Quantitative Financial Portfolio Optimization using Quantum Approximation

A PROJECT REPORT

Submitted by

Vijayalakshmi S 211420104304

in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

PANIMALAR ENGINEERING COLLEGE

(An Autonomous Institution, Affiliated to Anna University, Chennai)

APRIL 2024

PANIMALAR ENGINEERING COLLEGE

(An Autonomous Institution, Affiliated to Anna University, Chennai)

BONAFIDE CERTIFICATE

Certified that this project report **"Quantitative Financial Portfolio Optimization using Quantum Approximation"** is the bonafide work of **Vijayalakshmi S (211420104304)** who carried out the project work under my supervision.

Signature of the HOD with date	**Signature of the Supervisor with date**
Dr.L. JABASHEELA, M.E., Ph.D., **PROFESSOR** **HEAD OF THE DEPARTMENT**	**Dr. K. VALARMATHI, M.E., Ph.D.,** **SUPERVISOR** **PROFESSOR**
Department Of Computer Science and Engineering, Panimalar Engineering College, Chennai - 600 123.	Department Of Computer Science and Engineering, Panimalar Engineering College, Chennai - 600 123.

Certified that the above candidate was examined in the End Semester Project Viva-Voce Examination held on...........................

INTERNAL EXAMINER **EXTERNAL EXAMINER**

DECLARATION BY THE STUDENT

I **Vijayalakshmi S (211420104304)** hereby declare that this project report titled "**Quantitative Financial Portfolio Optimization using Quantum Approximation**", under the guidance of **Dr. K. Valarmathi, M.E., Ph.D.,** is the original work done by me and I have not plagiarized or submitted to any other degree in any university by me.

1. Vijayalakshmi S

ACKNOWLEDGEMENT

My profound gratitude is directed towards our esteemed Secretary and Correspondent, **Dr. P. CHINNADURAI, M.A., Ph.D**., for his fervent encouragement. His inspirational support proved instrumental in galvanizing our efforts, ultimately contributing significantly to the successful completion of this project

I want to express our deep gratitude to our Directors, **Tmt. C. VIJAYARAJESWARI, Dr. C. SAKTHI KUMAR, M.E., Ph.D., and Dr. SARANYASREE SAKTHI KUMAR, B.E., M.B.A., Ph.D.,** for graciously affording me the essential resources and facilities for undertaking of this project.

My gratitude is also extended to our Principal, **Dr. K. MANI, M.E., Ph.D.,** whose facilitation proved pivotal in the successful completion of this project.

I express my heartfelt thanks to **Dr. L. JABASHEELA, M.E., Ph.D.,** Head of the Department of Computer Science and Engineering, for granting the necessary facilities that contributed to the timely and successful completion of project.

I would like to express my sincere thanks to Project Guide and Project Coordinator **Dr. K. VALARMATHI M.E., Ph.D.,** and all the faculty members of the Department of CSE for their advice and encouragement for the successful completion of the project.

VIJAYALAKSHMI S (2114210104304)

ABSTRACT

Portfolio optimization is a primary component of the decision-making process in finance, aiming to tactfully allocate assets to achieve optimal returns while considering various constraints.

It is an optimization problem where we have a collection of assets and we want to select these assets that maximize our return but at the same time minimize the risk.

These optimization problems can be formulated as quadratic programs which are well studied classically and are very difficult to solve.

Quantum Mechanics principles native to Quantum Computing, make it suitable for implementing financial systems, also inline with business objectives; reducing execution time adding to higher returns.

The proposed system makes use of Quantum Approximate Optimization Algorithm to take in the input stock data from various portfolios, update the weights and asses the risks to find the optimum financial portfolio.

Thus, a system based on quantum approximation is used for finding a solution for quantitative financial portfolios.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

1.1 OVERVIEW

Financial institutions are testing early use-cases of Quantum Technologies for NP-hard problems which are uncertain or difficult to optimize. In this article, we are going to make use of quantum computers for building an optimal portfolio out of stocks using the mean-variance portfolio optimization technique.

Financial Systems are complex, strongly correlated and difficult to predict, full of optimization problems, Monte Carlo sampling, stochastic differential equations, classical machine learning and more.

Portfolio Optimization is one of the crucial financial problems that needs to be addressed to invest in the market. Classical machine learning approaches and Non-machine learning approaches have been employed to solve this problem.

Still, the efficiency and throughput of these systems can be improved upon by using quantum approach to solve the problem.

1.2 PROBLEM DEFINITION

Financial portfolio optimization is the problem of optimal allocation of a fixed budget to a collection of assets (commodities, bonds, securities etc.) which produces random returns over time.

The problem of portfolio optimization can be represented as finding a way to minimize a given cost function that is related to the risk taken when choosing an asset. Thus, the weight associated with each asset needs to be updated in order to obtain a portfolio that has a low risk but with good benefits too.

By the nature of the problem we are dealing with, it is indeed possible to implement a portfolio optimization solution using quantum computing. Quantum computing tends to use the properties of particles in order to perform different kinds of computations applied to specific problems.

To do so, quantum computers use what we call qubits or quantum bits of information that obey the properties of quantum mechanics. This offers a new way of thinking about computation using superposition or entanglement. However, to obtain a result in the end, these qubits need to be measured so that we have a classical result in the end. This result, like classical computing, when considering a single bit of information, is either 0 or 1.
This way of calculations obviously needs a different theoretical frame in order to be put at its advantage. Thus, we adapt the equation accordingly.

1.3 SCOPE OF THE PROJECT

The Quantum Approximate Optimization Algorithm (QAOA) is a widely-studied method for solving combinatorial optimization problems on NISQ (Noisy Intermediate-Scale Quantum) devices. The applications of QAOA are broad and far-reaching, and the performance of the algorithm is of great interest to the quantum computing research community.

1.4 OBJECTIVE

The goal of this process is to essentially ensure that one can acquire the highest amounts of profits with reasonable risk tolerance.
The input given are uniform random historical price data with budget and risk tolerance markers.
The output is a portfolio representing a list of investments and the expected returns.

CHAPTER 2
LITERATURE SURVEY

2.1 LITERATURE SURVEY

S. No.	Year	Title	Methodology	Limitations
1.	2023	A Deep Neural Network Algorithm for Linear-Quadratic Portfolio Optimization With MGARCH and Small Transaction Costs	Reinforcement Learning using multivariate generalized autoregressive conditional-heteroskedasticity(MGARCH)	Applicable only for small transactions. Scaled down feature dimensions.
2.	2024	Empirical Analysis of Quantum Approximate Optimization Algorithm for Knapsack-based Financial Portfolio Optimization	Knapsack Algorithm used to find complexity and QAOA is used to find the optimal solution	Model yields only 50% result in noisy conditions.
3.	2023	Quantum-inspired Computing: Entanglement-enhanced Technique for Short Portfolio in Global Markets	QIO based portfolio optimization	Model unstable with higher data input.
4.	2023	Reinforcement Learning for Stock Prediction and High-Frequency Trading	Classical learning approach using inverse reinforcement learning,	Higher execution time.

		With T+1 Rules	and multi-armed bandit learning.	
5.	2023	Trend Ratio-based Portfolio Optimization Model Adopting Entanglement-enhanced Quantum-inspired Evolutionary Computation in the Global Financial Markets	Quantum Inspired Optimization using Local and Global Search	Only finds critical areas in trends, limited application over dataset.
6.	2023	Real-Time Portfolio Management System Utilizing Machine Learning Techniques	Classical approach using K-means algorithm and metaheuristics algorithm	Model can not handle high dimension feature selection
7.	2023	Deep reinforcement learning for stock portfolio optimization by connecting with modern portfolio theory	Classical approach using reinforcement learning and modern portfolio theory	Higher execution time.
8.	2023	Multi-qubit quantum computing using discrete-time quantum walks on closed graphs	Quantum walk model based on Grover's algorithm	Applicable only for closed graphs
9.	2023	Application of quantum computing in discrete portfolio optimization	Quantum Walk Optimization Algorithm used for	Can not handle high dimension feature selection.

			portfolio optimization	
10.	2021	Quantum Optimization Heuristics with an Application to Knapsack Problems	QAOA used for finding greedy solutions	QAOA used for finding greedy solutions
11.	2021	PyPortfolioOpt: portfolio optimization in Python	Portfolio optimization using classical mean-variance optimization	Low throughput and high execution time
12.	2020	OSQP: an operator splitting solver for quadratic programs	Quadratic conversion of solver	Not applied to financial portfolios
13.	2019	Improving Variational Quantum Optimization using CVaR	Combinational Optimization using Variational Quantum Eigen solver	Not applied to financial portfolios.
14.	2019	Quantum computing for finance: overview and prospects	Research on QML techniques against classical methods	-
15.	2015	A Quantum Approximate Optimization Algorithm Applied to a Bounded Occurrence Constraint Problem	QAOA	Not applied for portfolio optimization.
16.	2011	Monte Carlo estimation of value-at-risk, conditional value-at-risk and their sensitivities	Classical approach using Monte Carlo simulation	Limited application

Fig 2.1.1 Comparison of Existing Research work

CHAPTER 3
THEORETICAL BACKGROUND

3.1 EXISTING SYSTEM

Investors in multiple fields such as banks, artmobiles, etc., need to conduct portfolio optimization and use methods like:

- **GRG Nonlinear Solver:**

  GRG stands for “Generalized Reduced Gradient.” Solver is a Microsoft Excel add-in program; can find an optimal (maximum or minimum) value for a formula in one cell — called the objective cell — subject to constraints, or limits, on the values of other formula cells on a worksheet.

- **Classical Machine Learning:**

  Using the Efficient Frontier Theory formulated by Harry Markowitz in 1952, an intelligent set of smart models for portfolio optimization were constructed.

  These models process the quantitative data inputs, analyze them, and produce an efficient allocation of capital.

DRAWBACKS:

GRG Nonlinear Solver:

- Highly dependent on initial condition.
- Solution obtained may not be the global optimum.
- Most likely to return the local optimum value nearest to the initial conditions.

Classical Machine Learning:

- Large datasets are common and are difficult to interpret.
- Higher execution time.

3.2 PROPOSED SYSTEM

- The problem of portfolio optimization can be represented as finding a way to minimize a given cost function that is related to the risk taken when choosing an asset.
- The proposed system thus updates the weight associated with each asset in order to obtain a portfolio that has lower risk with high gain.
- Using **Quantum Approximate Optimization Algorithm** (QAOA), we solve the combinatorial problem of handling vast feature dimension and obtain optimum solution against various constraints.

ADVANTAGES:

- High Performance Computing:
  The high speed processing power helps reduce the execution time required to process the data.
- Effective Handling of Large Datasets:
  It is possible to process datasets with high dimensions in the feature space and huge amounts of data.
- Acquire Optimal Return:
  It helps achieve the optimal returns against various constraints unlike existing systems which are likely to return local optimum.

3.3 DATASET DESCRIPTION:

The dataset consists of historical (2015 – 2020) trading data for 5 stocks from US Exchange Market:

- IBM (IT Industry)

- Pfizer (Healthcare / Pharmacy)
- Exxon Mobil Corp. (Oil & Gas)
- Bank of America (Finance / Banking)
- Tesla (Automobile / Technology)

In order to build a portfolio optimizer, 5 assets for stock data are based on lesser correlation.

3.4 DFD DIAGRAM

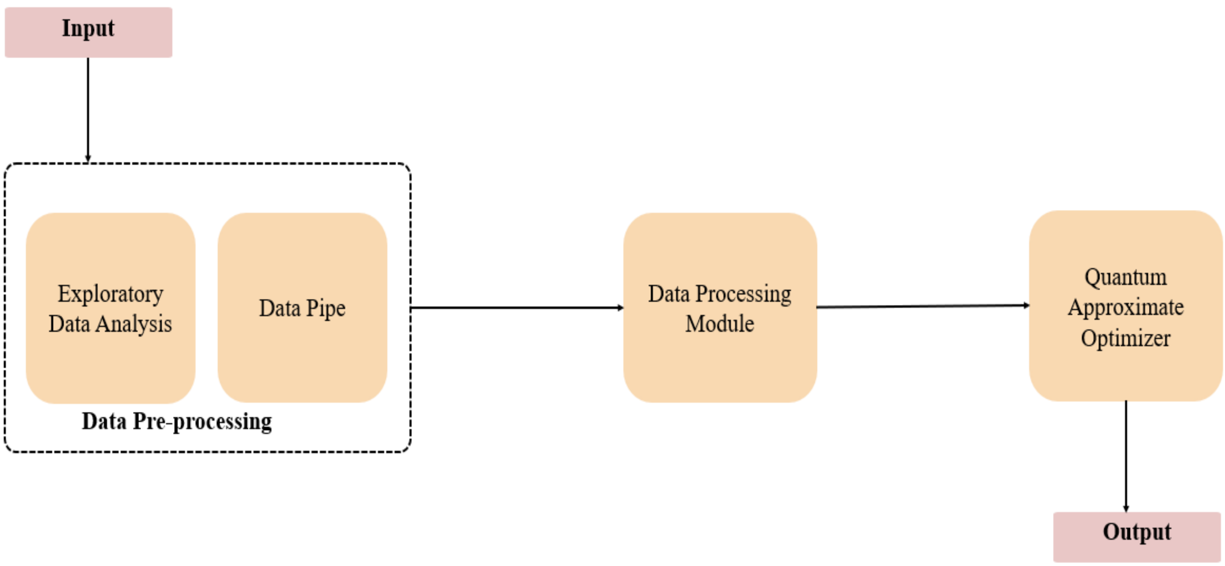

Fig 3.4.1 DFD Diagram

3.5 UML DIAGRAMS

3.5.1 USE CASE DIAGRAM

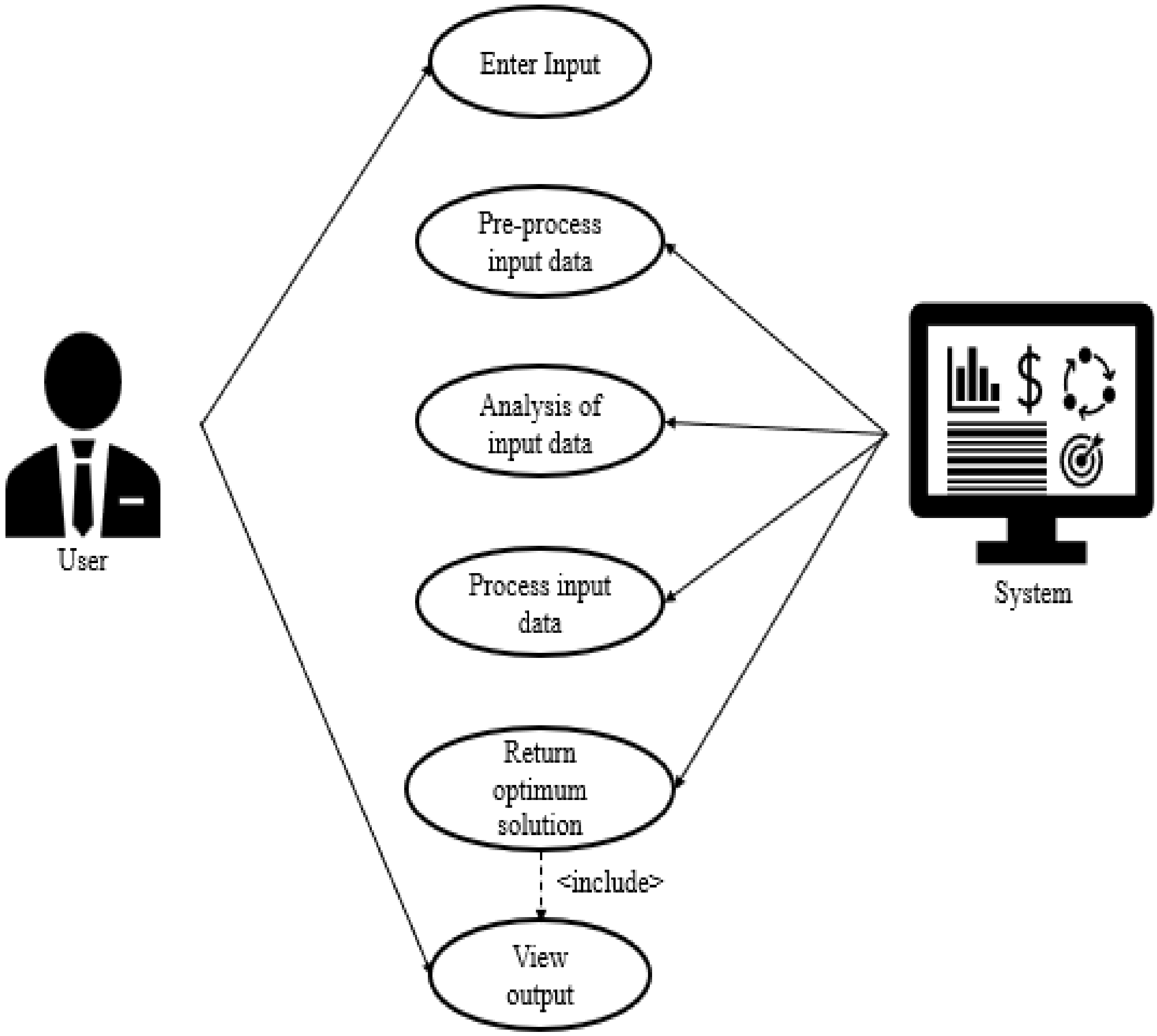

Fig 3.5.1.1 Use case diagram

3.6 ALGORITHMS USED

3.6.1 Portfolio Expected Return

The expected return of a portfolio is calculated by multiplying the weight of the asset by its return and summing the values of all the assets together. To introduce a forward looking estimate, probability may be introduced to generate and incorporate features in business and economy.

Expected Return, $E\left(R_{\rho}\right) = \sum w_i E(R_i)$

3.6.2 Portfolio Variance

Portfolio variance is used as the measure of risk in this model. A higher variance will indicate a higher risk for the asset class and the portfolio. The formula is expressed as;

Portfolio Variance, $\sigma_{\rho}^{2} = \sum_{i} w_i^2 \sigma_i^2 + \sum_{i} \sum_{j \neq i} w_i w_j \sigma_i \sigma_j \rho_{ij}$

3.6.3 Quantum Approximation Optimization Algorithm

IBM Qiskit is a python library that provides with quantum algorithms like QAOA which maps the problem to a Hamiltonian whose ground state corresponds to the optimal solution.

The unitary operator $U_C(\gamma)$ refers to the cost layer and $U_M(\beta)$ to the mixer layer. The first encodes the Hamiltonian of the problem with the cost function and the optimization problem.

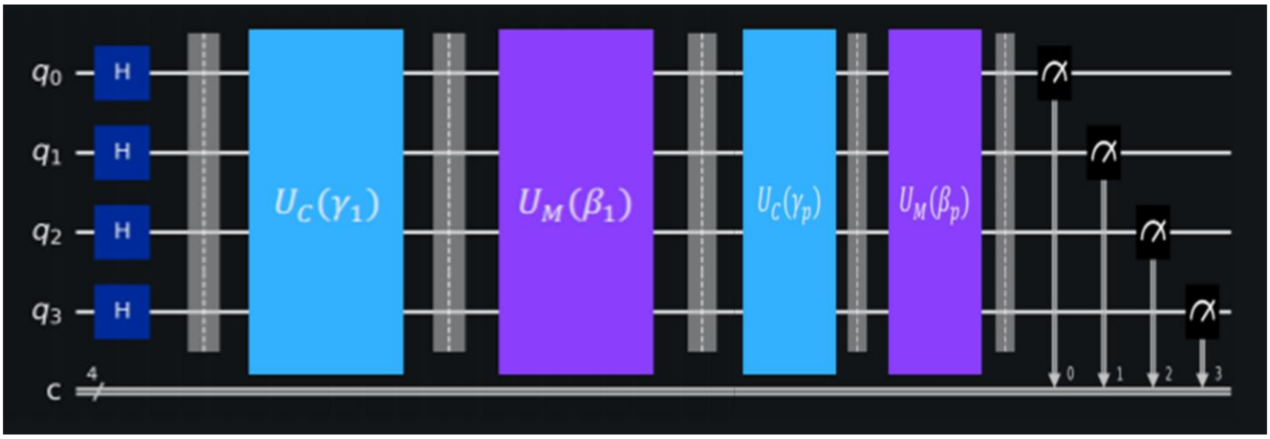

Fig 3.6.3.1 QAOA Circuit

The second mixes the results in order to make the assets that are valuable appear more often and thus they will come out more often after the measurement.

In the formalism that we use to solve this problem, we are given the following function to compute.

$$qx^T \Sigma x - \mu^T x$$

Respect the following condition: $1^T x = B$

With the following notation:

- $x \in \{0,1\}^n$ denoting the vector of binary decision variables which indicates the assets to pick $x[i] = 1$ or not $x[i] = 0$
- $\mu \in R^n$ the expected return for the assets
- $\Sigma \in R^{n \times n}$ the covariance between the assets
- $q > 0$ controls the risk appetite of the decision maker
- B describing the budget or the number of assets to be selected among the possibilities

We need to have the simplification that all assets have the same price, meaning a normalization factor between the assets and the full budget that has to be spent so all chosen assets will either be part of the portfolio or not as the result of the calculation. Thus, a constraint that follows is that the assets all have the same weight in the portfolio. The problem is to choose among the list that we have which assets are the best to pick in order to have the best profits.

CHAPTER 4
SYSTEM ARCHITECTURE

4.1 ARCHITECTURE OVERVIEW

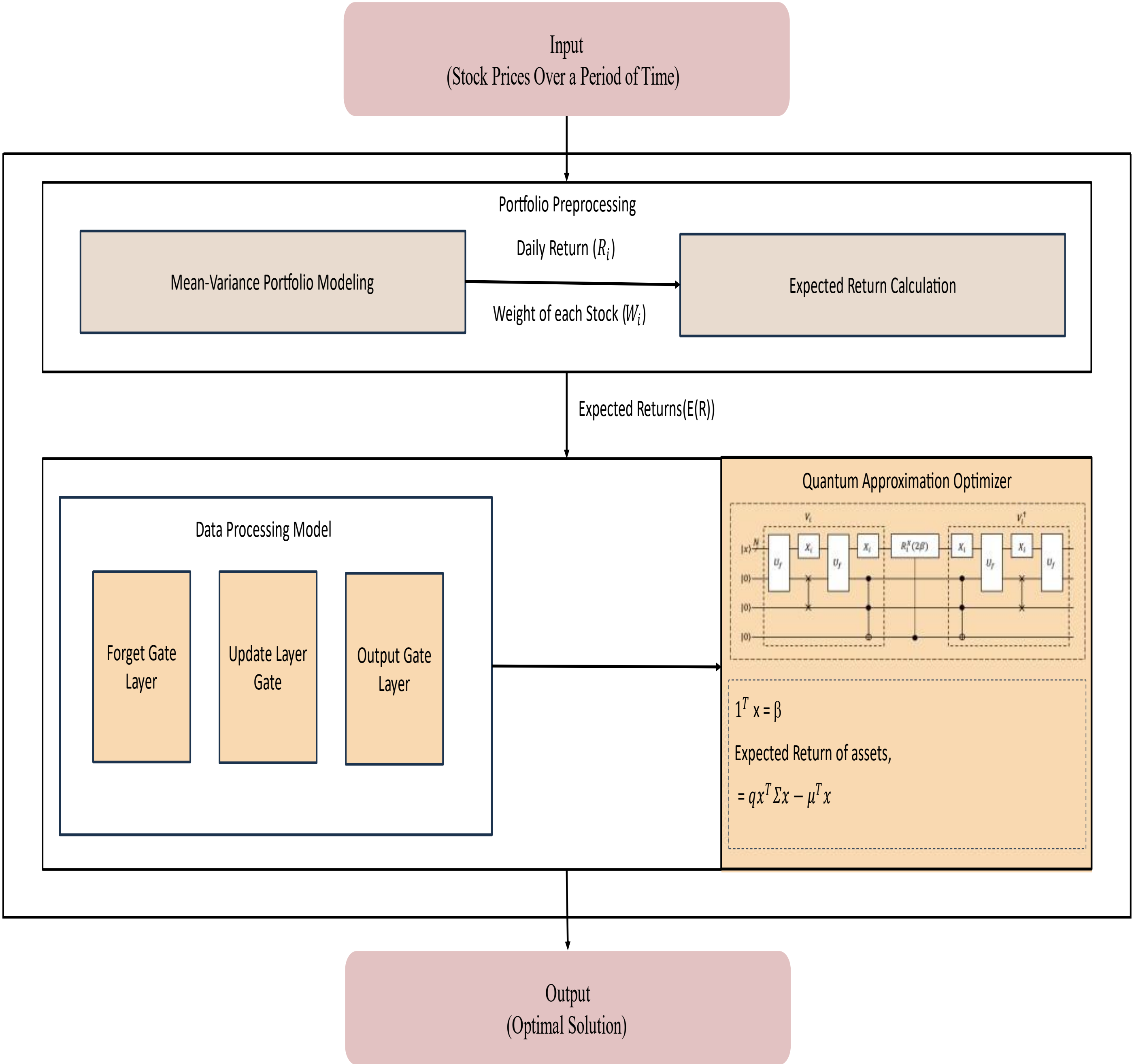

Fig 4.1 Architecture diagram

4.2 MODULE DESCRIPTION

Data Pre-processing Module:

- The basic information is explored.
- The data is checked for null values and cleaned if present.
- The tickers are set for the portfolios of Tesla, Bank of America Corporation (BAC), International Business Machines (IBM), Exxon Mobil Corp. (XOM), and Pfizer Inc. (PFE).
- For each stock, the input is a raw time series of the prices (High, Low, Open, Close). The output is a matrix of 4 rows and n (number of available data points) columns.

Data Processing Module:

- The parameters are set for the agent.
- The environment is created for the trading agent.
- The actor is defined and the trading cost and trading interest is calculated.
- The agent portfolio value and the baseline value are found.

Quantum Approximate Optimizer:

- The covariance is calculated.
- The weight of the assets is added
- The portfolio optimum and risks are returned.

CHAPTER 5

SYSTEM IMPLEMENTATION

6.1 DATA PROCESSING MODULE CODE:

```
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
import numpy as np
from collections import deque
import random
import pandas as pd
# !pip install ffn
import ffn
# !pip install gym
from environment import *
%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt
from tqdm import tqdm
path_data = './np_data/input.npy'

list_stock = ['BAC', 'IBM', 'PFE', 'TSLA', 'XOM']

data = np.load(path_data)
trading_period = data.shape[2]
nb_feature_map = data.shape[0]
nb_stocks = data.shape[1]
```

```
m = nb_stocks


dict_hp_net = {'n_filter_1': 2, 'n_filter_2': 20, 'kernel1_size':(1, 3)}
dict_hp_pb = {'batch_size': 50, 'ratio_train': 0.6,'ratio_val': 0.2, 'length_tensor': 10,
              'ratio_greedy':0.8, 'ratio_regul': 0.1}
dict_hp_opt = {'regularization': 1e-8, 'learning': 9e-2}
dict_fin = {'trading_cost': 0.25/100, 'interest_rate': 0.02/250, 'cash_bias_init': 0.7}
dict_train = {'pf_init_train': 10000, 'w_init_train': 'd', 'n_episodes':2, 'n_batches':10}
dict_test = {'pf_init_test': 10000, 'w_init_test': 'd'}



n_filter_1 = dict_hp_net['n_filter_1']
n_filter_2 = dict_hp_net['n_filter_2']
kernel1_size = dict_hp_net['kernel1_size']

# Size of mini-batch during training
batch_size = dict_hp_pb['batch_size']
# Total number of steps for pre-training in the training set
total_steps_train = int(dict_hp_pb['ratio_train']*trading_period)

# Total number of steps for pre-training in the validation set
total_steps_val = int(dict_hp_pb['ratio_val']*trading_period)

# Total number of steps for the test
total_steps_test = trading_period-total_steps_train-total_steps_val
```

```
# Number of the columns (number of the trading periods) in each input price matrix
n = dict_hp_pb['length_tensor']

ratio_greedy = dict_hp_pb['ratio_greedy']

ratio_regul = dict_hp_pb['ratio_regul']

# The L2 regularization coefficient applied to network training
regularization = dict_hp_opt['regularization']
# Parameter alpha (i.e. the step size) of the Adam optimization
learning = dict_hp_opt['learning']

optimizer = tf.train.AdamOptimizer(learning)
trading_cost= dict_fin['trading_cost']
interest_rate= dict_fin['interest_rate']
cash_bias_init = dict_fin['cash_bias_init']

sample_bias = 5e-5  # Beta in the geometric distribution for online training sample batches
w_init_train = np.array(np.array([1]+[0]*m))#dict_train['w_init_train']

pf_init_train = dict_train['pf_init_train']

n_episodes = dict_train['n_episodes']
```

```
n_batches = dict_train['n_batches']

w_init_test = np.array(np.array([1]+[0]*m))#dict_test['w_init_test']

pf_init_test = dict_test['pf_init_test']

w_eq = np.array(np.array([1/(m+1)]*(m+1)))

w_s = np.array(np.array([1]+[0.0]*m))
def get_random_action(m):
 random_vec = np.random.rand(m+1)
return random_vec/np.sum(random_vec)
env = TradeEnv(path=path_data, window_length=n,
        portfolio_value=pf_init_train, trading_cost=trading_cost,
        interest_rate=interest_rate, train_size=dict_hp_pb['ratio_train'])

env_eq = TradeEnv(path=path_data, window_length=n,
        portfolio_value=pf_init_train, trading_cost=trading_cost,
        interest_rate=interest_rate, train_size=dict_hp_pb['ratio_train'])

env_s = TradeEnv(path=path_data, window_length=n,
        portfolio_value=pf_init_train, trading_cost=trading_cost,
        interest_rate=interest_rate, train_size=dict_hp_pb['ratio_train'])

action_fu = list()
```

```
env_fu = list()


for i in range(m):
    action = np.array([0]*(i+1) + [1] + [0]*(m-(i+1)))
    action_fu.append(action)

    env_fu_i = TradeEnv(path=path_data, window_length=n,
               portfolio_value=pf_init_train, trading_cost=trading_cost,
               interest_rate=interest_rate, train_size=dict_hp_pb['ratio_train'])

    env_fu.append(env_fu_i)

class Policy(object):
    '''
    This class is used to instanciate the policy network agent

    '''

    def __init__(self, m, n, sess, optimizer,
                 trading_cost=trading_cost,
                 interest_rate=interest_rate,
                 n_filter_1=n_filter_1,
                 n_filter_2=n_filter_2):
```

```
# parameters
self.trading_cost = trading_cost
self.interest_rate = interest_rate
self.n_filter_1 = n_filter_1
self.n_filter_2 = n_filter_2
self.n = n
self.m = m

with tf.variable_scope("Inputs"):

    # Placeholder

    # tensor of the prices
    self.X_t = tf.placeholder(
        tf.float32, [None, nb_feature_map, self.m, self.n])  # The Price tensor
    # weights at the previous time step
    self.W_previous = tf.placeholder(tf.float32, [None, self.m+1])
    # portfolio value at the previous time step
    self.pf_value_previous = tf.placeholder(tf.float32, [None, 1])
    # vector of Open(t+1)/Open(t)
    self.dailyReturn_t = tf.placeholder(tf.float32, [None, self.m])

    #self.pf_value_previous_eq = tf.placeholder(tf.float32, [None, 1])
```

```
with tf.variable_scope("Policy_Model"):

    # variable of the cash bias
    bias = tf.get_variable('cash_bias', shape=[
                           1, 1, 1, 1], initializer=tf.constant_initializer(cash_bias_init))
    # shape of the tensor == batchsize
    shape_X_t = tf.shape(self.X_t)[0]
    # trick to get a "tensor size" for the cash bias
    self.cash_bias = tf.tile(bias, tf.stack([shape_X_t, 1, 1, 1]))
    # print(self.cash_bias.shape)

    with tf.variable_scope("Conv1"):
        # first layer on the X_t tensor
        # return a tensor of depth 2
        self.conv1 = tf.layers.conv2d(
            inputs=tf.transpose(self.X_t, perm=[0, 3, 2, 1]),
            activation=tf.nn.relu,
            filters=self.n_filter_1,
            strides=(1, 1),
            kernel_size=kernel1_size,
            padding='same')

    with tf.variable_scope("Conv2"):
```

```
    #feature maps
    self.conv2 = tf.layers.conv2d(
        inputs=self.conv1,
        activation=tf.nn.relu,
        filters=self.n_filter_2,
        strides=(self.n, 1),
        kernel_size=(1, self.n),
        padding='same')

with tf.variable_scope("Tensor3"):
    #w from last periods
    # trick to have good dimensions
    w_wo_c = self.W_previous[:, 1:]
    w_wo_c = tf.expand_dims(w_wo_c, 1)
    w_wo_c = tf.expand_dims(w_wo_c, -1)
    self.tensor3 = tf.concat([self.conv2, w_wo_c], axis=3)

with tf.variable_scope("Conv3"):
    #last feature map WITHOUT cash bias
    self.conv3 = tf.layers.conv2d(
        inputs=self.conv2,
        activation=tf.nn.relu,
        filters=1,
        strides=(self.n_filter_2 + 1, 1),
        kernel_size=(1, 1),
```

```
        padding='same')

with tf.variable_scope("Tensor4"):
    #last feature map WITH cash bias
    self.tensor4 = tf.concat([self.cash_bias, self.conv3], axis=2)
    # we squeeze to reduce and get the good dimension
    self.squeezed_tensor4 = tf.squeeze(self.tensor4, [1, 3])

with tf.variable_scope("Policy_Output"):
    # softmax layer to obtain weights
    self.action = tf.nn.softmax(self.squeezed_tensor4)

with tf.variable_scope("Reward"):
    # computation of the reward
    #please look at the chronological map to understand
    constant_return = tf.constant(
        1+self.interest_rate, shape=[1, 1])
    cash_return = tf.tile(
        constant_return, tf.stack([shape_X_t, 1]))
    y_t = tf.concat(
        [cash_return, self.dailyReturn_t], axis=1)
    Vprime_t = self.action * self.pf_value_previous
    Vprevious = self.W_previous*self.pf_value_previous

    # this is just a trick to get the good shape for cost
```

```
        constant = tf.constant(1.0, shape=[1])

        cost = self.trading_cost * \
            tf.norm(Vprime_t-Vprevious, ord=1, axis=1)*constant

        cost = tf.expand_dims(cost, 1)

        zero = tf.constant(
            np.array([0.0]*m).reshape(1, m), shape=[1, m], dtype=tf.float32)

        vec_zero = tf.tile(zero, tf.stack([shape_X_t, 1]))
        vec_cost = tf.concat([cost, vec_zero], axis=1)

        Vsecond_t = Vprime_t - vec_cost

        V_t = tf.multiply(Vsecond_t, y_t)
        self.portfolioValue = tf.norm(V_t, ord=1)
        self.instantaneous_reward = (
            self.portfolioValue-self.pf_value_previous)/self.pf_value_previous


    with tf.variable_scope("Reward_Equiweighted"):
        constant_return = tf.constant(
            1+self.interest_rate, shape=[1, 1])
        cash_return = tf.tile(
```

```
            constant_return, tf.stack([shape_X_t, 1]))
        y_t = tf.concat(
            [cash_return, self.dailyReturn_t], axis=1)



        V_eq = w_eq*self.pf_value_previous
        V_eq_second = tf.multiply(V_eq, y_t)

        self.portfolioValue_eq = tf.norm(V_eq_second, ord=1)

        self.instantaneous_reward_eq = (
            self.portfolioValue_eq-self.pf_value_previous)/self.pf_value_previous

    with tf.variable_scope("Max_weight"):
        self.max_weight = tf.reduce_max(self.action)
        print(self.max_weight.shape)



    with tf.variable_scope("Reward_adjusted"):

        self.adjested_reward = self.instantaneous_reward -
self.instantaneous_reward_eq - ratio_regul*self.max_weight

    #objective function
    #maximize reward over the batch
```

```
        # min(-r) = max(r)
        self.train_op = optimizer.minimize(-self.adjested_reward)

        # some bookkeeping
        self.optimizer = optimizer
        self.sess = sess

    def compute_W(self, X_t_, W_previous_):
        """
        This function returns the action the agent takes
        given the input tensor and the W_previous

        It is a vector of weight

        """

        return self.sess.run(tf.squeeze(self.action), feed_dict={self.X_t: X_t_,
self.W_previous: W_previous_})

    def train(self, X_t_, W_previous_, pf_value_previous_, dailyReturn_t_):
        """
        This function trains the neural network
        maximizing the reward
        the input is a batch of the differents values
        """
```

```
self.sess.run(self.train_op, feed_dict={self.X_t: X_t_,
                                        self.W_previous: W_previous_,
                                        self.pf_value_previous: pf_value_previous_,
                                        self.dailyReturn_t: dailyReturn_t_})
```

CHAPTER 6
SYSTEM ANALYSIS

6.1 RESULT AND ANALYSIS

The first part is the definition of the problem instance such as the number of considered assets and the encoding using the Hamiltonian. In our case, we choose the following assets: Exxon, Bank of America, IBM, Pfizer, and Tesla. This problem configuration allows us to have assets from a large range of big companies in very different industries. We use 5 qubits to represent each asset in the end. Then, we load the data from Yahoo Finance between 01/01/2015 until 12/31/2020 into a data frame and plot the covariance matrix.

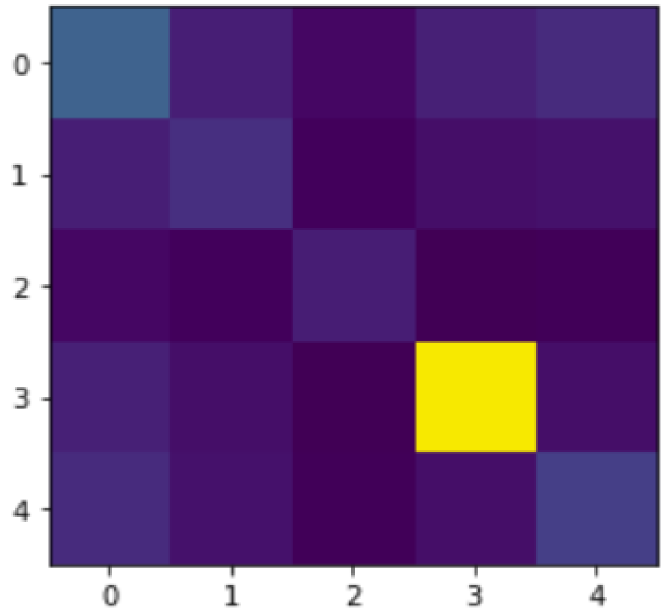

Fig. 6.1 - Covariance matrix

We can see this result confirms the portfolio we chose. Indeed, the covariance matrix shows no direct correlation between the different sets of data referring to the time evolution of each asset. This can offer a variety of portfolio optimization because the different assets do not influence one another.

Now that the assets and the number of qubits used for the algorithm are defined, we can set the parameters for QAOA. We set the following:

$$q = 0.5$$

$$B = num\_assets \; // \; 2$$

$$Penalty = num\_assets$$

This penalty variable allows us to set a parameter to scale the budget penalty term as presented in the definition of the condition equation.

From that, we can use the get_operator() function from the portfolio class of Qiskit finance that takes the covariance terms as implementation of the data to obtain qubitOp and offset in order to run the quantum algorithm. We obtain the following configuration.

```
budget 2 penalty 5 qubitOp Representation: paulis, qubits: 5, size: 15 offset 7.498412964453026
```

Fig. 6.2 - Parameters setting

Moreover, to compare the results obtained by the Qiskit implementation of QAOA, we use the Numpy eigensolver as a classical reference. We obtain the following result for our case.

```
Optimal: selection [0 1 0 0 1], value -0.0030

----------------- Full result --------------------
selection          value             probability
---------------------------------------------------
[0 1 0 0 1]        -0.0030           1.0000
[1 1 1 1 1]        44.9969           0.0000
[0 1 1 1 0]        4.9990            0.0000
[1 0 0 0 0]        5.0002            0.0000
[0 1 0 0 0]        4.9994            0.0000
[1 1 0 0 0]        -0.0004           0.0000
[0 0 1 0 0]        4.9999            0.0000
[1 0 1 0 0]        0.0001            0.0000
[0 1 1 0 0]        -0.0007           0.0000
[1 1 1 0 0]        4.9996            0.0000
[0 0 0 1 0]        4.9996            0.0000
[1 0 0 1 0]        -0.0001           0.0000
[0 1 0 1 0]        -0.0010           0.0000
[1 1 0 1 0]        4.9993            0.0000
[0 0 1 1 0]        -0.0005           0.0000
[1 0 1 1 0]        4.9998            0.0000
[1 1 1 1 0]        19.9992           0.0000
[0 1 1 1 1]        19.9966           0.0000
[0 0 0 0 1]        4.9976            0.0000
[1 0 0 0 1]        -0.0021           0.0000
[1 1 0 0 1]        4.9973            0.0000
[0 0 1 0 1]        -0.0025           0.0000
[1 0 1 0 1]        4.9978            0.0000
[0 1 1 0 1]        4.9970            0.0000
[1 1 1 0 1]        19.9972           0.0000
[0 0 0 1 1]        -0.0027           0.0000
[1 0 0 1 1]        4.9975            0.0000
[0 1 0 1 1]        4.9967            0.0000
[1 1 0 1 1]        19.9969           0.0000
[0 0 1 1 1]        4.9972            0.0000
[1 0 1 1 1]        19.9974           0.0000
[0 0 0 0 0]        20.0000           0.0000
```

Fig. 6.3 - Numpy eigensolver results

We can see the optimal portfolio being [0,1,0,0,1] with a high probability of 1. As presented before, in this quantum approach, because we are dealing with qubits resulting either in 0 or 1, the portfolio is whether we take the asset or not. Thus, the classical eigensolver indicates the optimal portfolio is to take the full budget and split between Bank of America and Tesla.

Now, we can use the QAOA function of Qiskit algorithms library and set the following parameters: qubitOp as the operator and cobyla as the optimizer. We obtain the following results.

```
Optimal: selection [1. 0. 1. 0. 0.], value 0.0001

----------------- Full result --------------------
selection        value           probability
--------------------------------------------------
[1 0 1 0 0]      0.0001          0.0741
[1 0 0 1 0]      -0.0001         0.0739
[1 1 0 0 0]      -0.0004         0.0739
[0 0 1 1 0]      -0.0005         0.0738
[0 1 1 0 0]      -0.0007         0.0737
[0 1 0 1 0]      -0.0010         0.0736
[1 0 0 0 1]      -0.0021         0.0732
[0 0 1 0 1]      -0.0025         0.0731
[0 0 0 1 1]      -0.0027         0.0730
[0 1 0 0 1]      -0.0030         0.0729
[1 0 1 1 0]      4.9998          0.0205
[1 1 1 0 0]      4.9996          0.0205
[1 1 0 1 0]      4.9993          0.0205
[0 1 1 1 0]      4.9990          0.0204
[1 0 1 0 1]      4.9978          0.0202
[1 0 0 1 1]      4.9975          0.0202
[1 1 0 0 1]      4.9973          0.0201
[0 0 1 1 1]      4.9972          0.0201
[0 1 1 0 1]      4.9970          0.0201
[0 1 0 1 1]      4.9967          0.0200
[1 0 0 0 0]      5.0002          0.0107
[0 0 1 0 0]      4.9999          0.0106
[0 0 0 1 0]      4.9996          0.0106
[0 1 0 0 0]      4.9994          0.0106
[0 0 0 0 1]      4.9976          0.0104
[0 0 0 0 0]      20.0000         0.0045
[1 1 1 1 1]      44.9969         0.0033
[0 1 1 1 1]      19.9966         0.0003
[1 1 0 1 1]      19.9969         0.0003
[1 1 1 0 1]      19.9972         0.0003
[1 0 1 1 1]      19.9974         0.0003
[1 1 1 1 0]      19.9992         0.0003
```

Fig. 6.4 - QAOA results

The optimal selection found by the QAOA algorithm is [1,0,1,0,0] with probability of 0.0741 which is very low and close to the 9 other configurations with 2 assets taken into account. This portfolio is thus different from the Numpy selection because it takes Exxon and Tesla as the best assets to allocate the budget.

CHAPTER 7
CONCLUSION

7.1 CONCLUSION

The result is a portfolio where we divide all the budget into the different chosen assets. The weights are thus equally shared among the assets that were found relevant at the end of the QAOA calculations.

From the provided raw data input for discrete stocks, we are able to analyze and pre-process the given data, then convert the raw data into a time series of the prices.

Using this, in accordance to the portfolio vector memory the raw data is processed to find the portfolio value of each agent, the quadratic problem is defined and the quantum circuits find the portfolio weights returning the optimum solution.

We obtain the final optimal selection for QAOA based on the scope of the study which corresponds to a portfolio with 50% on Exxon and 50% on Tesla.

Quantum computers are expected to have a substantial impact on the Finance industry, as they will eventually be able to solve certain problems considerably faster than the best known classical algorithms.

In conclusion, Quantum Machine-Learning is a viable solution to take into account when talking about this optimization problem.

7.2 FUTURE ENHANCEMENTS

1. Migrating to Quantum Systems:

The current systems runs on classical computers, migrating the system to a quantum computing environment will improve the system efficiency.

2. Integration with Big Systems:

The system has capacity to handle big data integration which will improve the system throughput.

3. Market Volatility:

The financial market is volatile in nature and maybe subjected to unexpected changes beyond available scope.

APPENDICES

APPENDICES

	Date	Open	High	Low	Close	ticker
22	2019-02-04	20.865334	21.020000	20.125334	20.859333	TSLA
72	2019-04-16	138.049713	138.996170	137.686417	138.757172	IBM
455	2020-10-21	35.436432	35.531307	35.161289	35.180267	PFE
305	2020-03-19	30.569260	30.597723	28.472486	28.861481	PFE
166	2019-08-29	68.300003	68.669998	68.089996	68.430000	XOM

Fig A.1 Data after analysis and pre-processing

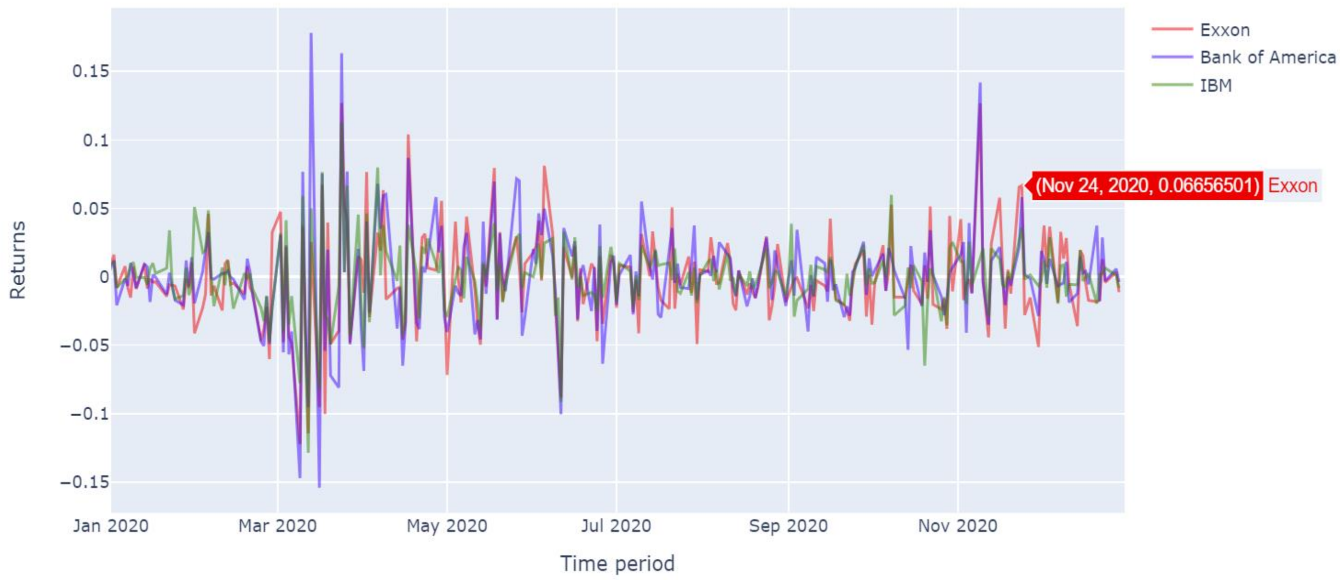

Fig A.2 Asset Returns for discrete stocks

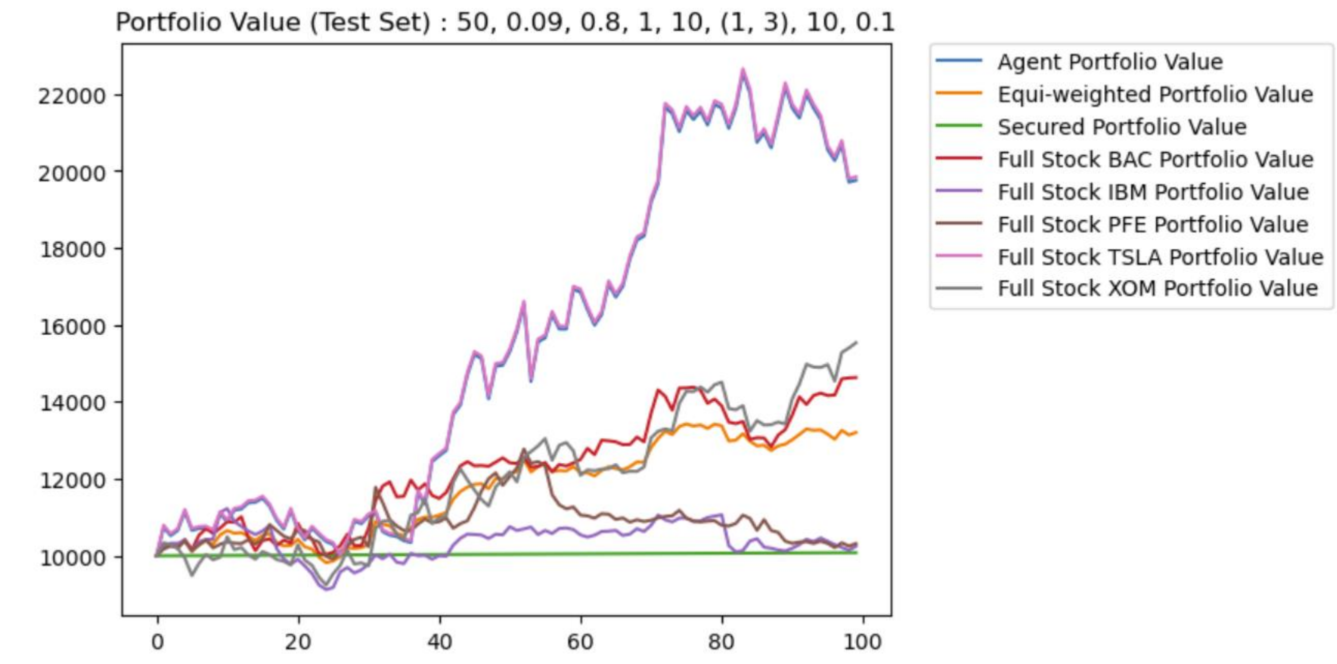

Fig A.3 Portfolio value graph after testing and training

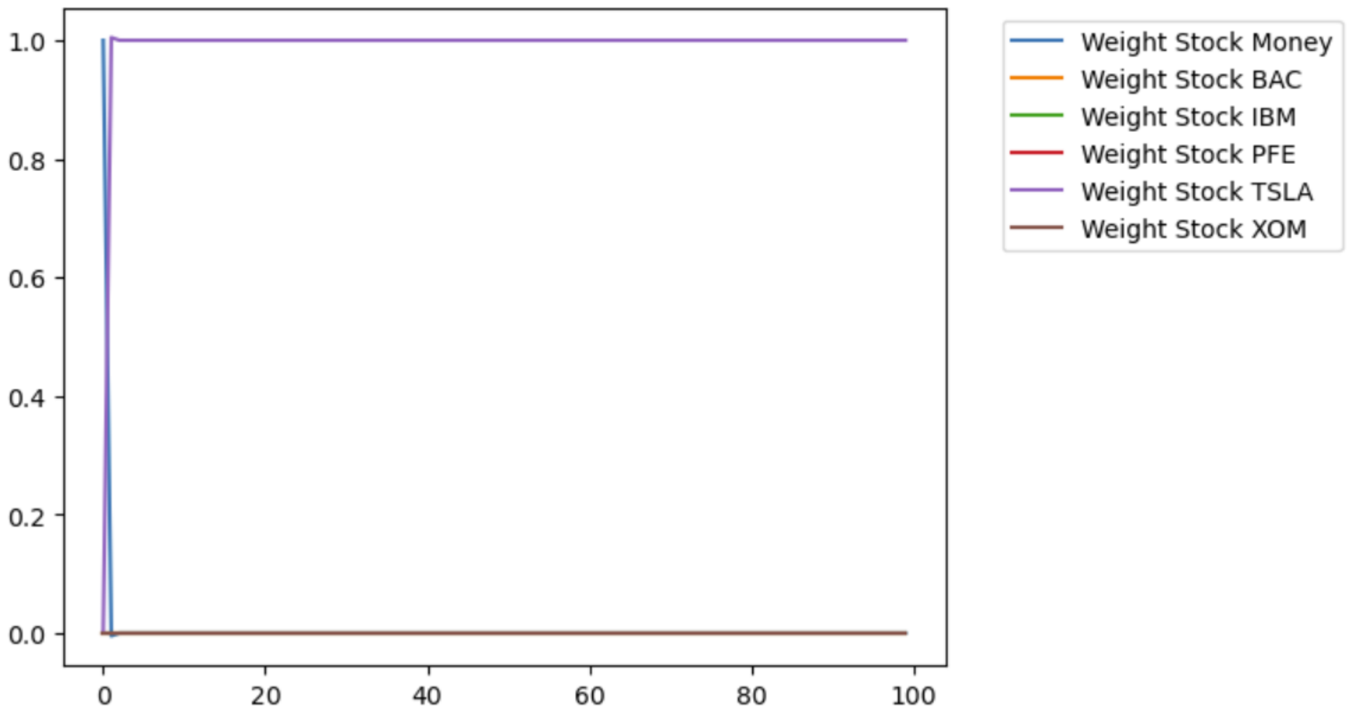

Fig A.4 Weight Evolution during training

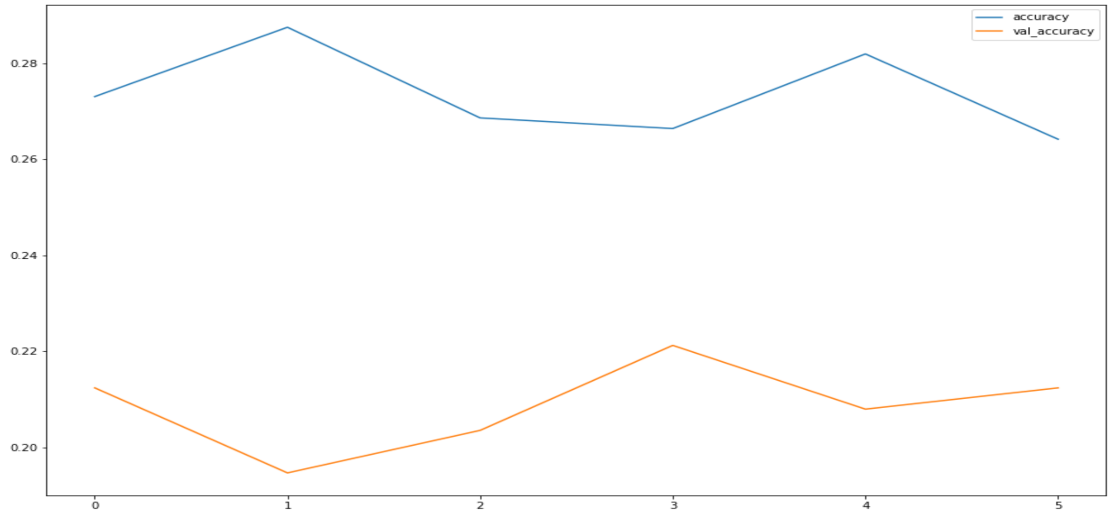

Fig A.5 Plot of accuracy

REFERENCES

1. Crooks GE (2018) *Performance of the quantum approximate optimization algorithm on the maximum cut problem*. arXiv preprint arXiv:181108419 https://doi.org/10. 48550/arXiv.1811.08419
2. E. Campbell, A. Khurana, and A. Montanaro, *Applying quantum algorithms to constraint satisfaction problems*, Tech. Rep. (2018) arXiv:1810.05582v1.
3. Gavin E Crooks. *Performance of the quantum approximate optimization algorithm on the maximum cut problem*. arXiv preprint arXiv:1811.08419, 2018. URL https:// arxiv.org/abs/1811.08419.
4. Hao Fu, Prashanth Krishnamurthy, et al., *A Deep Neural Network Algorithm for Linear-Quadratic Portfolio Optimization With MGARCH and Small Transaction Costs*, IEEE Xplore (2023), DOI: 10.1109/ACCESS.2023.3245570
5. IBM Qiskit Ecosystem, *Portfolio Optimization*, https://qiskit-community.github.io/qiskit-finance/tutorials/01_portfolio_optimization.html
6. Junkyu Jang, NohYoon Seong, *Deep reinforcement learning for stock portfolio optimization by connecting with modern portfolio theory*, ScienceDirect(2023), https://doi.org/10.1016/j.eswa.2023.119556
7. N. Moll, P. Barkoutsos, L. S. Bishop, J. M. Chow, A. Cross, D. J. Egger, et al., *Quantum optimization using variational algorithms on near-term quantum devices*, Quantum Science and Technology 3, 030503 (2018).
8. Prakash K. Aithal; M. Geetha, et al., *Real-Time Portfolio Management System Utilizing Machine Learning Techniques*, IEEE Xplore (2023), DOI: 10.1109/ACCESS.2023.3263260
9. Roman Orus, Samuel Mugel, Enrique Lizaso, *Quantum computing for finance: overview and prospects*,(2024) arXiv preprint *https://arxiv.org/abs/1807.03890*
10. Schäfer LE, Dietz T, Barbati M, Figueira JR, Greco S, Ruzika S *The binary knapsack problem with qualitative levels*. European Journal of Operational

Research 289(2):508–514, (2021) https://doi.org/10.1016/j.ejor. 2020.07.040

11. Shunza J, Akinyemi M, Yinka-Banjo C (2023) *Application of quantum computing in discrete portfolio optimization*. Journal of Management Science and Engineering https: //doi.org/10.1016/j.jmse.2023.02.001
12. Speidell LS, Miller DH, Ullman JR (1989) *Portfolio optimization: A primer. Financial Analysts Journal* 45(1):22– 30, https://doi.org/10.2469/faj.v45.n1.22
13. Tae-Kyung Kim, Youngsun Han, et al., *Empirical Analysis of Quantum Approximate Optimization Algorithm for Knapsack-based Financial Portfolio Optimization,* (2024) arXiv preprint *https://arxiv.org/abs/2402.07123*
14. Weipeng Zhang, Tao Yin, et al., *Reinforcement Learning for Stock Prediction and High-Frequency Trading With T+1 Rules,* IEEE Xplore (2023), DOI: 10.1109/ACCESS.2022.3197165
15. Yu-Chi Jiang, Ming-Ho Chang, et al., *Trend Ratio-Based Portfolio Optimization Model Adopting Entanglement-enhanced Quantum-Inspired Evolutionary Computation in the Global Financial Markets,* IEEE(2023), DOI: 10.1109/CEC53210.2023.10254025
16. Yu-Chi Jiang, Yun-Ting Lai, et al., *Quantum-inspired Computing: Entanglement-enhanced Technique for Short Portfolio in Global Markets*, IEEE Xplore (2023), DOI: 10.1109/NANO58406.2023.10231234
17. Zhou L, Wang ST, Choi S, Pichler H, Lukin MD (2020*) Quantum approximate optimization algorithm: Performance, mechanism, and implementation on near-term devices.* Phys Rev X 10:021067, https://doi.org/10.1103/ PhysRevX.10.021067