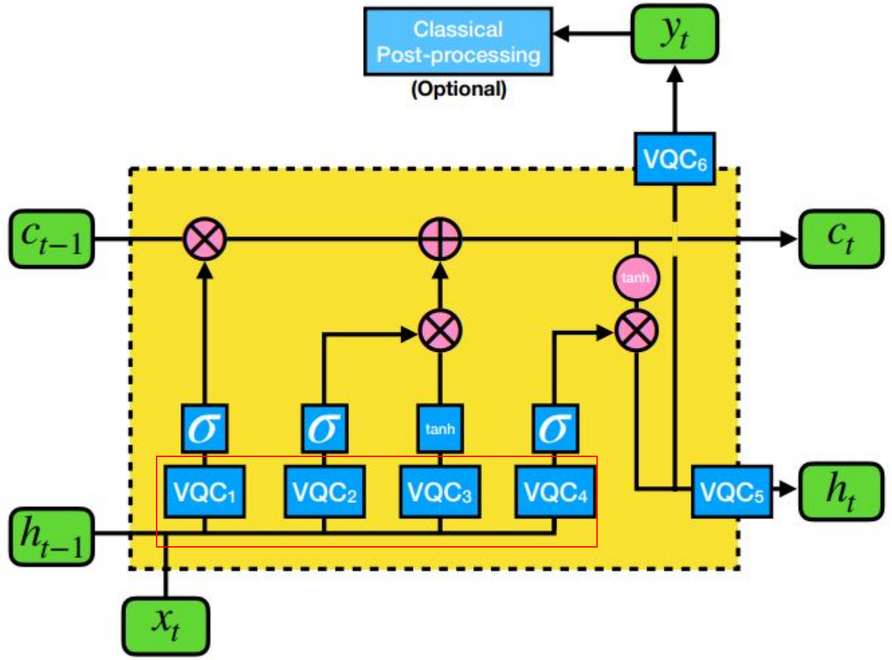


VQC comprised of Hadamard gates, Rotational gates for parameters, and Control Nots.
First segment is for data encoding, second segment is the variational layer
This is the VQC we will be using in our QSLTM

QLSTM visualized

All that changed was the Classical Neural Networks, denoted by the Ws, have been replaced by VQCs.

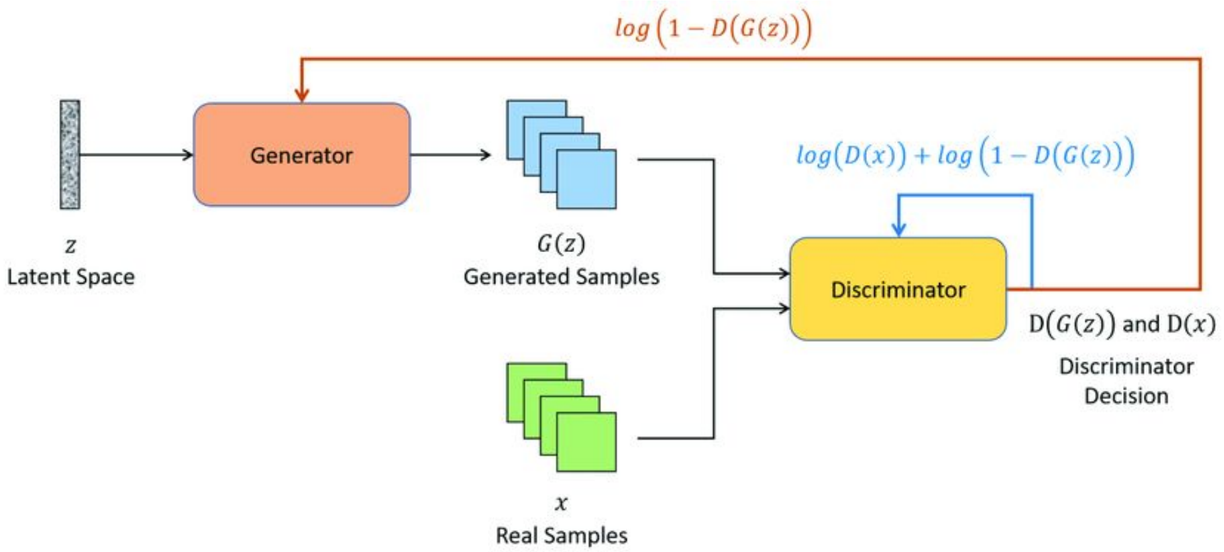


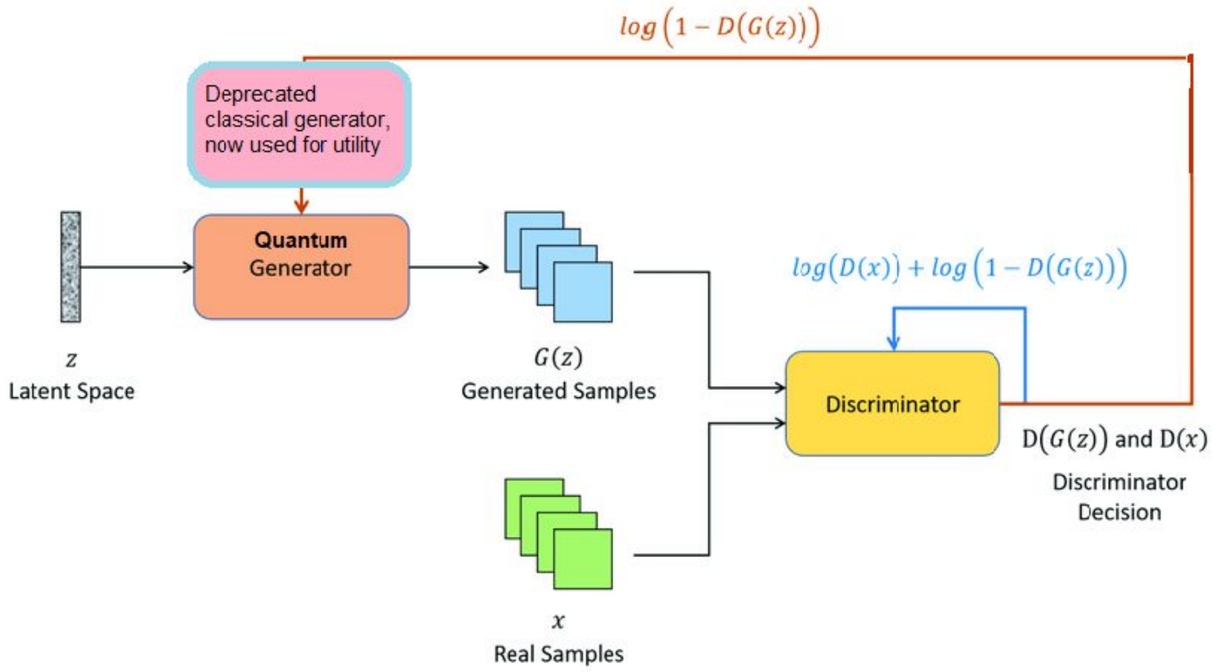


Creating the Quantum GAN

- Implemented a QLSTM I had previously used in another time-series forecasting project by replacing the classical LSTM generator with the QLSTM generator

Implementing the Quantum Generator was the hardest of the project taking 30+ hours, despite it seeming like just copy pasting the generator. Many errors were caused, but could be summarized by trying to use a PyTorch generator in a Tensorflow GAN architecture, and then bugs with error messages that I had 0 experience with arose, played around with using ChatGPT to debug, and ended up becoming a huge mess





Apple ^

Intel V

Synopsys ->

Stock Selection

- Very simple, basically just selecting 3 different stocks with differing historical trends in the last 2 years
- Trends that were important to have were stocks which oscillated between lows and highs, stagnant growth, and affected by Covid-19 for rare event effects
- Finalized on Apple, Intel, and Synopsys

I was not paid by any of these companies to select their stocks!

5.2 Results

The following data tables show the performance of each model across the evaluation metrics defined above:

RMSE	**CGAN**	**QGAN**
Intel prediction	7.11	13.10
Apple prediction	7.97	3.61
Synopsys prediction	4.77	8.50

Epochs taken to reach convergence	
QGAN	6 out of 15 total epochs
CGAN	82 out of 100 total epochs

Total parameters (Generator + Discriminator)	
QGAN	233+6199041
CGAN	109429+6199041

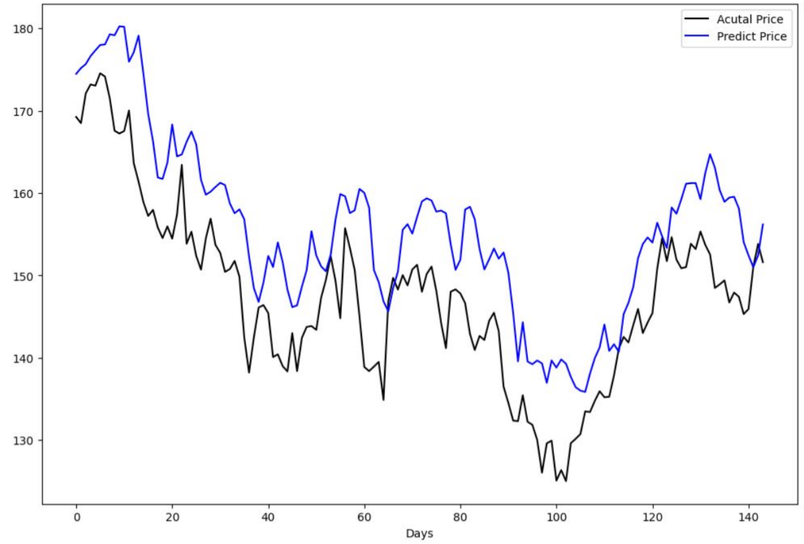
Performance Evaluation

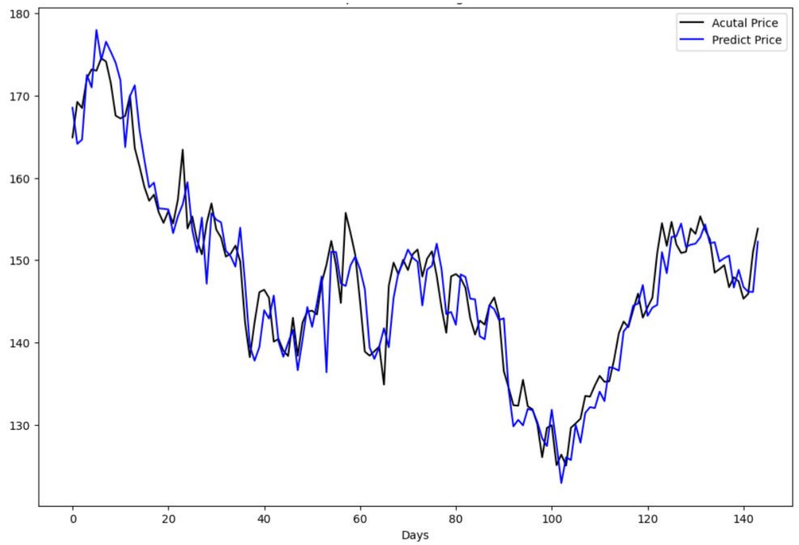
- Metrics to evaluate performance were RMSE, Epochs to Converge, and total parameter count of the entire model (Generator + Discriminator
- Working backwards, parameter-wise the QGAN had significantly less parameters, the bulk of its parameters were from the classical discriminator, a Convolutional Neural Network.
- Epochs to converge was also considerably less for the QGAN when compared to the CGAN, as a result of a QML model to learn information faster
- However, RMSE wise it is somewhat hard to draw a definitive conclusion on which model is superior. However, when looking at the graphs, it becomes clear.

Classical

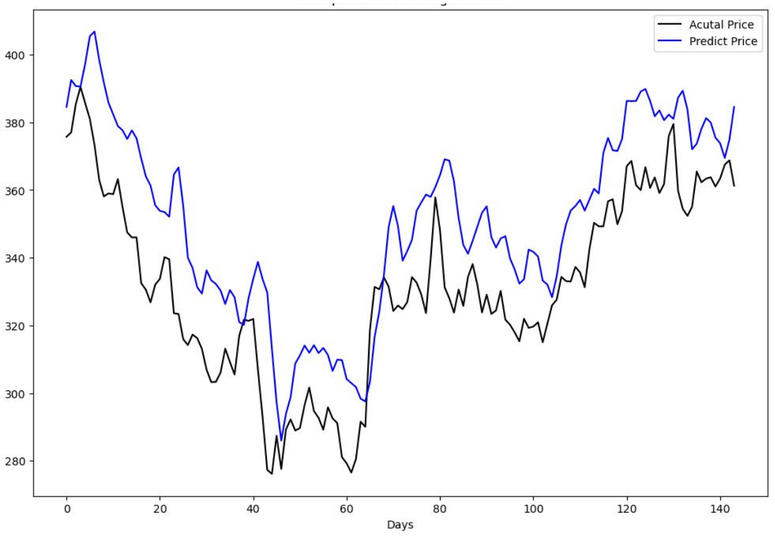
Quantum

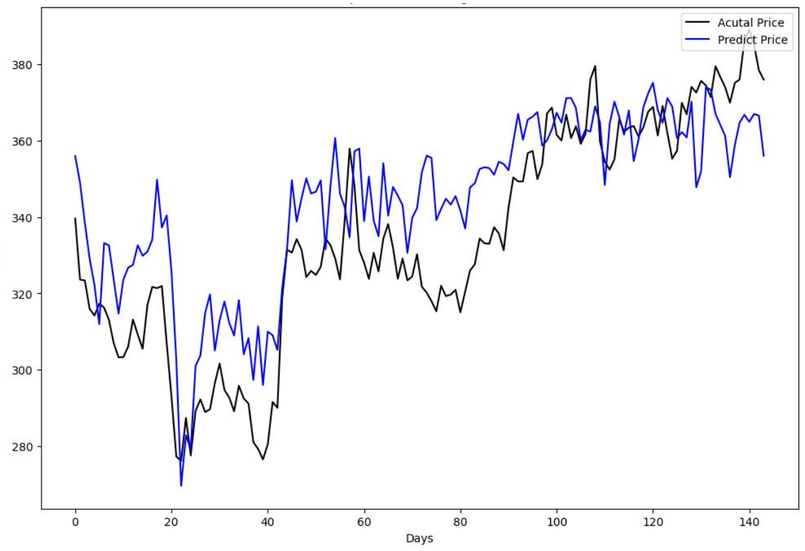
Apple





Synopsys





- The Quantum RMSE is off due to fluctuations that are “out-of-sync” with the actual graph, but still hugging tight to the graph. The classical RMSE is close but still not perfect, due to being “in-sync” with fluctuations but shifted upwards or downwards by a constant amount
- While both models are not the best at time-series forecasting, it at least shows that QGANs are not able to be completely ruled out as objectively worse than Classical GANs

Project Summary

- Quantum GANs are applicable in time-series forecasting and can have comparable prediction accuracies when contrasted with Classical GANs
- Outperforms Classical GANs in terms of parameter count and epochs to converge
- When ran on a Quantum computer SIMULATOR instead of a real machine, it takes considerably longer to train a QGAN than a CGAN

Project Summary continued

- Using the Z space generated from a Quantum GAN would *probably* yield results similar to using one from a classical GAN

- Project could be taken one step further by using different models as generators (e.g GRU, LSTM, Quantum GRU, Quantum LSTM, etc)

- Might be easier to just skip the step of using a z-space and just let the GAN fully run until the final output is made, and use that final output