Configuration Manual for Combined Genetic Algorithm and Gradient Descent Algorithm to Optimize Server Selection in Mobile Edge Computing

MSc Research Project
Masters in Cloud Computing

Tamaraebi Besife Pibowei
Student ID: x20217871

School of Computing
National College of Ireland

Supervisor: Mr. Sean Heeney

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: Tamaraebi Besife Pibowei

Student ID: x20217871

Programme: Master of Science in Cloud Computing **Year:** 2021

Module: ……………Research Project……….………

Lecturer: ……………Mr. Sean Heeney.………

Submission Due Date: ………………16th December 2021……….………

Project Title: Combined Genetic Algorithm and Gradient Descent Algorithm to Optimize Server Selection in Mobile Edge Computing

Word Count: 7296 **Page Count: 27**

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: ……Tamaraebi Besife Pibowei……………

Date: …………09th December 2021………………

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual for Combined Genetic Algorithm and Gradient Descent Algorithm to Optimize Server Selection in Mobile Edge Computing

Tamaraebi Besife Pibowei
x20217871

1 Introduction

This document contains a brief description of the configuration manual i.e a complete guide that was used the implementation of the research work "Combined Genetic Algorithm and Gradient Descent Algorithm to Optimize Server Selection in Mobile Edge Computing". For this research project, an experiment was performed to simulation the implementation of optimization of server selection in Mobile Edge Computing (MEC). The EUA dataset was applied to the experiments and used to generate instances for the mobile user and base stations. The goal of the experiment was evaluating the performance of combine Genetic Algorithm and Gradient Descent Algorithm in optimizing server selection in MEC. The remainder of the document is divided into four sections: section 2 describes the hardware specification and the software requirement for implementing the research project, section 3 describes the software installation guide for software used in implementing the research work. Section 4 describes the implementation and evaluation process used carry out this experiment and section 5 will be the conclusion note

2 System Specification

2.1 Hardware Requirement

The hardware configuration of the machine used to in the implementation the research project is:

- 3.6 GHz Intel core i7 processor
- 16GB of RAM
- 256 GB SSD Storage

2.2 Software Requirement

The software requirements for this project includes Linux Ubuntu Desktop Operation System, Anaconda Navigator, Web Browser Application. Below is the software environment used in the implementation of the project.

- OS - Ubuntu Desktop 21.04 Linux
- Anaconda Navigator 2.1.0
- Web browser – Mozilla Firefox
- Jupyter Notebook v-6.3.0

2.2.1 Ubuntu Desktop 21.04 Linux

Ubuntu is a distro of Linux operating system (OS). Ubuntu Desktop 21.04 OS is the resource manager for the hardware and other software running on the implementation machine. All other software used in the implementation of this research experiment was installed on this OS.

2.2.2 Anaconda Navigator

Anaconda Navigator is an open-source package manager, environment manager, and distribution of the Python Programming language. This application was used to run Jupyter Notebook Interactive Python IDE used in the implementation of research experiment.

2.2.3 Jupyter Notebook

Jupyter Notebook is a web-based development environment that is used execute Python script that was used implementation and present research experiment and the generated outputs.

2.2.4 Web browse – Mozilla Firefox

Jupyter Notebook uses web browser for rendering the Jupyter Notebook IDE use for our research experimentation.

3 Software Installation Guide

This section gives a guide on how to install the required software and important python libraries that were used for the research work.

3.1 Installing Anaconda

Before you begin with this guide, you should have a non-root user with sudo privileges set up on your Ubuntu PC

Step 1:

Change directory to ~/, then use ***curl*** to download the link that you copied from the Anaconda website. We'll output this to a file called *anaconda.sh* for quicker use.

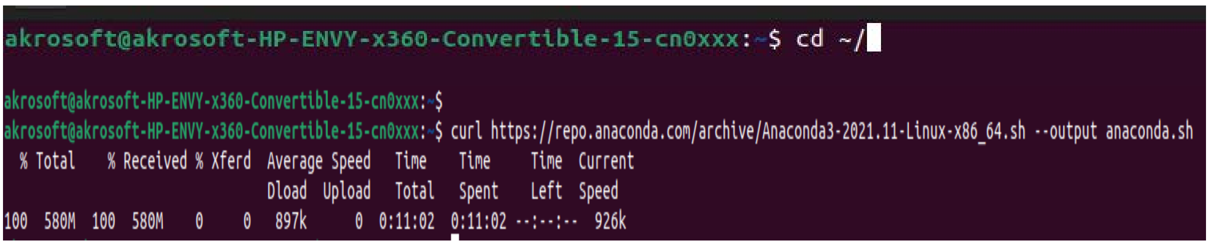

Figure 1:. Download Anaconda using curl from terminal

Step 2:

Verify the data integrity of the installer through the SHA-256 checksum

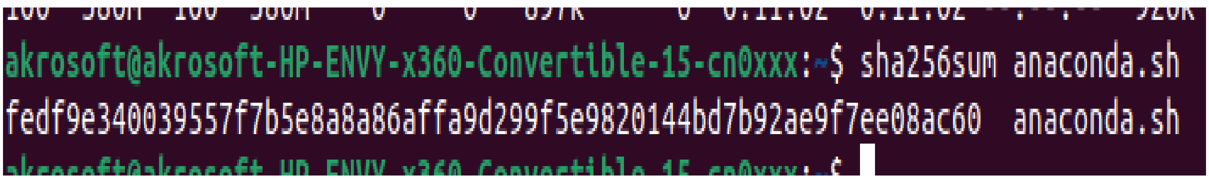

Figure 2:.   Verifying data integrity downloaded file

You should check the output against the hashes available at the Anaconda with Python 3 on 64-bit Linux page for your appropriate Anaconda version. As long as your output matches the hash displayed in the *sha2561* row, you're good to go.

Step 3:

After verifying data integrity of the installer, run the script using command below

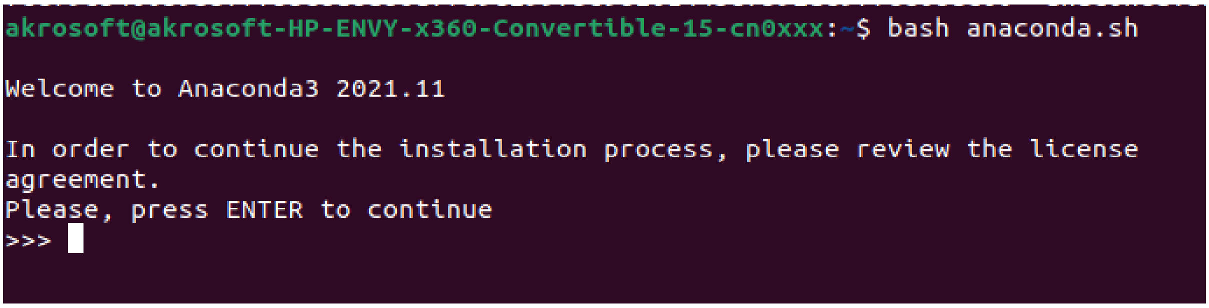

Figure 3:.   Installing Anaconda from the terminal

To continue with the installation press "Enter" on the keyboard. The next prompt will require you to accept the license terms. You have to type **yes** to continue with the installation.

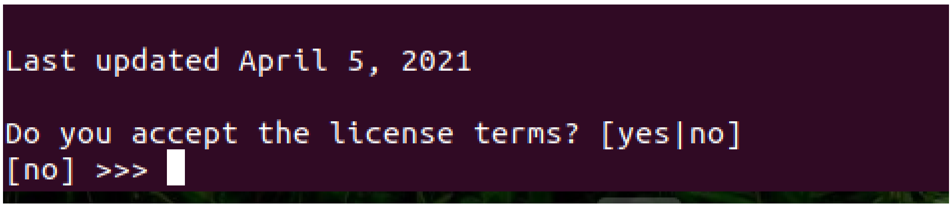

Figure 4:.   Accepting license term

Step 4:

At this point, you'll be prompted to choose the location of the installation. You can press ENTER to accept the default location, or specify a different location to modify it.

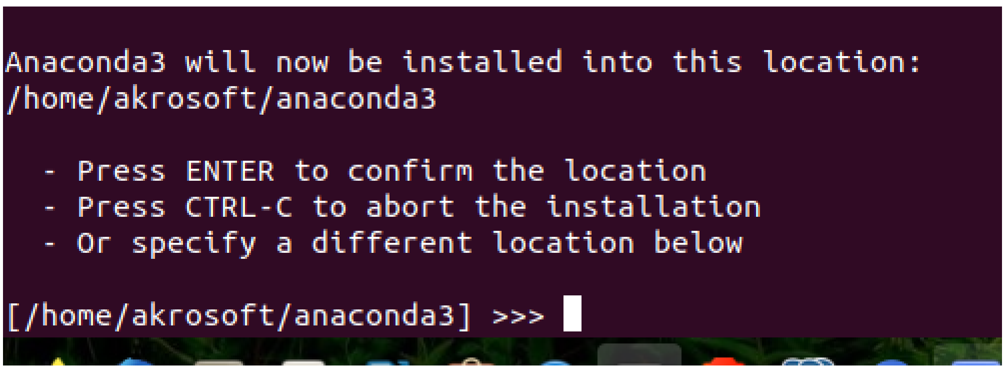

Figure 5:. Selecting Anaconda installation directory

The installation process will continue. Note that it may take some time. Once the installation is complete, you'll receive the following output:

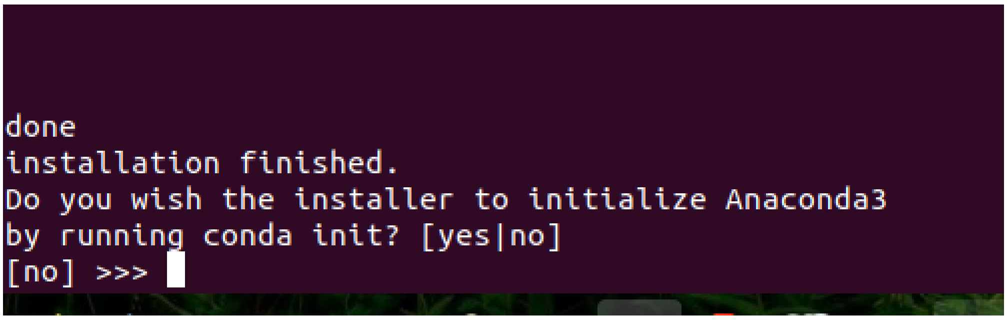

Figure 6:. Completed installation prompt

Type yes so that you can initialize Anaconda3. You'll receive some output that states changes made in various directories. One of the lines you receive will thank you for installing Anaconda. The installation has been completed; next step will start anaconda navigator on Ubuntu.

Step 5:

To start anaconda navigator which is the GUI tool, on the terminal the following command and press the Enter key on your keyboard

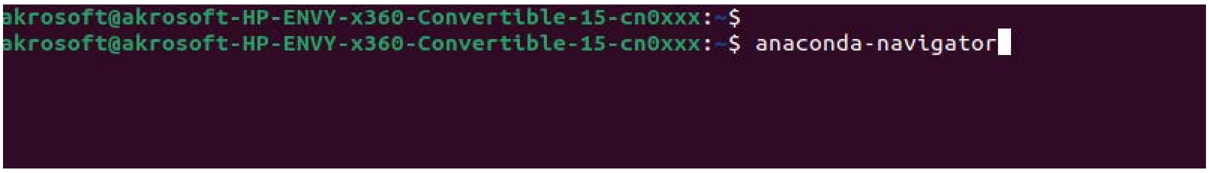

Figure 7:. Start Anaconda Navigator from the Terminal

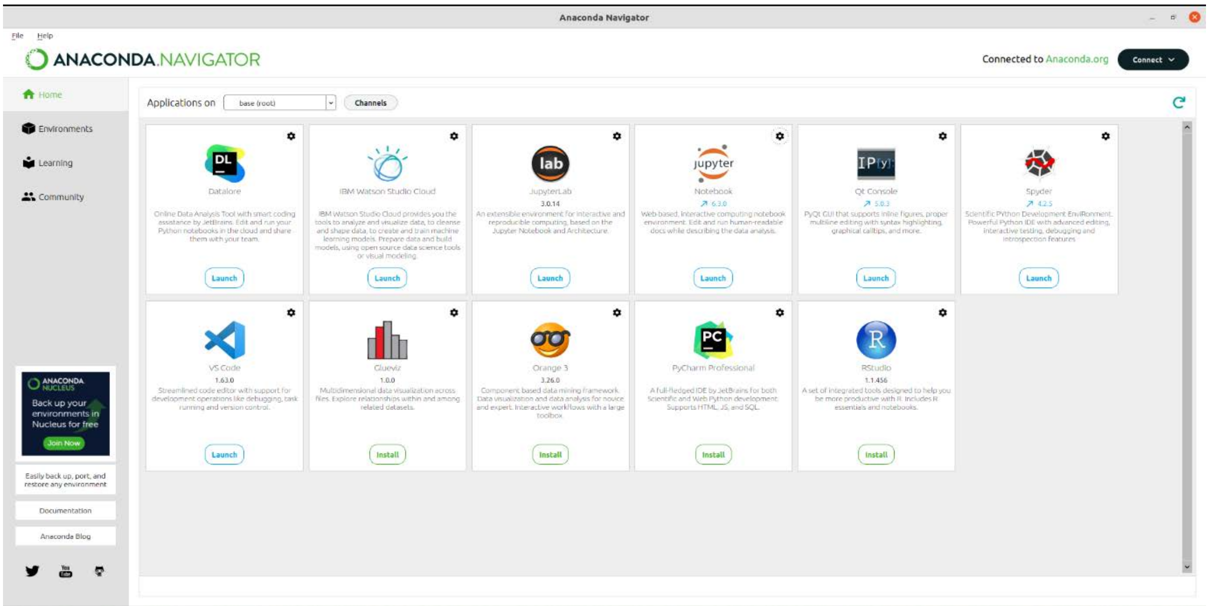

Figure 8:. Anaconda home page

4 Implementation and Evaluation

4.1 Starting a new project in Jupyter Notebook

Now that Anaconda is installed and running, the step listed below will guide you on how to load Jupyter notebook and start a new project

Step 1:

From the Anaconda home screen, click on **Launch** button on Jupyter Notebook

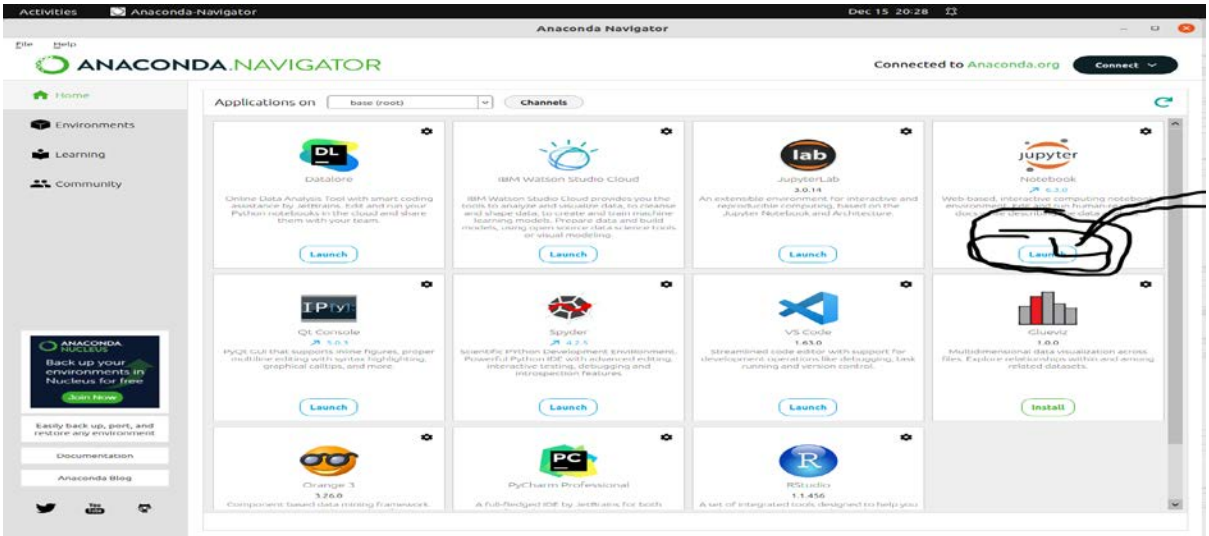

Figure 9:. Launch Jupyter Notebook from Anaconda Navigator

Step 2:
In Jupyter, on the loaded root directory click new dropdown button and select Python 3 to start a new project

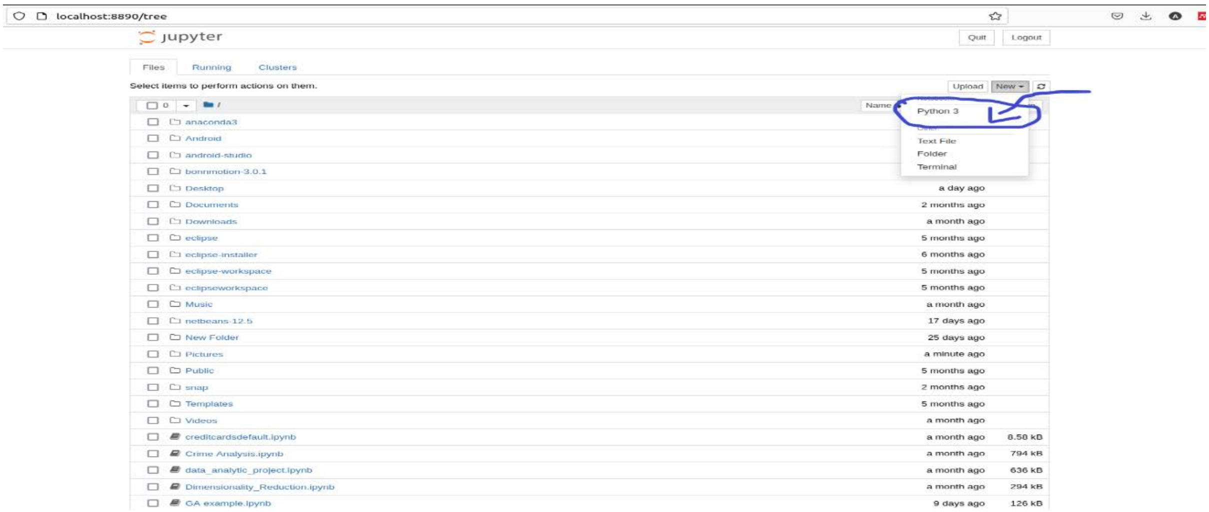

Figure 10:. Jupyter Notebook root directory page

4.2 Importing the required libraries from python into our IDE workspace

The libraries used during the implementation of various functionality of this research work are listed and shown in figure 11 below

In [1]:
```
import sys
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.path as mpath

import time
import math
import enum

from collections import namedtuple
from random import choices, randint, randrange, random, uniform
from typing import List, Callable, Tuple, Dict, Union, TypeVar
from functools import partial
```

Figure 11:. Imported Python libraries

4.3 Import Dataset and Perform pre-processing task on the Dataset

The EUA dataset was downloaded from github.com, stored in the same directory as the project file and imported into Jupyter Notebook by using panda's library and viewed the first 5 records. Figure 12 below shows the implementation

```
In [3]: df_users = pd.read_csv(os.getcwd() + '/dataset/users.csv')
        df_base_stations = pd.read_csv(os.getcwd() + '/dataset/edge_servers.csv')
```

```
In [4]: df_base_stations = df_base_stations[['LATITUDE', 'LONGITUDE']]

        df_users.rename(columns = {'Latitude':'LATITUDE', 'Longitude':'LONGITUDE'}, inplace = True)
```

```
In [5]: # df_users = df_users.head(10)
        df_users.head()
```

Out[5]:

	LATITUDE	LONGITUDE
0	-37.814619	144.974443
1	-37.810140	144.970454
2	-37.819892	144.957305
3	-37.814524	144.953632
4	-37.814100	144.963000

```
In [6]: # df_base_stations = df_base_stations.head(20)
        df_base_stations.head()
```

Out[6]:

	LATITUDE	LONGITUDE
0	-37.81517	144.97476
1	-37.81524	144.95256
2	-37.81239	144.97120
3	-37.81679	144.96918
4	-37.81808	144.95692

Figure 12:. Importing EUA dataset into IDE

4.4 Definition of helper functions and Variables

4.4.1 Variable declaration

Some variables were required to track data custom data-structure and computational values from the experiment execution for later use. These variables were declared and initialized with a default.

```
In [9]: user_ids = list()
        server_ids = list()
        base_ids = list()
        request_ids = list()

        user_instances = list()
        station_instances = list()
        server_instances = list()
        request_instances = list()

        user_id_init = "mobus"
        serv_id_init = "serv"
        reqs_id_init = "reqs"
        bs_id_init = "bs"

        earth_radius = 6373.0 # Earth Radius in Km
        base_station_radius = 0.4 # 400m
        fitness_limit = None

        SISPBaseStation = NestedDict
```

Figure 13:. Declared and Initialized variables

4.4.2 Helper Function definitions

Some helper functions were created to facilitate reusability and also to create specialization. Below are some helper functions

4.4.2.1 Compute x and y coordinates from latitude and longitude

This function was created to help compute the x-axis and y-axis value for a 2-dimensional plane from the latitude and longitude argument supplied to the function

```
In [10]: def  compute_xy_coords_latlng(lat, lng):
             x = earth_radius * math.cos(lat) * math.cos(lng)
             y = earth_radius * math.cos(lat) * math.sin(lng)
             return x, y
```

Figure 14:. Compute xy axis values function definition

4.4.2.2 Generate ID for custom datatype

These 3 sets of function were created to generate unique ids for all custom define data structure. ***Generate_random_string()*** is used to generate random string with 20 characters in length, ***generate_id()*** is used to generate new instance id while the ***get_id()*** is used to ensured that the generated id is unique the class for which the id is being generated

```
In [11]: def generate_random_string():
             charset = "AaBbCcDdEeFfGgHhIiJjKkLlMmNnOoPpQqRrSsTtUuVvWwXxYyZz";
             length_charset = len(charset)
             rand_string = ""
             for i in range(20):
                 rand_string += charset[randint(0, (length_charset - 1))]

             return rand_string
```

```
In [12]: def generate_id(id_initials):
             id = id_initials + "-" + str(int(time.time()*1000)) + generate_random_string()
             return id
```

```
In [13]: def get_id(initials):
             isValid = False
             id = ""
             while not isValid:
                 id = generate_id(initials)
                 if user_id_init == initials and (not (id in user_ids)):
                     user_ids.append(id)
                     isValid = True
                 elif serv_id_init == initials and (not (id in server_ids)):
                     server_ids.append(id)
                     isValid = True
                 elif reqs_id_init == initials and (not (id in request_ids)):
                     request_ids.append(id)
                     isValid = True
                 elif bs_id_init == initials and (not (id in base_ids)):
                     base_ids.append(id)
                     isValid = True

             return id
```

Figure 15:. Generate id helper functions

4.4.2.3 Retrieve custom datatype by ID

```
In [14]: def get_request_by_id(requests, req_id):
             target = None

             for request in requests:
                 if request.get_id() == req_id:
                     target = request

             return target
```

```
In [15]: def get_station_by_id(base_stations, stat_id):
             target = None

             for base_station in base_stations:
                 if base_station.get_id() == stat_id:
                     target = base_station

             return target
```

```
In [16]: def get_user_by_id(users, user_id):
             target = None

             for user in users:
                 if user.get_id() == user_id:
                     target = user

             return target
```

```
In [17]: def get_server_by_id(servers, serv_id):
             target = None

             for server in servers:
                 if server.get_id() == serv_id:
                     target = server

             return target
```

Figure 16:. Helper function to retrieve custom datatype instance from a list item

Figure 16 above show a group of functions create to help retrieve instances of the defined custom datatype for the different classes created. These functions accept two arguments and these includes a list of instances from the target class and id of the target instance.

4.5 Definition Custom/User Defined Data-structure

Four custom data structures were created to model Mobile User, Base Station, Server and Request datatypes. These implementations are listed below

4.5.1 MobileUser Class

The MobileUser class defines a set of attributes that help to store and track data about an instance of this class and defines getters and setters' methods used to update and retrieve these data. The class also defined other methods which enable the use perform functions such as connecting to a base station, check if all user request has been processed.

In [19]:
```
class MobileUser:
    def __init__(self, latitude, longitude):
        self.id = get_id(user_id_init)
        self.latitude = latitude
        self.longitude = longitude
        self.x_axis = None
        self.y_axis = None
        self.connected_to = None
        self.requests = list()
        self.compute_xy_coords()

    def get_id(self):
        return self.id

    def get_latitude(self):
        return self.latitude

    def get_longitude(self):
        return self.longitude

    def set_x_axis(self, x_value):
        self.x_axis = x_value

    def get_x_axis(self):
        return self.x_axis

    def set_y_axis(self, y_value):
        self.y_axis = y_value

    def get_y_axis(self):
        return self.y_axis

    def add_request(self, request):
        self.requests.append(request)

    def remove_request(self, request):
        self.requests.remove(request)

    def empty_requests(self):
        self.requests = list()

    def get_all_requests(self):
        return self.requests

    def get_request(self, requests, req_id):
        return get_request_by_id(requests, req_id)

    def set_connected_to(self, station_id):
        self.connected_to = station_id

    def get_connected_to(self):
        return self.connected_to

    def get_base_station(self, base_stations):
        return get_station_by_id(base_stations, self.get_connected_to())
```

```
def compute_xy_coords(self):
    x, y = compute_xy_coords_latlng(self.get_latitude(), self.get_longitude())
    self.set_x_axis(x)
    self.set_y_axis(y)

def get_requests_status(self, requests):
    if len(self.get_all_requests) <= 0:
        return Status.EMPTY
    else:
        if self.processed_all_request(requests):
            return Status.COMPLETED
        else:
            return Status.PENDING

def processed_all_request(self, requests):
    all_processed = True

    for req in self.get_all_requests():
        request = self.get_request(requests, req)

        if not request.get_status() == Status.COMPLETED:
            all_processed = False

    return all_processed

def show_base_station_distance_from_user(self, base_stations):
    base_servers = list()
    for index in range(len(base_stations)):
        print("station (" + base_stations[index].get_id() + ")  " + str(index) + " distance ==>>  " + str(se

def connect_to_base_station(self, base_stations):
    stations = list()
    for index in range(len(base_stations)):
        distance = self.distance_from_selected_base_station(base_stations[index].get_latitude(), base_station
        if distance <= base_stations[index].get_radius():
            bs_info = dict()
            bs_info["instance"] = base_stations[index]
            bs_info["distance"] = distance
            bs_info["cumpute_resource"] = 0.0
            stations.append(bs_info)

    return stations

def distance_from_selected_base_station(self, station_lat, station_lng):
    user_lat = math.radians(self.get_latitude())
    user_lng = math.radians(self.get_longitude())
    station_lat = math.radians(station_lat)
    station_lng = math.radians(station_lng)

    delta_lat = station_lat - user_lat
    delta_lng = station_lng - user_lng

    a = math.sin(delta_lat / 2)**2 + math.cos(user_lat) * math.cos(station_lat) * math.sin(delta_lng / 2)**2
    c = 2 * math.atan2(math.sqrt(a), math.sqrt(1 - a))

    return earth_radius * c
```

Figure 17:.   Definition of Mobile User Class

4.5.2 Server Class

The Server class defines a set of attributes that help to store and track data about an instance of this class and defines getters and setters' methods used to update and retrieve these data. The class defined other methods to start and process all request sent to the server instance and send out signals and response data to waiting processes and objects. Figure 18 below show the class implementation

```
In [20]: class Server:
    def __init__(self):
        self.id = get_id(serv_id_init)
        self.base_station = None
        self.cpu_layer = None
        self.cpu_status = None
        self.workload_capacity = None
        self.workload = 0
        self.request_queue = list()
        self.execute_time = None
        self.initialize_cpus()

    def get_id(self):
        return self.id

    def set_base_station(self, station):
        self.base_station = station

    def get_base_station(self):
        return self.base_station

    def get_base_station_instance(self, base_stations):
        return get_station_by_id(base_stations, self.get_base_station())

    def add_request_to_queue(self, request):
        if self.is_available(request):
            self.request_queue.append(request.get_id())
            self.set_workload(request.get_load_weight())
            return True
        else:
            return False


    def compute_request_weight_load(self, load_weight):
        return int((load_weight/self.get_workload_capacity()) * 100)

    def remove_request_from_queue(self, request):
        negative_workload = -1 * request.get_load_weight()
        self.set_workload(negative_workload)
        self.request_queue.remove(request.get_id())

    def empty_request_queue(self):
        self.request_queue = list()

    def get_all_requests_from_queue(self):
        return self.request_queue

    def get_request_from_queue(self, requests, req_id):
        return get_request_by_id(requests, req_id)

    def set_cpu_layer(self):
        self.cpu_layer = randint(7, 18)

    def get_cpu_layer(self):
        return self.cpu_layer
```

```python
    def initialize_cpus(self):
        self.set_cpu_layer()
        self.set_workload_capacity(self.compute_max_workload())
        self.set_workload(self.compute_initial_workload())
        pass

    def set_execute_time(self, chorus_eff):
        self.execute_time = self.generate_execution_time(chorus_eff)

    def get_execute_time(self):
        used, free, use_percentage, use_ratio = self.compute_cpu_usage()
        return self.execute_time

    def set_workload(self, workload):
        self.workload += workload

    def get_workload(self):
        return self.workload

    def compute_max_workload(self):
        return randint(400, 500)

    def compute_initial_workload(self):
        qtr_max_wl = int(self.get_workload_capacity() / 10)
        return randint(1, qtr_max_wl)


    def set_workload_capacity(self, max_workload):
        self.workload_capacity = max_workload

    def get_workload_capacity(self):
        return self.workload_capacity

    def compute_cpu_usage(self):
        workload_percentage = int((self.get_workload() / self.get_workload_capacity())*100)
        used_cpu_layers = int((workload_percentage/100) * self.get_cpu_layer())

        free_cpu_layers = self.get_cpu_layer() - used_cpu_layers
        usage_ratio = str(used_cpu_layers) + "/" + str(self.get_cpu_layer())
        usage_percetage = int((used_cpu_layers / self.get_cpu_layer()) * 100)

        return used_cpu_layers, free_cpu_layers, usage_percetage, usage_ratio

    def get_cpu_status(self):
        used, free, use_percentage, use_ratio = self.compute_cpu_usage()
        pass

    def is_available(self, request):
        avail = False
        estimated_workload = request.get_load_weight() + self.get_workload()
        if estimated_workload < self.get_workload_capacity():
            avail = True

        return avail
```

Figure 18:. Definition of Server Class

4.5.3 BaseStation Class

The BaseStation class defines a set of attributes that help to store and track data about an instance of this class. Also, this class defines getters and setters' methods used to update and retrieve these attributes values. The class defined other methods which enables the class instance initiate the process of executing all request mobile user connected to the station, check if all requests sent via the station has been completed. Figure 19 below show the class implementation

```
In [21]: class BaseStation:
    def __init__(self, latitude, longitude):
        self.id = get_id(bs_id_init)
        self.latitude = latitude
        self.longitude = longitude
        self.x_axis = None
        self.y_axis = None
        self.radius = None
        self.transmission_rate = None
        self.downtime_latency = None
        self.roundtrip_latency = None,
        self.request_queue = list()
        self.connected_user = list()
        self.servers = list()
        self.compute_xy_coords()
        self.set_transmission_rate()
        self.set_downtime_latency()
        self.set_roundtrip_latency()

    def get_id(self):
        return self.id

    def get_latitude(self):
        return self.latitude

    def get_longitude(self):
        return self.longitude

    def set_x_axis(self, x_value):
        self.x_axis = x_value

    def get_x_axis(self):
        return self.x_axis

    def set_y_axis(self, y_value):
        self.y_axis = y_value

    def get_y_axis(self):
        return self.y_axis

    def set_radius(self):
        self.radius = uniform(0.3, 0.4)

    def get_radius(self):
        return self.radius

    def compute_xy_coords(self):
        x, y = compute_xy_coords_latlng(self.get_latitude(), self.get_longitude())
        self.set_x_axis(x)
        self.set_y_axis(y)
        self.set_radius()

    def set_transmission_rate(self):
        self.transmission_rate = randint(3, 10)
```

```python
    def get_transmission_rate(self):
        return self.transmission_rate

    def set_downtime_latency(self):
        self.downtime_latency = 0

    def get_downtime_latency(self):
        return self.downtime_latency

    def set_roundtrip_latency(self):
        self.roundtrip_latency = 0

    def get_roundtrip_latency(self):
        return self.roundtrip_latency

    def add_request_to_queue(self, request):
        self.request_queue.append(request)

    def remove_request_from_queue(self, request):
        self.request_queue.remove(request)

    def empty_request_queue(self):
        self.request_queue = list()

    def get_all_requests_from_queue(self):
        return self.request_queue

    def get_request_from_queue(self, requests, req_id):
        return get_request_by_id(requests, req_id)

    def add_server_to_station(self, server):
        self.servers.append(server)

    def remove_server_from_station(self, server):
        self.servers.remove(server)

    def empty_server_from_station(self):
        self.servers = list()

    def add_user(self, user):
        self.connected_user.append(user)

    def remove_user(self, user):
        self.connected_user.remove(user)

    def get_connected_users(self):
        return self.connected_user

    def reset_all(self):
        self.request_queue = list()
        self.connected_user = list()
        self.servers = list()

    def get_all_servers_from_station(self):
        return self.servers

    def get_server_from_station(self, servers, serv_id):
        return get_server_by_id(servers, serv_id)
```

Figure 19:. Definition of BaseStation Class

4.5.4 Request Class

The Request class defines a set of attributes that help to store and track data about an instance of this class and also defines getters and setters' methods used to update and retrieve these data. The class also defined another method which is invoked when the request instance has been executed to track to facilitate the update process for the target instance of this class.

```
In [22]: class Request:
    def __init__(self):
        self.id = get_id(reqs_id_init)
        self.belongs_to = None
        self.request_data_size = None
        self.response_data_size = None
        self.execution_time = None
        self.load_weight = None
        self.status = None
        self.cpu_usage = None
        data_size = randint(5, 20)
        self.set_request_data_size(data_size)
        self.set_response_data_size(data_size)
        self.set_load_weight()
        self.set_status(Status.PENDING)

    def get_id(self):
        return self.id

    def set_belongs_to(self, user_id):
        self.belongs_to = user_id

    def get_belongs_to(self):
        return self.belongs_to

    def set_request_data_size(self, request_data_size):
        self.request_data_size = request_data_size

    def get_request_data_size(self):
        return self.request_data_size

    def set_response_data_size(self, response_data_size):
        self.response_data_size = response_data_size

    def get_response_data_size(self):
        return self.response_data_size

    def set_execution_time(self, execution_time):
        self.execution_time = execution_time

    def get_execution_time(self):
        return self.execution_time

    def set_load_weight(self):
        self.load_weight = randint(1, 50)

    def get_load_weight(self):
        return self.load_weight

    def set_status(self, status):
        self.status = status

    def get_status(self):
        return self.status

    def set_cpu_usage(self, cpu_usage):
        self.cpu_usage = cpu_usage

    def get_cpu_usage(self):
        return self.cpu_usage

    def update_request_execution(self, transmission_rate, server_response_time, roundtrip_latency, downtime_latency):
        time_taken_to_upload_data = self.get_request_data_size() / transmission_rate
        time_taken_to_download_server_response = self.get_response_data_size() / transmission_rate

        time_taken_to_execute_request = time_taken_to_upload_data + server_response_time + time_taken_to_download_server_response

        self.set_execution_time(time_taken_to_execute_request)
        self.set_status(Status.COMPLETED)
```

Figure 20:. Definition of Request Class

4.6 Algorithm Implementation

4.6.1 Gradient Descent Algorithm Implementation

The Gradient descent algorithm was implemented to evaluate the optimal BaseStation instance a mobile user can connect to. The algorithm accepts two arguments namely the genome (chromosome) which is the output from Genetic algorithm and a list of base stations the user is within its radius. Setting up the given argument as a linear function, using the line equation, we attempt to find the best fit line and the use the equation to estimate the optimal base station the user can connect to.

```
In [24]: def SISP_GD(genome, station):

             x = np.array(genome)
             y = np.array(set_y_value_for_best_fit_genome(genome, station))

             target = None

             s_curr = i_curr = 0
             iterations = 1000
             n = len(x)
             learning_rate = 0.08
             cost = 0
             prev_cost = 100000000000000

             pred_i = None
             min_y_pred = None


             for i in range(iterations):
                 y_predicted = s_curr * x + i_curr
                 prev_cost = cost
                 cost = (1/n) * sum([val**2 for val in (y - y_predicted)])
                 s_derv = -(2/n) * sum(x * (y-y_predicted))
                 i_derv = -(2/n) * sum((y-y_predicted))
                 s_curr = s_curr - learning_rate * s_derv
                 i_curr = i_curr - learning_rate * i_derv

                 if math.isclose(prev_cost, cost):
                     break

             for i, base_station in enumerate(station):
                 if genome[i] == 1:
                     y_pred = s_curr * x[i] + i_curr
                     if min_y_pred == None :
                         min_y_pred = y_pred
                         pred_i = i
                     else:
                         if (y_pred < min_y_pred) and (not(min_y_pred == 0)):
                             min_y_pred = y_pred
                             pred_i = i

             if not pred_i == None:
                 return station[pred_i]["instance"]

             return None
```

Figure 21:. Gradient Descent Algorithm implementation code

4.6.2 Genetic Algorithm Implementation

The Genetic algorithm was implemented to evaluate the neighbourhood for the optimal solution by finding the genome (chromosome) with the fittest gene information. This the algorithm attempt to archive by implementing the following functionalities as show in Figure 22 through Figure 24 below:

```
In [25]: Genome = List[int]
         Population = List[Genome]
         FitnessFunc = Callable[[Genome], List[Request]]
         PopulateFunc = Callable[[], Population]
         SelectionFunc = Callable[[Population, FitnessFunc], Tuple[Genome, Genome]]
         CrossoverFunc = Callable[[Genome, Genome], Tuple[Genome, Genome]]
         MutationFunc = Callable[[Genome], Genome]
         SISPBaseStation = List[NestedDict]
```

```
In [26]: def generate_genome(length: int) -> Genome:
             return choices([0,1], k=length)
```

```
In [27]: def generate_population(size: int, genome_length: int) -> Population:
             return [generate_genome(genome_length) for _ in range(size)]
```

```
In [28]: def fitness(genome: Genome, base_stations: [SISPBaseStation], requests: [Request]) -> int:

             if len(genome) != len(base_stations):
                 raise ValueError("genome and things must be of the same length")

             total_distance = get_total_distance(base_stations, genome)
             max_val = 0

             for i, base_station in enumerate(base_stations):
                 val = 0
                 if genome[i] == 1:
                     val = 1/(base_station["distance"]/total_distance)
                     if val > max_val:
                         max_val = val

             return max_val
```

```
In [29]: def selection_pair(population: Population, fitness_func: FitnessFunc) -> Population:

             pop = choices(
                 population=population,
                 weights=[fitness_func(genome) for genome in population],
                 k=2
             )

             return pop
```

Figure 22:. Implementation of functions to generate a Genome, generate Population, Fitness of a genome and a select function for the Genetic Algorithm implementation

```
In [30]: def single_point_crossover(a: Genome, b: Genome) -> Tuple[Genome, Genome]:
             if len(a) != len(b):
                 raise ValueError("Genomes a and b must be of same length")

             length = len(a)
             if length < 2:
                 return a, b

             p = randint(1, length-1)

             return a[0:p] + b[p:], b[0:p] + a[p:]
```

```
In [31]: def mutation(genome: Genome, num: int=1, probability: float=0.5) -> Genome:
             for _ in range(num):
                 index = randrange(len(genome))
                 genome[index] = genome[index] if random() > probability else abs(genome[index] -1)

             return genome
```

```
In [32]: def set_y_value_for_best_fit_genome(genome: Genome, base_stations: [SISPBaseStation]) -> List[float]:
             result = []

             for i, base_station in enumerate(base_stations):
                 if genome[i] == 1:
                     result += [base_station["distance"]]
                 else:
                     result += [0]

             return result
```

Figure 23:. Implementation of crossover function, and mutation function

Figure 24 below shows the copulation of the Genetic Algorithm by pulling together all the component functionality defined to facilitate the search the best and optimal viable genome which is feed into the Gradient Descent algorithm.

```
In [33]: def SISP(
    mobile_user,
    stations,
    populate_func: PopulateFunc,
    fitness_func: FitnessFunc,
    fitness_limit: int,
    selection_func: SelectionFunc = selection_pair,
    crossover_func: CrossoverFunc = single_point_crossover,
    mutation_func: MutationFunc = mutation,
    generation_limit: int = 100
) -> Tuple[Population, int]:

    population = populate_func()

    for i in range(generation_limit):
        population = sorted(
            population,
            key = lambda genome: fitness_func(genome),
            reverse = True
        )

        if fitness_func(population[0]) >= fitness_limit:
            break

        next_generation = population[0:2]

        for j in range(int(len(population)/2) - 1):
            parents = selection_func(population, fitness_func)
            offspring_a, offspring_b = crossover_func(parents[0], parents[1])
            offspring_a = mutation_func(offspring_a)
            offspring_b = mutation_func(offspring_b)
            next_generation += [offspring_a, offspring_b]

        population = next_generation

    population = sorted(
        population,
        key = lambda genome: fitness_func(genome),
        reverse = True
    )

    target_station = SISP_GD(population[0], mobile_user.connect_to_base_station(stations))

    if target_station == None:
        print("Failed to connect to BS")
    else:
        #print("Connected to BS ***")
        mobile_user.set_connected_to(target_station.get_id())
        station = get_station_by_id(stations, target_station.get_id())

        if not station == None:
            station.add_user(mobile_user.get_id())
```

Figure 24:. Copulation of the component element that makes up the Genetic Algorithms

4.7 Run Experiment and Result

This section will show the code implementation used to run the experiment and to display the outputs as the experiment will product output when such output is solicited.

4.7.1 Run Experiment

The code snippet show below is the implementation of function that actually pull together all the code written so far and also display output to the console when an experiment has been completed. A total of 8 experiment was perform iteratively using a loop. the function also attempts to product a summary for each experiment performed.

```
In [44]: def execute_experiment():

             experiment_summary_data = list()
             iterator_counter = 0
             class_count = 8

             for index,row in df_users.iterrows():
                 mobus_inst = MobileUser(df_users.loc[index, "LATITUDE"], df_users.loc[index, "LONGITUDE"])
                 user_instances.append(mobus_inst)

             exp_users_sample_sizes = get_sample_sizes_bounds(len(user_instances), class_count)


             for iter_count in range(len(exp_users_sample_sizes)):

                 server_instances = list()
                 request_instances = list()
                 station_instances = list()
                 reset_users_requests(user_instances)

                 if exp_users_sample_sizes[iter_count] >= len(user_instances):
                     exp_users = user_instances
                 else:
                     exp_users = select_users_randomly(user_instances, exp_users_sample_sizes[iter_count])

                 for index,row in df_base_stations.iterrows():
                     station_inst = BaseStation(df_base_stations.loc[index, "LATITUDE"], df_base_stations.loc[index, "LONG
                     station_instances.append(station_inst)

                 for station in station_instances:
                     server_count = randint(2, 5)
                     for count in range(server_count):
                         server = Server()
                         server.set_base_station(station.get_id())
                         station.add_server_to_station(server.get_id())
                         server_instances.append(server)

                 for user in exp_users:
                     request_count = randint(3, 6)
                     for count in range(request_count):
                         request = Request()
                         request.set_belongs_to(user.get_id())
                         user.add_request(request.get_id())
                         request_instances.append(request)

        for user in exp_users:
            SISP(
                user,
                station_instances,
                populate_func = partial(
                    generate_population, size=10, genome_length=len(user.connect_to_base_station(station_instance
                ),
                fitness_func = partial(
                    fitness, base_stations=user.connect_to_base_station(station_instances), requests=user.get_all
                ),
                fitness_limit = 35,
                generation_limit=100
            )

        for station in station_instances:
            if len(station.get_connected_users()) > 0:
                station.process_all_requests_for_servers(user_instances, server_instances, request_instances, len

        base_station_count, server_count = get_used_base_stations_used(station_instances)
        average_cpu_usage, average_response_time = compute_averages_for_all_request(request_instances)
        experiment_summary_data.append([len(exp_users), base_station_count, server_count, len(request_instances),


        print()
        print()
        print()
        print("Experiment Summary for Experiment " + str(iterator_counter + 1))
        print("=====================================================================================")
        print()
        print("Total User Count                         : " + str(len(exp_users)))
        print("Total Base Station Count                 : " + str(base_station_count))
        print("Total Server Count                       : " + str(server_count))
        print("Total Request Count                      : " + str(len(request_instances)))
        print("Processed Request Count                  : " + str(get_completed_request_count(request_instances)))
        print("Average Response Time                    : " + str(round(average_response_time, 2)))
        print("Average CPU Usage                        : " + str(round(average_cpu_usage, 2))+" %")
        print()
        print()
        print()

        iterator_counter += 1

    return experiment_summary_data
```

Figure 25:. Implementation of execute experiment function

Figure 26 below shows the invocation of execute experiment function. The figure also shows a summary output for an experiment.

```
In [45]: experiment_output = execute_experiment()

         Experiment Summary for Experiment 1
         ==================================================================================

         Total User Count                      : 102
         Total Base Station Count              : 19
         Total Server Count                    : 69
         Total Request Count                   : 457
         Processed Request Count               : 457
         Average Response Time                 : 4.64
         Average CPU Usage                     : 70.78 %
```

Figure 26:. Code snippet show the invocation of execute experiment functionality

4.7.2 Presentation of experiment results

The section will present the visual to summary the experiment output. The code snippets shown below does exactly just that.

Figure 27 below show the code snippet use to add cell data to the summary table used to display the summary or outcome for the entire experiment after execution.

```
In [42]: def append_tatble_content(content, position):
             cell_data = "| "

             pad_diff_in_string_len = 12 - len(content)

             cell_data += content

             for i in range(pad_diff_in_string_len):
                 cell_data += " "

             if position == 1:
                 cell_data += " "
             elif position == 7:
                 cell_data += "|"
                 cell_data += "\n"
             else:
                 cell_data += " "

             return cell_data
```

Figure 27:. Implementation of a function used to append a cell to summary table for our experiment

Figure 28 below show the code snippet use to generate the summary table used to for the entire 8 experiment.

```
In [73]: def display_experiment_summary_in_tabular_view(output):
    table_designed = ""
    cell_size = 11
    table_width = 105
    new_line = "\n"
    users_count = servers_count = stations_count = requests_count = cpu_count = response_time_count = 0

    table_width_bar = ""
    for bar in range(table_width):
        table_width_bar += "="

    table_width_bar += new_line
    table_designed += table_width_bar
    table_designed += append_tatble_content("Iter", 1) + append_tatble_content("Users", 2) + append_tatble_content
    table_designed += table_width_bar

    for index in range(len(output)):
        users_count += output[index][0]
        servers_count += output[index][2]
        stations_count += output[index][1]
        requests_count += output[index][3]
        cpu_count += output[index][4]
        response_time_count += output[index][5]
        table_designed += append_tatble_content(str(index+1), 1) + append_tatble_content(str(output[index][0]), 2)
        table_designed += table_width_bar

    table_designed += append_tatble_content("TOTAL", 1) + append_tatble_content(str(users_count), 2) + append_tatbl
    table_designed += table_width_bar
    table_designed += append_tatble_content("Average", 1) + append_tatble_content("", 2) + append_tatble_content("'
    table_designed += table_width_bar
    print()
    print()
    print()
    print()
    print(table_designed)
    print()
    print()
    print()
    print()
```

Figure 28:. Generate summary table for the entire experiment function

Figure 29 below shows the invocation of the function display experiment summary table. The figure also shows a summary output for an experiment.

```
In [74]: display_experiment_summary_in_tabular_view(experiment_output)
```

Iter	Users	EdegCells	Servers	Requests	AvResTime	AvCPUUsage
1	102	19	69	457	4.64	70.78 %
2	204	21	68	892	5.41	73.33 %
3	306	30	117	1360	6.71	76.96 %
4	408	34	122	1838	7.27	80.54 %
5	510	29	111	2282	9.48	82.49 %
6	612	28	106	2773	10.52	85.89 %
7	714	35	114	3244	13.15	87.08 %
8	816	34	128	3689	15.51	89.6 %
TOTAL	3672	230	835	16535	72.7	646.67
Average					9.09	80.83 %

Figure 29:. Code snippet showing the invocation display summary table and output of the invoked function

Figure 30 below show a code snippet used generate the data that will be used to visualize the out of the experiment on line graph.

```
In [70]: avrestime_x = list()
         avrestime_y = list()

         server_x = list()
         avcpuusage_y = list()

         count = len(experiment_output)


         for index in range(count):
             avrestime_x.append(experiment_output[index][0])
             avrestime_y.append(round(experiment_output[index][5], 2))

             server_x.append(experiment_output[index][3])
             avcpuusage_y.append(round(experiment_output[index][4], 2))
```

Figure 30:. Code snippet used to generate data for plotting the graphs

Figure 31 and Figure 32 show the output of the experiment performed visually on a graph.

```
In [71]: circle = mpath.Path.unit_circle()

         plt.rcParams["figure.figsize"] = [12, 7]
         plt.rcParams["figure.autolayout"] = True

         plt.plot(avrestime_x, avrestime_y, '--r', marker=circle, markersize=5)
         plt.title("Performance comparisons of average response time with respect to mobile users connected")
         plt.xlabel("Number of Users")
         plt.ylabel("Average Response Time")
         # plt.legend([''])
         plt.show()
```

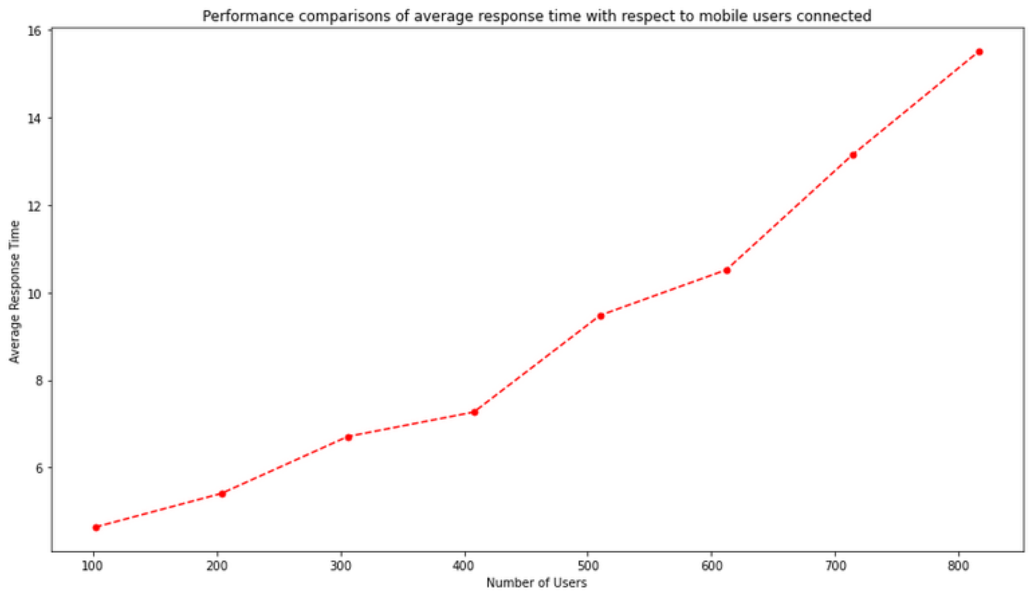

Figure 31:. Code snippet used to plot a line graph showing the relationship between number of users and the average response time

```
In [72]: circle = mpath.Path.unit_circle()

         plt.rcParams["figure.figsize"] = [12, 7]
         plt.rcParams["figure.autolayout"] = True

         plt.plot(server_x, avcpuusage_y, '--r', marker=circle, markersize=5)
         plt.title("Server overhead analysis using number of requests to cpu usage")
         plt.xlabel("Number of Request")
         plt.ylabel("Average CPU Usage")
         # plt.legend([''])
         plt.show()
```

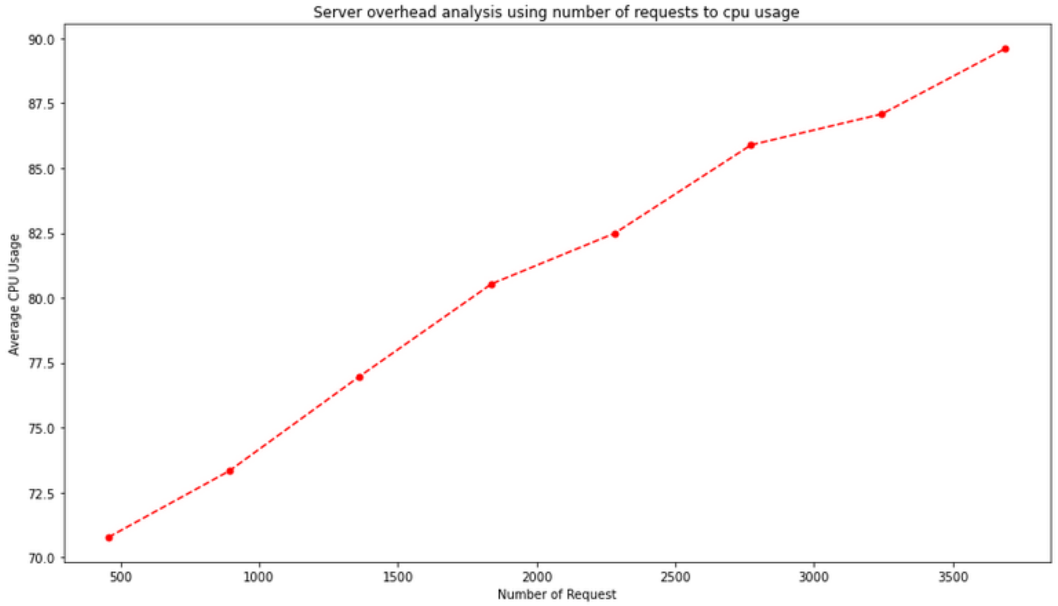

Figure 32:. Code snippet used to plot a line graph showing the relationship between number of request and the average CPU usage

5 Conclusion

This configuration manual has been designed for anyone to serve as a guide in working on this same project, implementing the same data structure as the code are tested and they work perfectly fine. In this way, the code demonstrated above was used to achieve the main aim of the research work.

6 References

DigitalOcean. 2021. *How To Install the Anaconda Python Distribution on Ubuntu 20.04 | DigitalOcean*. [online] Available at: https://www.digitalocean.com/community/tutorials/how-to-install-the-anaconda-python-distribution-on-ubuntu-20-04 [Accessed 15 December 2021].

En.wikipedia.org. 2021. *Anaconda (Python distribution) - Wikipedia*. [online] Available at: https://en.wikipedia.org/wiki/Anaconda_(Python_distribution) [Accessed 15 December 2021].

GitHub. 2021. *GitHub - swinedge/eua-dataset: Edge server, user dataset for Edge Computing research*. [online] Available at: https://github.com/swinedge/eua-dataset [Accessed 15 December 2021].

Jupyter.org. 2021. *Project Jupyter*. [online] Available at: https://jupyter.org [Accessed 15 December 2021].