Configuration Manual

MSc Research Project
MSc in Cloud Computing

Prathista Santhosh Kumar Shetty
x21146802

School of Computing
National College of Ireland

Supervisor: Vikas Sahni

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Prathista Santhosh Kumar Shetty
Student ID:	X21146802
Programme:	MSc in Cloud Computing **Year:** 2022
Module:	MSc Research Project
Lecturer:	Vikas Sahni
Submission Due Date:	15th December 2022
Project Title:	A New Offloading Technique Based on Deep Learning for Mobile-Edge Computing

Word Count: 625 **Page Count:** 8

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: Prathista Santhosh Kumar Shetty

Date: 15/12/2022

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

Prathista Santhosh Kumar Shetty
x21146802

1 System Configuration

The newly proposed algorithm was implemented on Google Colab cloud environment. The following setup was used at the time of execution:

- GPU: Nvidia with 12 GB memory
- Connected to Python 3
- RAM: 1.20 GB used from 25.45 GB availability
- Disk: 22.93 GB used from 166.77 GB

2 Libraries

The libraries installed on Google Colab is mentioned in the below table:

Libraries
Python 3
TensorFlow 2
Numpy
Scipy
Matplotlib

The libraries are imported as shown in the below figure:

3 Datasets

All the training data sets and testing data sets are contained in the folder name data. The dataset is generated by using the coordinate descent method mentioned in (Bi et al. 2018) when the optimization.py file is executed. This includes the data in .mat format with different number of wireless devices which randomly trains and sends the previously trained memory to deep neural network in every iteration with different time frame.

```
def cd_method(h):
    N = len(h)
    M0 = np.random.randint(2,size = N)
    gain0,a,Tj= bisection(h,M0)
    g_list = []
    M_list = []
    while True:
        for j in range(0,N):
            M = np.copy(M0)
            M[j] = (M[j]+1)%2
            gain,a,Tj= bisection(h,M)
            g_list.append(gain)
            M_list.append(M)
        g_max = max(g_list)
        if g_max > gain0:
            gain0 = g_max
            M0 = M_list[g_list.index(g_max)]
        else:
            break
    return gain0, M0
```

4 Reproducing the algorithm

In this research, the algorithm is evaluated with two different experiments. In one algorithm, we have considered that the number of networks is equal to the number of tasks and the other has different number of wireless devices having varying weights. The methods to execute of these algorithms is explained below:

- To reproduce the algorithm when the parameters are same for K and N, the python file DRL_offloading.py should be executed using the command python3 DRL_offloading.py.

```python
# Model Training

start_time=time.time()

rate_his = []
rate_his_ratio = []
mode_his = []
k_idx_his = []
K_his = []
for i in range(n):
    if i % (n//10) == 0:
        print("%0.1f"%(i/n))
    if i> 0 and i % Delta == 0:
        # index counts from 0
        if Delta > 1:
            max_k = max(k_idx_his[-Delta:-1]) +1;
        else:
            max_k = k_idx_his[-1] +1;
        K = min(max_k +1, N)

    if i < n - num_test:
        # training
        i_idx = i % split_idx
    else:
        # test
        i_idx = i - n + num_test + split_idx

    h = channel[i_idx,:]

    # the action will select KNN method
    m_list = mem.decode(h, K, decoder_mode)

    r_list = []
    for m in m_list:
        r_list.append(bisection(h/1000000, m)[0])

    # compute the largest reward
    mem.encode(h, m_list[np.argmax(r_list)])
    # training of algorithm ends here
```

The trained model shows the average normalized computation rate as 0.9996

```
1/1 [==============================] - 0s 14ms/step
1/1 [==============================] - 0s 15ms/step
1/1 [==============================] - 0s 14ms/step
1/1 [==============================] - 0s 15ms/step
1/1 [==============================] - 0s 15ms/step
1/1 [==============================] - 0s 15ms/step
1/1 [==============================] - 0s 15ms/step
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 15ms/step
Averaged normalized computation rate: 0.9996670236095926
Total time consumed:1635.3216528892517
Average time per channel:0.05451072176297505 6
```

- To reproduce the algorithm when the parameters has varying weights for K and N, the python file varying_weights.py should be executed using the command python3 DRL_offloading.py.

```python
start_time=time.time()

rate_his = []
rate_his_ratio = []
mode_his = []
k_idx_his = []
K_his = []
h = channel[0,:]

# code where the algorithm will have varying weights
weight, rate = alternate_weights(0)
print("WD weights at time frame %d:"%(0), weight)
```

The trained model shows the average normalized computation rate with varying weights as 0.9987.

```
1/1 [==============================] - 0s 15ms/step
1/1 [==============================] - 0s 14ms/step
1/1 [==============================] - 0s 14ms/step
1/1 [==============================] - 0s 14ms/step
1/1 [==============================] - 0s 14ms/step
1/1 [==============================] - 0s 15ms/step
1/1 [==============================] - 0s 15ms/step
1/1 [==============================] - 0s 15ms/step
1/1 [==============================] - 0s 14ms/step
Averaged normalized computation rate: 0.9987449242648451
Total time consumed:571.921135187149
Average time per channel:0.05719211351871491
```

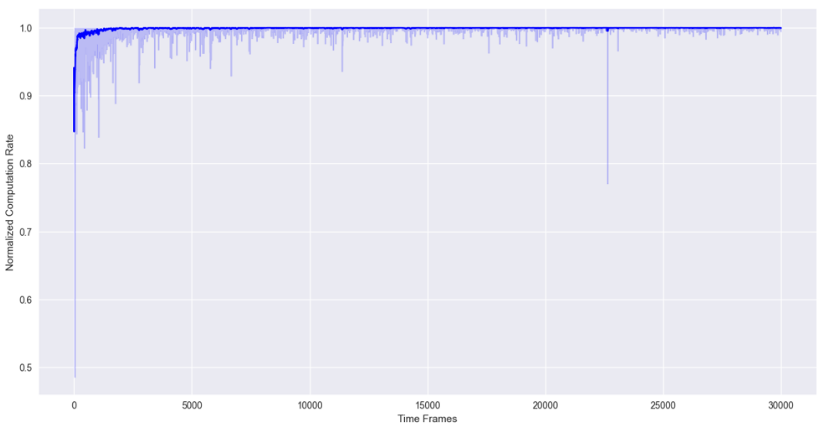
5 Plots

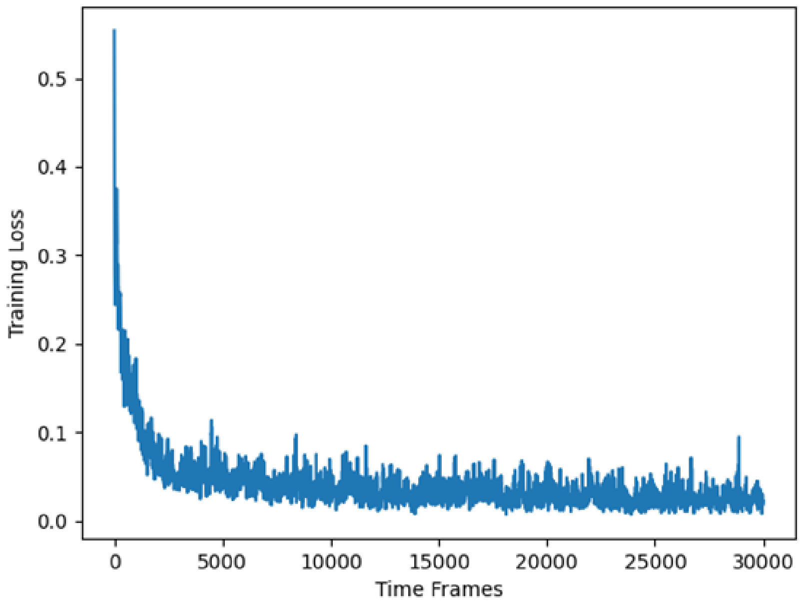
Below were the codes used to generate the graphs:

```
def plot_rate( rate_his, rolling_intv = 50):
    import matplotlib.pyplot as plt
    import pandas as pd
    import matplotlib as mpl

    rate_array = np.asarray(rate_his)
    df = pd.DataFrame(rate_his)


    mpl.style.use('seaborn')
    fig, ax = plt.subplots(figsize=(15,8))
#    rolling_intv = 20

    plt.plot(np.arange(len(rate_array))+1, np.hstack(df.rolling(rolling_intv, min_periods=1).mean().values), 'b')
    plt.fill_between(np.arange(len(rate_array))+1, np.hstack(df.rolling(rolling_intv, min_periods=1).min()[0].values), np.hstack(df.rolli
    plt.ylabel('Normalized Computation Rate')
    plt.xlabel('Time Frames')
    plt.show()
```

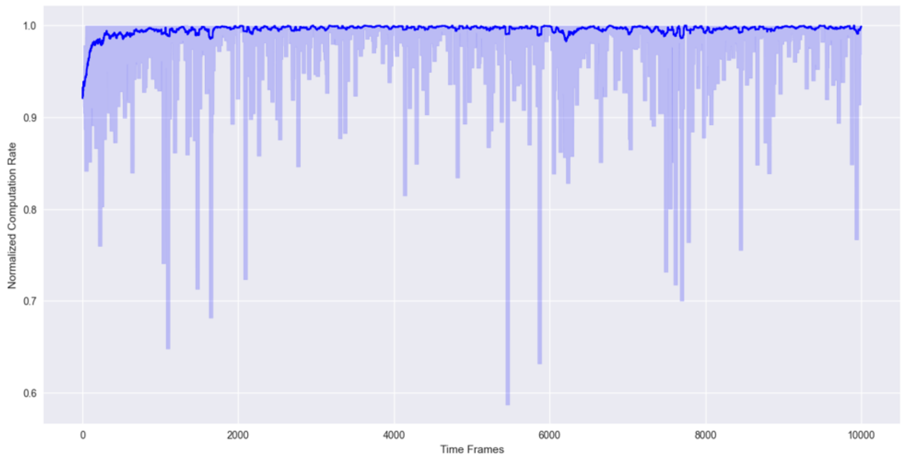
```
def plot_cost(self):
    import matplotlib.pyplot as plt
    plt.plot(np.arange(len(self.cost_his))*self.training_interval, self.cost_his)
    plt.ylabel('Training Loss')
    plt.xlabel('Time Frames')
    plt.show()
```

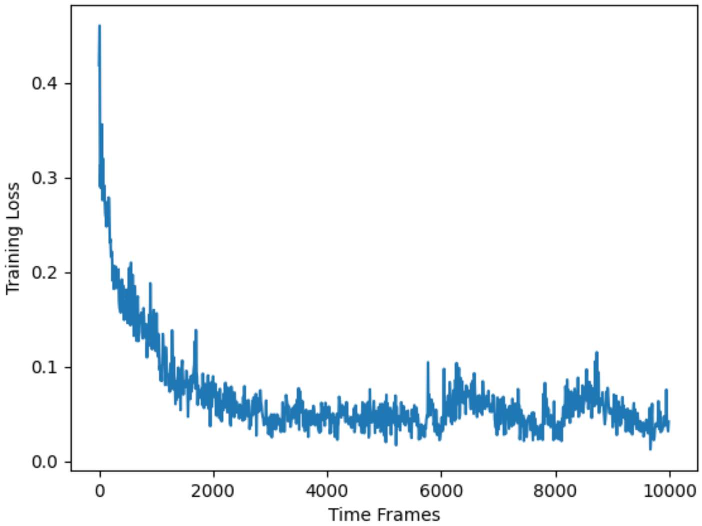
- When k = N, the below graphs were generated





- For varying weights, the below graphs were generated:





References

Bi, S. and Zhang, Y.J., 2018. Computation rate maximization for wireless powered mobile-edge computing with binary computation offloading. IEEE Transactions on Wireless Communications, 17(6), pp.4177-4190.