Configuration Manual

MSc Research Project
Cloud Computing

Shubham Kumbhar
Student ID: x22119345

School of Computing
National College of Ireland

Supervisor: Shaguna Gupta

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Shubham Kumbhar
Student ID:	x22119345
Programme:	Cloud Computing
Year:	2024
Module:	MSc Research Project
Supervisor:	Shaguna Gupta
Submission Due Date:	19/08/2024
Project Title:	Configuration Manual
Word Count:	XXX
Page Count:	45

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:	
Date:	18th August 2024

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

Shubham Kumbhar
x22119345

1 Requirements

1. **System Requirements**

A. Hardware Requirements

- CPU: Minimum 4 cores, 2.5GHz or higher
- RAM: Minimum 16 GB
- GPU: NVIDIA/AMD with 4 GB VRAM (for gaming and neural network training)
- Disk Space: Minimum 100 GB free space
- Network: Stable internet connection, preferably 5G for testing cloud gaming performance

B.Software Requiremnets

- Operating System: Windows 10/11, Linux (Ubuntu 20.04 or later), macOS 10.15 or later
- Python: Version 3.8 or higher
- IDE/Code Editor: Visual Studio (for local development and Python scripting)

C. Tools & Platforms:

- Google Colab (for cloud-based model training and visualization)
- AWS SageMaker (for large-scale model training and deployment)
- AWS Lambda (for serverless execution of Python scripts)
- AWS S3 (for data storage)
- Wireshark (for network data capture and analysis)
- Xbox Cloud Gaming (for cloud gaming sessions with Fortnite)
- Python Libraries:
  - TensorFlow / Keras (Neural Networks)
  - XGBoost
  - Scikit-learn (Random Forest, SVR)
  - Pandas, NumPy (Data Processing)
  - Matplotlib, Seaborn (Data Visualization)

2 Environmental Setup

2.1 Installation of Visual studio

I.Download from its offical website : `https://visualstudio.microsoft.com/downloads/`

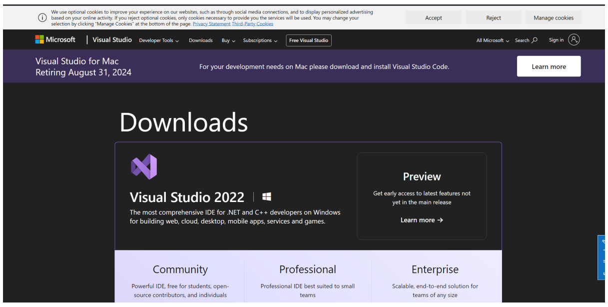

Figure 1: Visual stduio website

II. **Install the Python development environment within Visual Studio.**

2.2 Installing Python and Required Libraries

I. Install Python from `https://www.python.org`.

II. **Install the required Python libraries:**

- pip install tensorflow xgboost scikit-learn pandas numpy matplotlib seaborn
- pip install lightgbm
- pip install xgboost
- pip install numpy==1.23 tensorflow==2.11

2.3 Google Colab

A. Open Google Colab : `https://colab.google`

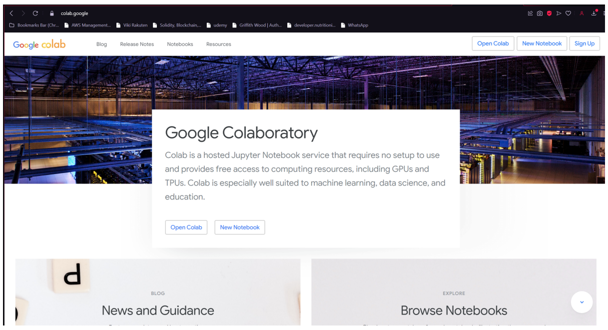

Figure 2: Google Colab

B. Create New Notebook

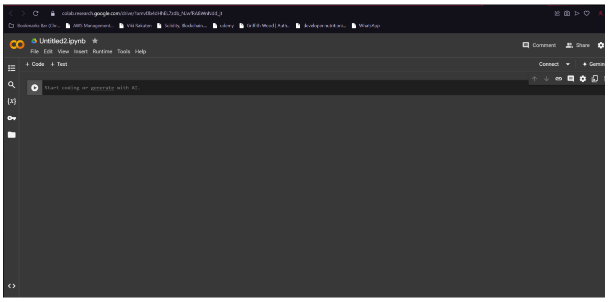

Figure 3: New Notebook

C. Import the necessary libraries in Colab notebooks:

- import tensorflow as tf
- import xgboost as xgb
- from sklearn.ensemble import RandomForestRegressor
- from sklearn.svm import SVR

2.4 Setting Up AWS Services

2.4.1 S3

- Create an S3 bucket to store your datasets, trained models, and results.
- Create two buckets

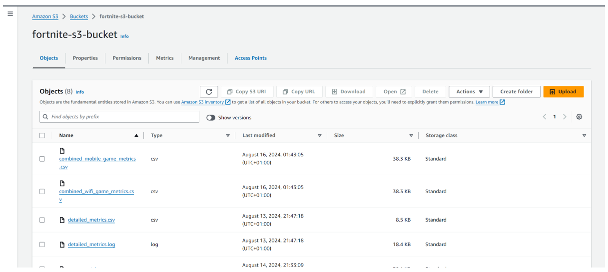

Figure 4: S3 bucket : fortnite-s3-bucket

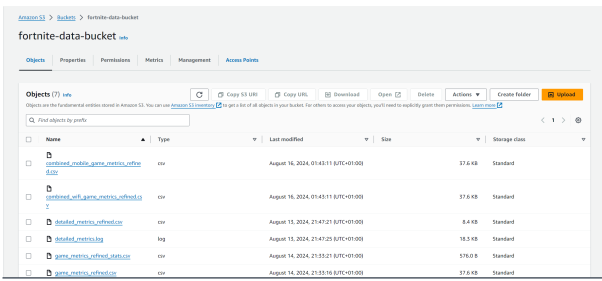

Figure 5: S3 buckect: fortnite-data-bucket

2.4.2 AWS Lambda

- Use Lambda to run serverless Python scripts.
- Deploy Python scripts to Lambda for tasks like data preprocessing or invoking models.

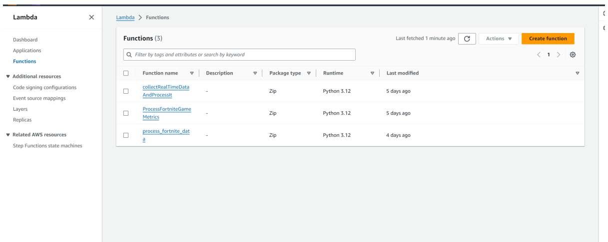

Figure 6: Creating Lambda Function

2.4.3 AWS SageMaker

- Use SageMaker for training machine learning models at scale.
- Refer to SageMaker Setup : `https://docs.aws.amazon.com/sagemaker/latest/dg/gs-console.html` for more details.
- Create two notebook for this research
  - 1. To perform action related to gameMetrics.csv (dataset)
  - 2. To perform action related to combined Dataset

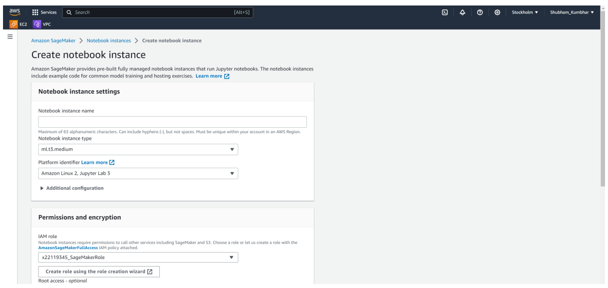

Figure 7: Creating SageMaker Notebook

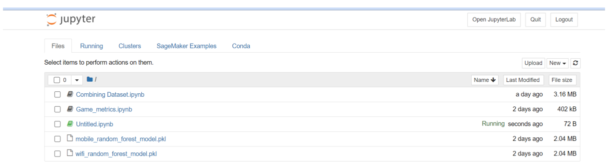

Figure 8: SageMaker jupter notebook open : see the list of notebook created

3 Game and Network Data Collection

3.1 Running game on Cloud Platform

- For this project we selected Fortnite game
- Run game on official website via xbox cloud gaming
- While playing game , run python script and wireshark in background to collect data.

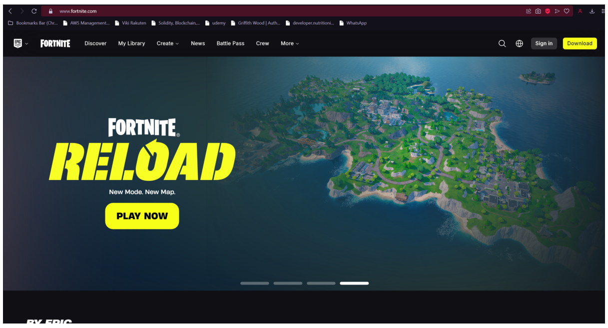

Figure 9: Fortnite Official Website dashboard

- Click on play button as show in 9 figure
- After that select xbox cloud gaming. 10

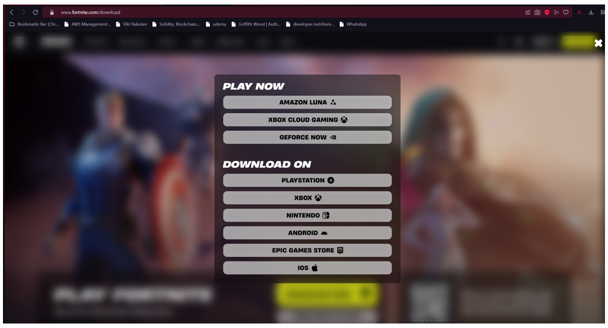

Figure 10: Gaming platform : Fortnite game to be played on

3.1.1 Visual Studio to collected game metrics data

- Run python script on visual studio in background while playing game
- After collecting data send it to S3 bucket

game_metrics.py > ...

```python
import psutil
import time
import GPUtil
from ping3 import ping
import boto3
import csv
import os
import speedtest

# Function to capture CPU usage
def get_cpu_usage():
    return psutil.cpu_percent(interval=1)

# Function to capture memory usage
def get_memory_usage():
    memory = psutil.virtual_memory()
    return memory.percent

# Function to capture GPU usage
def get_gpu_usage():
    gpus = GPUtil.getGPUs()
    if gpus:
        gpu = gpus[0]  # Assuming the first GPU
        return gpu.load * 100  # Convert to percentage
    return 0  # If no GPU is found, return 0

# Function to capture network latency (ping to a server)
def get_network_latency(host='www.fortnite.com'):
    latency = ping(host)
    if latency is not None:
        return latency * 1000  # Convert to milliseconds
    else:
        return None  # If ping fails

# Function to capture disk I/O usage
def get_disk_io():
    disk_io = psutil.disk_io_counters()
    return disk_io.read_bytes + disk_io.write_bytes  # Total I/O in bytes

# Function to collect actual jitter
def get_jitter(host='www.fortnite.com', count=5, interval=0.2):
    latencies = []
    for _ in range(count):
        latency = ping(host)
        if latency is not None:
            latencies.append(latency * 1000)  # Convert to milliseconds
        time.sleep(interval)

    if len(latencies) < 2:
        return None  # Not enough data to calculate jitter

    # Calculate jitter as the average of absolute differences between consecutive ping results
    jitter = sum(abs(latencies[i] - latencies[i-1]) for i in range(1, len(latencies))) / (len(latencies) - 1)
    return jitter

# Function to measure actual bandwidth usage
def get_bandwidth():
    try:
        st = speedtest.Speedtest()
        st.get_best_server()
        download_speed = st.download() / 1_000_000  # Convert from bits per second to Mbps
        upload_speed = st.upload() / 1_000_000  # Convert from bits per second to Mbps
        return download_speed, upload_speed
    except Exception as e:
        print(f"Error measuring bandwidth: {e}")
        return None, None

# Function to download a file from S3
def download_from_s3(s3_client, bucket, file_name, download_path):
    try:
        s3_client.download_file(bucket, file_name, download_path)
        print(f"File '{file_name}' downloaded successfully from '{bucket}'.")
    except Exception as e:
        print(f"Error downloading file '{file_name}': {e}")

# Function to upload a file to S3
def upload_to_s3(file_name, bucket, object_name=None):
    s3_client = boto3.client('s3')
    try:
        s3_client.upload_file(file_name, bucket, object_name or file_name)
        print(f"File '{file_name}' uploaded successfully to '{bucket}'.")
    except Exception as e:
        print(f"Error uploading file '{file_name}': {e}")
```

```python
# Function to capture and log data
def log_metrics(log_file='game_metrics.log', csv_file='game_metrics.csv', interval=3
, bucket_name='fortnite-s3-bucket'):
    s3_client = boto3.client('s3')

    # Check if files exist in S3 and download them
    if os.path.exists(log_file):
        os.remove(log_file)
    if os.path.exists(csv_file):
        os.remove(csv_file)

    download_from_s3(s3_client, bucket_name, log_file, log_file)
    download_from_s3(s3_client, bucket_name, csv_file, csv_file)

    # Append mode for both log and CSV files
    with open(log_file, 'a') as f, open(csv_file, 'a', newline='') as csvfile:
        fieldnames = ['Time', 'CPU Usage', 'Memory Usage', 'GPU Usage', 'Network Latency', 'Disk I/O', 'Jitter', 'Download Bandwidth',
        writer = csv.DictWriter(csvfile, fieldnames=fieldnames)

        # Write header only if the CSV file is empty
        if csvfile.tell() == 0:
            writer.writeheader()

        try:
            while True:
                cpu_usage = get_cpu_usage()
                memory_usage = get_memory_usage()
                gpu_usage = get_gpu_usage()
                network_latency = get_network_latency()
                disk_io = get_disk_io()
```

```python
                disk_io = get_disk_io()
                jitter = get_jitter()
                download_bandwidth, upload_bandwidth = get_bandwidth()

                log_data = {
                    "Time": time.strftime('%Y-%m-%d %H:%M:%S'),
                    "CPU Usage": cpu_usage,
                    "Memory Usage": memory_usage,
                    "GPU Usage": gpu_usage,
                    "Network Latency": network_latency,
                    "Disk I/O": disk_io,
                    "Jitter": jitter,
                    "Download Bandwidth": download_bandwidth,
                    "Upload Bandwidth": upload_bandwidth
                }

                log_line = ', '.join([f"{key}: {value}" for key, value in log_data.items()]) + '\n'
                print(log_line.strip())

                # Write to log file
                f.write(log_line)
                f.flush()  # Ensure data is written to disk

                # Write to CSV file
                writer.writerow(log_data)
                csvfile.flush()  # Ensure data is written to disk

                time.sleep(interval)  # Capture data every `interval` seconds
        except KeyboardInterrupt:
```

```python
                time.sleep(interval)  # Capture data every `interval` seconds
        except KeyboardInterrupt:
            print("Stopping data collection. Uploading log and CSV files to S3...")
            # Upload the log file to S3 and overwrite the existing file
            upload_to_s3(log_file, bucket_name, log_file)
            # Upload the CSV file to S3 and overwrite the existing file
            upload_to_s3(csv_file, bucket_name, csv_file)

if __name__ == '__main__':
    log_metrics(interval=0.5)
```

Figure 11: Python Script to collect game metrics data

```
Time: 2024-08-14 21:23:51, CPU Usage: 4.7, Memory Usage: 91.4, GPU Usage: 0.0, Network Latency: 23.645639419555664, Disk I/O: 2814089399808, Jitter: 5.453050136566162, Download Bandwidth: 336.1572383209611, Upload Bandwidth: 49.657084113341476
Time: 2024-08-14 21:24:15, CPU Usage: 4.3, Memory Usage: 91.5, GPU Usage: 0.0, Network Latency: 34.792423248291016, Disk I/O: 2814110117376, Jitter: 2.8319358825683594, Download Bandwidth: 325.31473298913227, Upload Bandwidth: 44.13202705554605
Time: 2024-08-14 21:24:38, CPU Usage: 4.8, Memory Usage: 91.2, GPU Usage: 0.0, Network Latency: 27.07958221435547, Disk I/O: 2814340582912, Jitter: 1.4553070068359375, Download Bandwidth: 380.0354251978254, Upload Bandwidth: 49.93538916643131
Time: 2024-08-14 21:25:10, CPU Usage: 4.0, Memory Usage: 91.1, GPU Usage: 0.0, Network Latency: 32.692909240722656, Disk I/O: 2814386254848, Jitter: 2.9726624488830566, Download Bandwidth: 142.8536564864819, Upload Bandwidth: 50.08260528091976
Time: 2024-08-14 21:25:34, CPU Usage: 5.0, Memory Usage: 88.4, GPU Usage: 0.0, Network Latency: 26.823043823242188, Disk I/O: 2814460259328, Jitter: 3.454744815826416, Download Bandwidth: 154.95725571388272, Upload Bandwidth: 49.45374966428738
Time: 2024-08-14 21:26:01, CPU Usage: 5.1, Memory Usage: 88.6, GPU Usage: 0.0, Network Latency: 25.960922241210938, Disk I/O: 2814505167872, Jitter: 2.1123886108398438, Download Bandwidth: 152.41374051415266, Upload Bandwidth: 10.14391142481117
Time: 2024-08-14 21:26:31, CPU Usage: 4.1, Memory Usage: 88.6, GPU Usage: 0.0, Network Latency: 24.274110794067383, Disk I/O: 2814533311488, Jitter: 5.537807941436768, Download Bandwidth: 200.46805145785265, Upload Bandwidth: 50.381565659550354
Time: 2024-08-14 21:26:55, CPU Usage: 4.1, Memory Usage: 88.7, GPU Usage: 0.0, Network Latency: 27.397871017456055, Disk I/O: 2814573401088, Jitter: 4.481971263885498, Download Bandwidth: 326.68122838413666, Upload Bandwidth: 50.18284925788117
Time: 2024-08-14 21:27:17, CPU Usage: 4.2, Memory Usage: 88.8, GPU Usage: 0.0, Network Latency: 29.14142608642578, Disk I/O: 2814599160832, Jitter: 2.6023387908935547, Download Bandwidth: 405.88198720641117, Upload Bandwidth: 49.79242237660506
Time: 2024-08-14 21:27:40, CPU Usage: 4.7, Memory Usage: 88.9, GPU Usage: 0.0, Network Latency: 31.897544860839844, Disk I/O: 2814625158656, Jitter: 2.6622414588928223, Download Bandwidth: 388.83499673380817, Upload Bandwidth: 50.36264609322505
Time: 2024-08-14 21:28:03, CPU Usage: 4.7, Memory Usage: 90.3, GPU Usage: 0.0, Network Latency: 31.455039978027344, Disk I/O: 2814822116352, Jitter: 5.512058734893799, Download Bandwidth: 380.8024835023475, Upload Bandwidth: 49.6660119499885
Time: 2024-08-14 21:28:28, CPU Usage: 4.7, Memory Usage: 93.8, GPU Usage: 0.0, Network Latency: 32.10186958312988, Disk I/O: 2815677351936, Jitter: 3.157496452331543, Download Bandwidth: 323.4713462743089, Upload Bandwidth: 49.1081069463292
Time: 2024-08-14 21:28:52, CPU Usage: 4.8, Memory Usage: 93.9, GPU Usage: 0.0, Network Latency: 24.983644485473633, Disk I/O: 2816450499072, Jitter: 2.9124021530151367, Download Bandwidth: 325.77205883604813, Upload Bandwidth: 50.343291024617706
Time: 2024-08-14 21:29:16, CPU Usage: 8.8, Memory Usage: 94.6, GPU Usage: 0.0, Network Latency: 26.1228084564209, Disk I/O: 2816785068544, Jitter: 0.9818077087402344, Download Bandwidth: 322.3589031522112, Upload Bandwidth: 50.415754867507786
Time: 2024-08-14 21:29:41, CPU Usage: 7.3, Memory Usage: 95.3, GPU Usage: 0.0, Network Latency: 34.57498550415039, Disk I/O: 2816985491968, Jitter: 5.479037761688232, Download Bandwidth: 221.81595026002503, Upload Bandwidth: 50.2757036394513
Time: 2024-08-14 21:30:06, CPU Usage: 7.1, Memory Usage: 95.6, GPU Usage: 0.0, Network Latency: 29.788494110107422, Disk I/O: 2817170190848, Jitter: 5.09035587310791, Download Bandwidth: 285.26425811340204, Upload Bandwidth: 50.35183100211468
Error measuring bandwidth: Unable to connect to servers to test latency.
Time: 2024-08-14 21:30:09, CPU Usage: 6.1, Memory Usage: 94.3, GPU Usage: 0.0, Network Latency: 37.00757026672363, Disk I/O: 2817357316608, Jitter: 1.8817782402038574, Download Bandwidth: None, Upload Bandwidth: None
Error measuring bandwidth: Unable to connect to servers to test latency.
Time: 2024-08-14 21:30:13, CPU Usage: 7.2, Memory Usage: 94.3, GPU Usage: 0.0, Network Latency: 32.63592720031738, Disk I/O: 2817388570112, Jitter: 2.870500087738037, Download Bandwidth: None, Upload Bandwidth: None
Time: 2024-08-14 21:30:37, CPU Usage: 7.8, Memory Usage: 94.4, GPU Usage: 0.0, Network Latency: 31.43477439880371, Disk I/O: 2817415263744, Jitter: 4.6874284744262695, Download Bandwidth: 265.0988981931841, Upload Bandwidth: 50.33443571973032
Time: 2024-08-14 21:31:01, CPU Usage: 7.4, Memory Usage: 95.2, GPU Usage: 0.0, Network Latency: 30.005931854248047, Disk I/O: 2817568228864, Jitter: 7.152557373046875, Download Bandwidth: 270.91564574877356, Upload Bandwidth: 49.89014080482966
Time: 2024-08-14 21:31:26, CPU Usage: 9.4, Memory Usage: 95.5, GPU Usage: 0.0, Network Latency: 29.5565128326416, Disk I/O: 2817758636544, Jitter: 2.2756457328796387, Download Bandwidth: 252.40785594324188, Upload Bandwidth: 47.75659110448995
Time: 2024-08-14 21:31:50, CPU Usage: 10.4, Memory Usage: 94.0, GPU Usage: 0.0, Network Latency: 33.45894813537598, Disk I/O: 2818027410432, Jitter: 1.7952322959899902, Download Bandwidth: 255.80404906171253, Upload Bandwidth: 48.611600120725214
Time: 2024-08-14 21:32:15, CPU Usage: 8.5, Memory Usage: 94.1, GPU Usage: 0.0, Network Latency: 28.14793586730957, Disk I/O: 2818254839808, Jitter: 7.157564163208008, Download Bandwidth: 233.95339708985728, Upload Bandwidth: 49.639029131282456
Time: 2024-08-14 21:32:39, CPU Usage: 5.0, Memory Usage: 95.5, GPU Usage: 0.0, Network Latency: 28.8698673248291, Disk I/O: 2818434415104, Jitter: 2.782583236694336, Download Bandwidth: 136.9792025345425, Upload Bandwidth: 49.464454608054396
Time: 2024-08-14 21:33:03, CPU Usage: 14.0, Memory Usage: 96.4, GPU Usage: 0.0, Network Latency: 30.10416030883789, Disk I/O: 2819065078784, Jitter: 1.407921314239502, Download Bandwidth: 244.06062733108206, Upload Bandwidth: 49.625002370801546
Stopping data collection. Uploading log and CSV files to S3...
File 'game_metrics.log' uploaded successfully to 'fortnite-s3-bucket'.
File 'game_metrics.csv' uploaded successfully to 'fortnite-s3-bucket'.
```

Figure 12: Snapshot of : collecting game metrics data

3.1.2 Wireshark

- Install Wireshark
- Open Wireshark
- Capture wifi connected data

- We captured two dataset during this resarch when connected to wifi (5G network connection) and other when connected to Mobile Hotspot (4G network connection)
- Collecting this data in background while running game

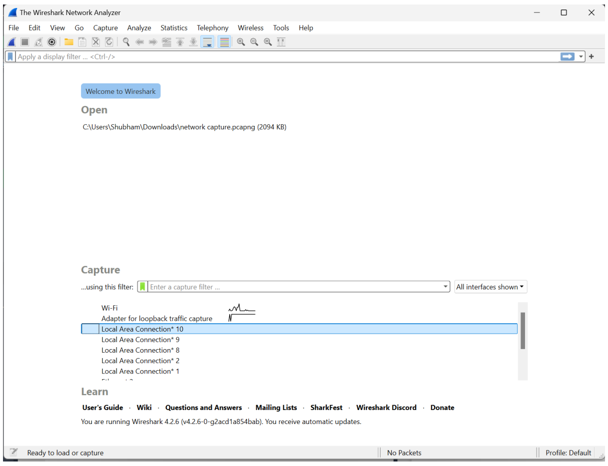

Figure 13: Open wireshark : we can see the list of connected network to capture data

4 AWS Cloud Services

4.1 AWS S3

- Once the data is collected by Python script and wireshark, that data is send to S3 bucket

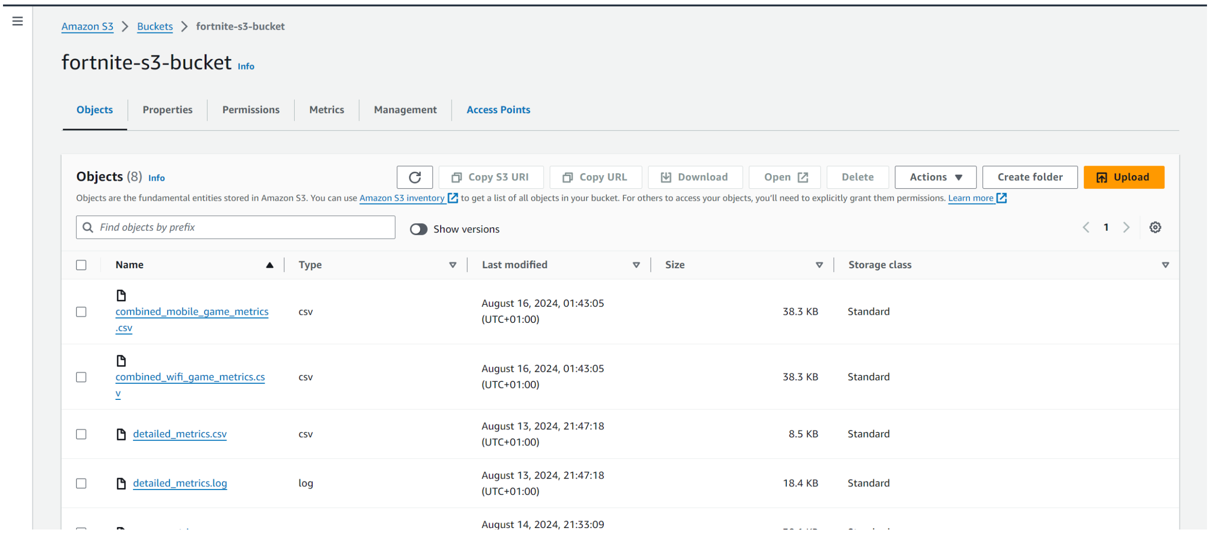

Figure 15: Files uploaded on S3 bucket

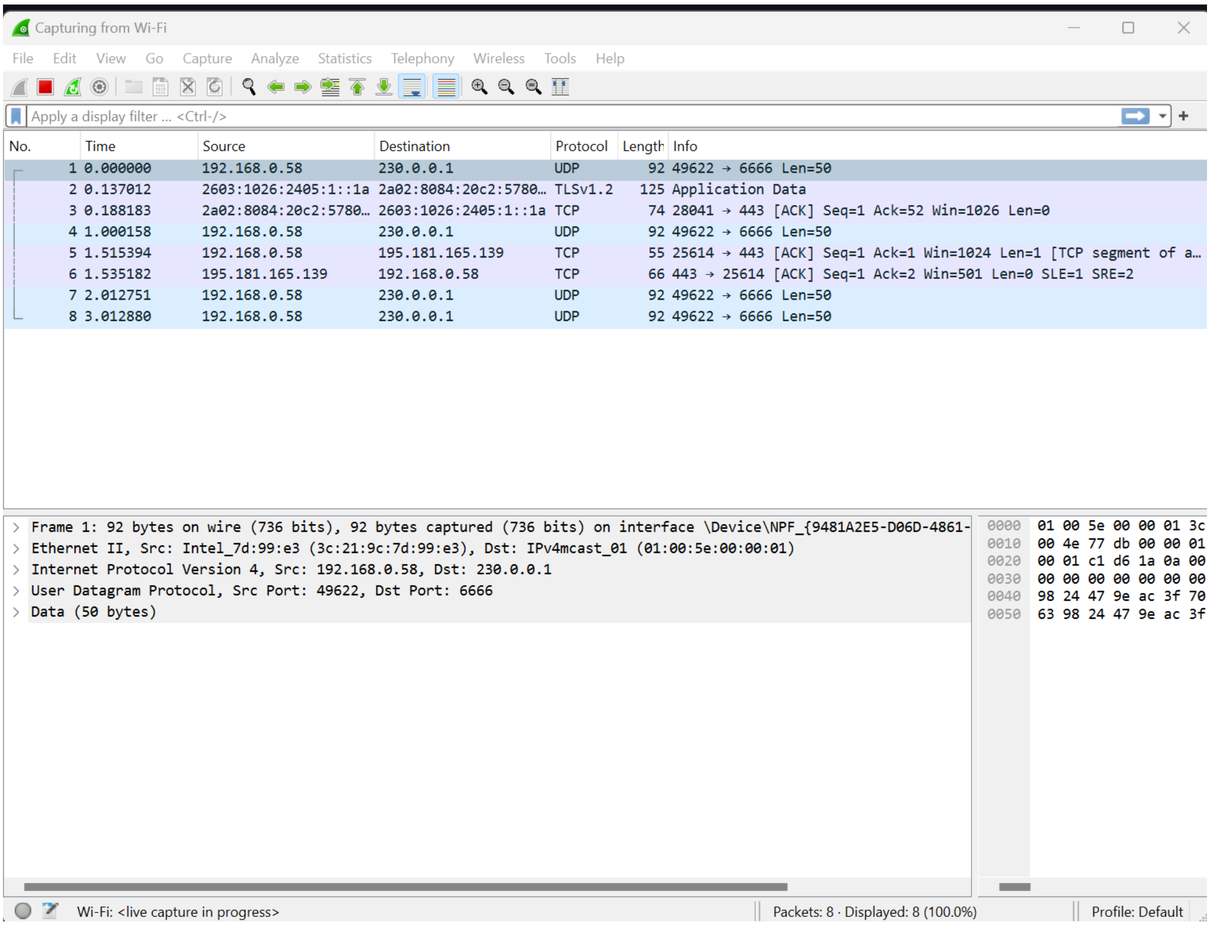

Figure 14: Capturing Network Data

4.1.1 AWS Lambda

- Create Lambda function and set a trigger when this function should be executed.
- We created two lambda function:
  - 1. To clean, process and refine raw dataset and store the new dataset in new S3 bucket.
  - 2. To compute the game metrics resources to find mean, median and variance.

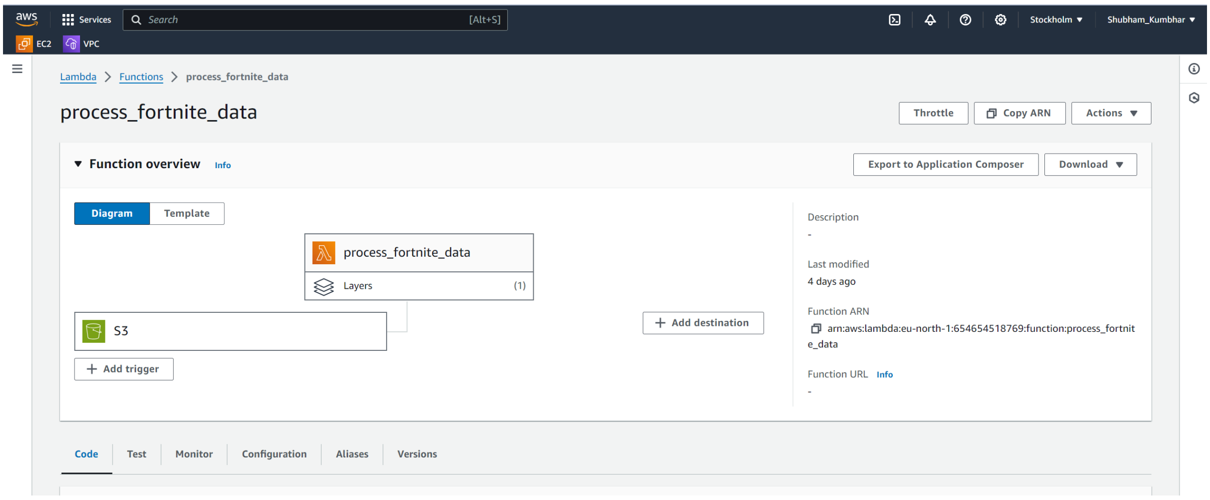

Figure 16: Lambda Function overview

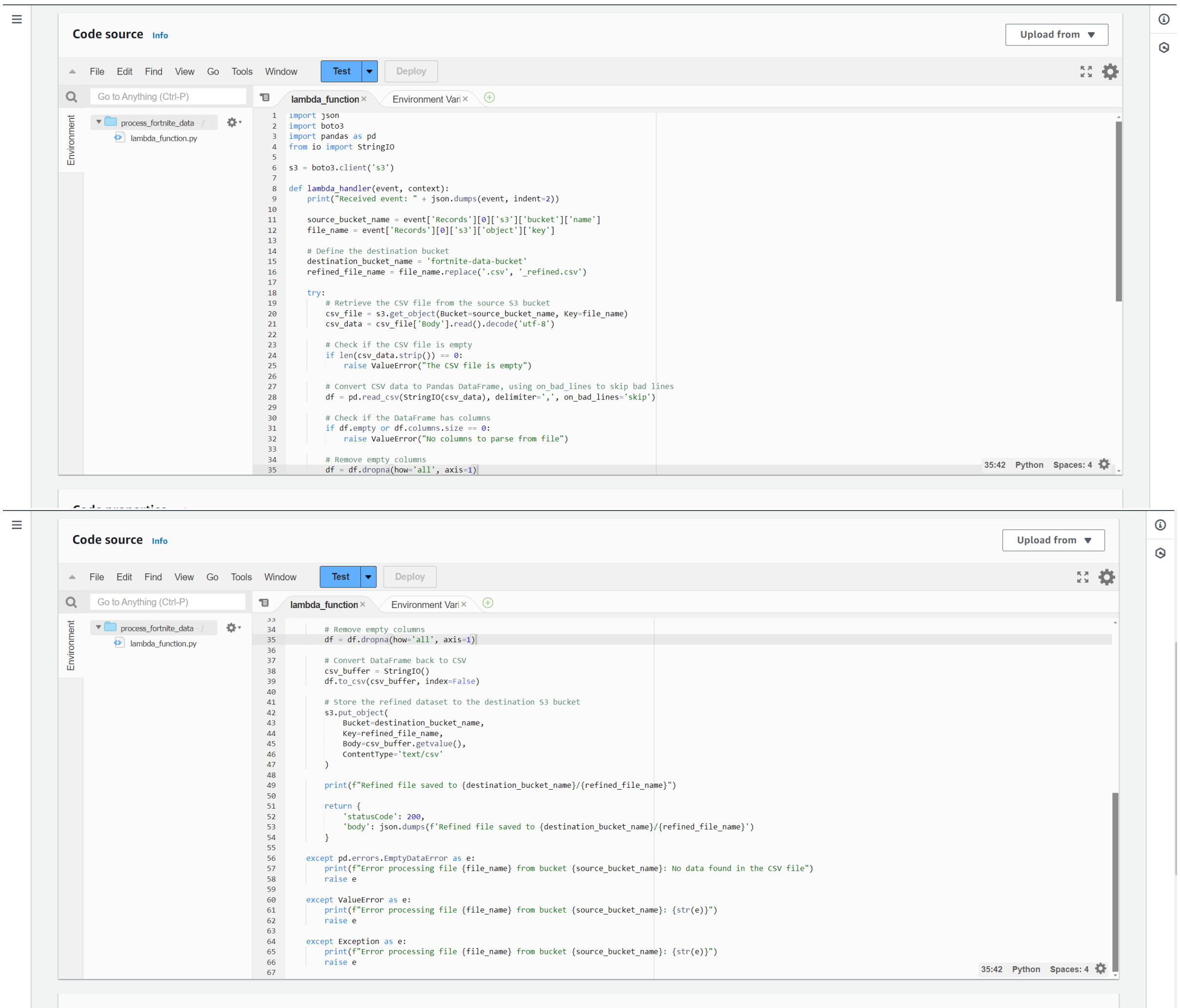

Figure 17: Lambda Function code : to refine raw dataset

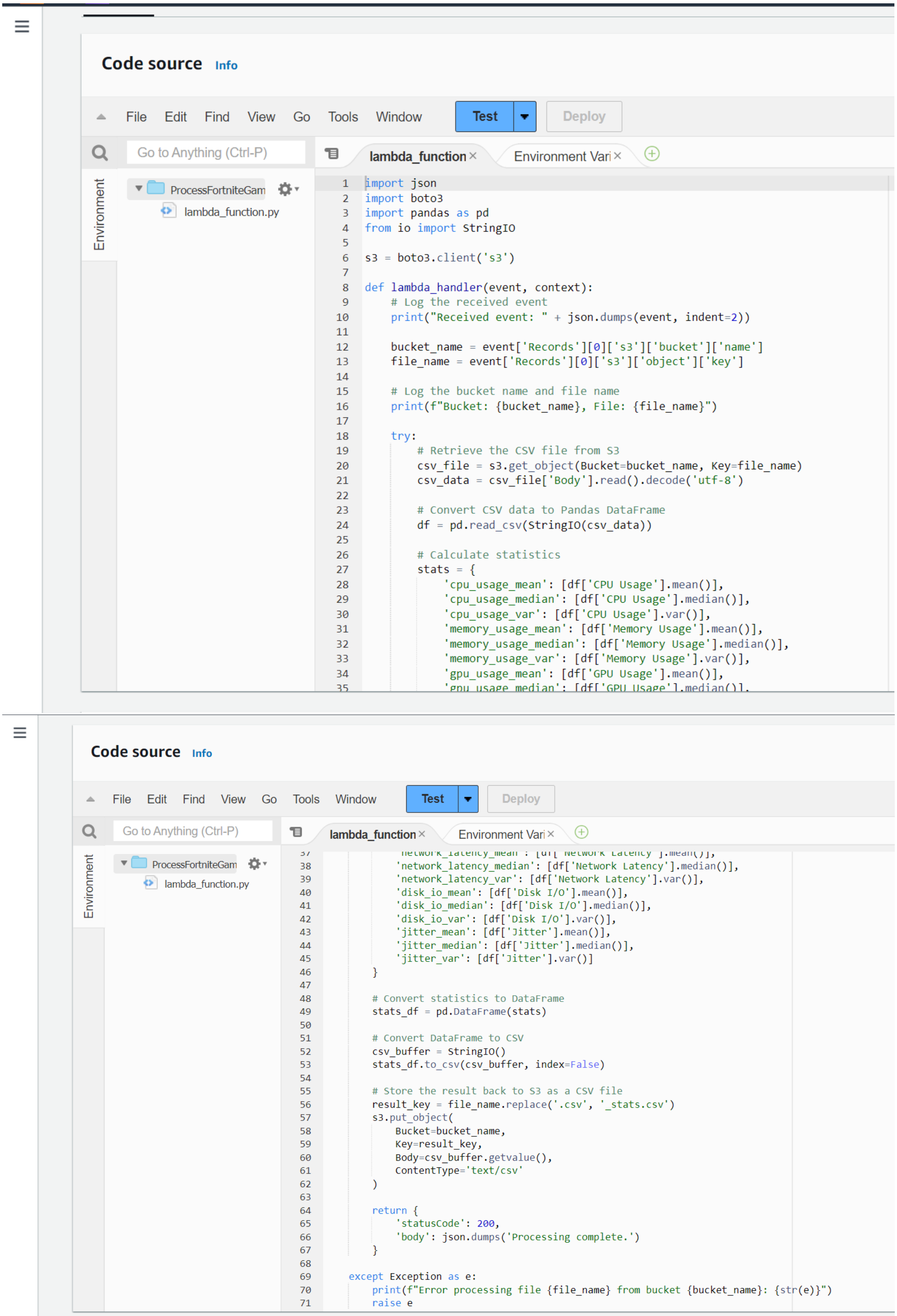

Figure 18: Lambda Function Code : to calculate mean, median and mode

4.1.2 AWS SageMaker

- Create Notebook
- Open Notebook
- Running code for performing action on game-metrics dataset individually and another code for performing action on combined dataset.

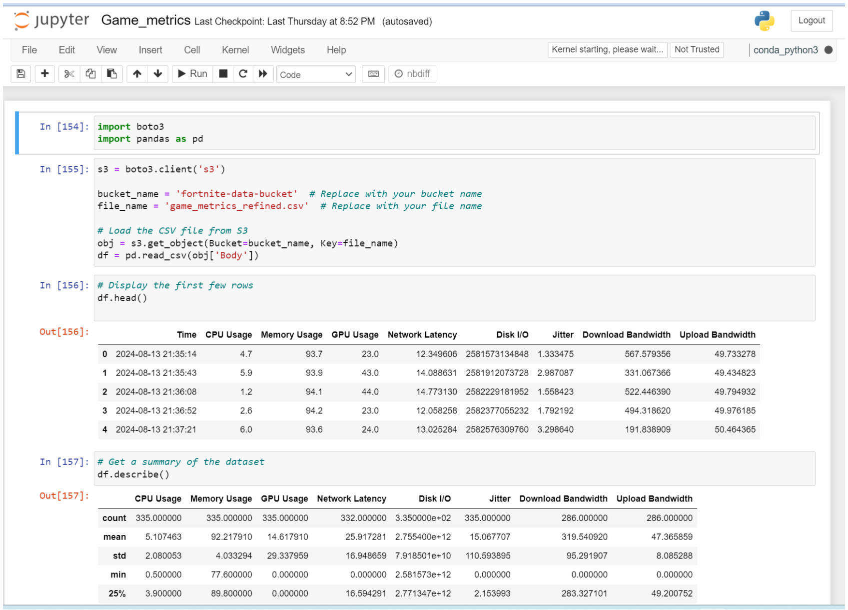

Figure 19: SageMaker Notebook 1

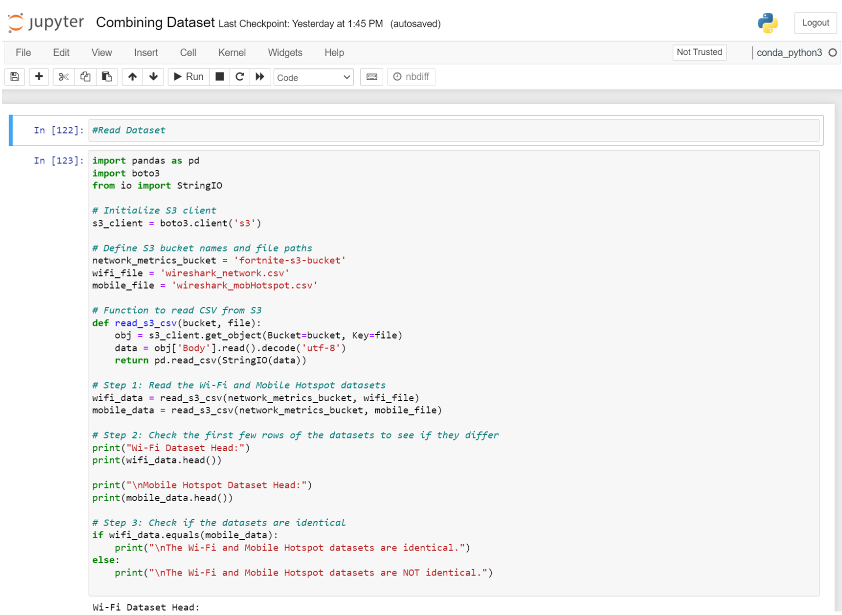

Figure 20: SageMaker Notebook 2

5 Processing Dataset and Training Model on Individual Datset

5.1 Game Metrics Dataset Individual

5.1.1 Calculating Performance Metrics

In [154]:
```python
import boto3
import pandas as pd
```

In [155]:
```python
s3 = boto3.client('s3')

bucket_name = 'fortnite-data-bucket'  # Replace with your bucket name
file_name = 'game_metrics_refined.csv'  # Replace with your file name

# Load the CSV file from S3
obj = s3.get_object(Bucket=bucket_name, Key=file_name)
df = pd.read_csv(obj['Body'])
```

In [156]:
```python
# Display the first few rows
df.head()
```

Out[156]:

	Time	CPU Usage	Memory Usage	GPU Usage	Network Latency	Disk I/O	Jitter	Download Bandwidth	Upload Bandwidth
0	2024-08-13 21:35:14	4.7	93.7	23.0	12.349606	2581573134848	1.333475	567.579356	49.733278
1	2024-08-13 21:35:43	5.9	93.9	43.0	14.088631	2581912073728	2.987087	331.067366	49.434823
2	2024-08-13 21:36:08	1.2	94.1	44.0	14.773130	2582229181952	1.558423	522.446390	49.794932
3	2024-08-13 21:36:52	2.6	94.2	23.0	12.058258	2582377055232	1.792192	494.318620	49.976185
4	2024-08-13 21:37:21	6.0	93.6	24.0	13.025284	2582576309760	3.298640	191.838909	50.464365

In [157]:
```python
# Get a summary of the dataset
df.describe()
```

Out[157]:

	CPU Usage	Memory Usage	GPU Usage	Network Latency	Disk I/O	Jitter	Download Bandwidth	Upload Bandwidth
count	335.000000	335.000000	335.000000	332.000000	3.350000e+02	335.000000	286.000000	286.000000
mean	5.107463	92.217910	14.617910	25.917281	2.755400e+12	15.067707	319.540920	47.365859
std	2.080053	4.033294	29.337959	16.948659	7.918501e+10	110.593895	95.291907	8.085288
min	0.500000	77.600000	0.000000	0.000000	2.581573e+12	0.000000	0.000000	0.000000
25%	3.900000	89.800000	0.000000	16.594291	2.771347e+12	2.153993	283.327101	49.200752

Figure 21: Load Game Metrics Dataset

In [158]:
```python
import pandas as pd

# Check for Network Latency column
print(df['Network Latency'].describe())

# Calculate average, min, and max network latency
average_latency = df['Network Latency'].mean()
min_latency = df['Network Latency'].min()
max_latency = df['Network Latency'].max()

# Print results
print(f"Average Network Latency: {average_latency:.4f} ms")
print(f"Minimum Network Latency: {min_latency:.4f} ms")
print(f"Maximum Network Latency: {max_latency:.4f} ms")

# Visualize latency distribution
import matplotlib.pyplot as plt
plt.hist(df['Network Latency'], bins=50, alpha=0.7, color='blue')
plt.title('Network Latency Distribution')
plt.xlabel('Latency (ms)')
plt.ylabel('Frequency')
plt.show()
```

```
count    332.000000
mean      25.917281
std       16.948659
min        0.000000
25%       16.594291
50%       26.742935
75%       30.311882
max      246.974230
Name: Network Latency, dtype: float64
Average Network Latency: 25.9173 ms
Minimum Network Latency: 0.0000 ms
Maximum Network Latency: 246.9742 ms
```

Figure 22: Calculating Network Latency Performance

```
In [159]: # Check CPU usage statistics
print(df['CPU Usage'].describe())

# Calculate average, min, and max CPU usage
average_cpu_usage = df['CPU Usage'].mean()
min_cpu_usage = df['CPU Usage'].min()
max_cpu_usage = df['CPU Usage'].max()

# Print results
print(f"Average CPU Usage: {average_cpu_usage:.2f}%")
print(f"Minimum CPU Usage: {min_cpu_usage:.2f}%")
print(f"Maximum CPU Usage: {max_cpu_usage:.2f}%")

# Visualize CPU Usage distribution
plt.hist(df['CPU Usage'], bins=50, alpha=0.7, color='green')
plt.title('CPU Usage Distribution')
plt.xlabel('CPU Usage (%)')
plt.ylabel('Frequency')
plt.show()
```

```
count    335.000000
mean       5.107463
std        2.080053
min        0.500000
25%        3.900000
50%        4.800000
75%        6.150000
max       14.000000
Name: CPU Usage, dtype: float64
Average CPU Usage: 5.11%
Minimum CPU Usage: 0.50%
Maximum CPU Usage: 14.00%
```

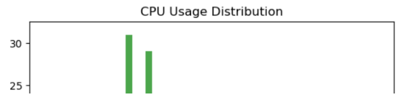

Figure 23: Calculating CPU Usage Performance

```
In [160]: import pandas as pd
import matplotlib.pyplot as plt

# Assuming df is your DataFrame with GPU Usage data
# Let's first check basic statistics of GPU Usage

gpu_usage_stats = df['GPU Usage'].describe()

print("GPU Usage Statistics:")
print(gpu_usage_stats)

# Plotting GPU Usage Distribution
plt.figure(figsize=(8, 6))
plt.hist(df['GPU Usage'], bins=20, color='orange', alpha=0.7)
plt.title("GPU Usage Distribution")
plt.xlabel("GPU Usage (%)")
plt.ylabel("Frequency")
plt.show()

# Specific statistics
average_gpu_usage = df['GPU Usage'].mean()
min_gpu_usage = df['GPU Usage'].min()
max_gpu_usage = df['GPU Usage'].max()

print(f"Average GPU Usage: {average_gpu_usage:.2f}%")
print(f"Minimum GPU Usage: {min_gpu_usage:.2f}%")
print(f"Maximum GPU Usage: {max_gpu_usage:.2f}%")
```

```
GPU Usage Statistics:
count    335.000000
mean      14.617910
std       29.337959
min        0.000000
25%        0.000000
50%        0.000000
75%        0.000000
max       98.000000
Name: GPU Usage, dtype: float64
```

GPU Usage Distribution

Figure 24: Calculating GPU Usage Performance

```
In [161]: import matplotlib.pyplot as plt

# Memory Usage Statistics
memory_stats = df['Memory Usage'].describe()
print(memory_stats)

average_memory_usage = df['Memory Usage'].mean()
min_memory_usage = df['Memory Usage'].min()
max_memory_usage = df['Memory Usage'].max()

print(f"Average Memory Usage: {average_memory_usage:.2f}%")
print(f"Minimum Memory Usage: {min_memory_usage:.2f}%")
print(f"Maximum Memory Usage: {max_memory_usage:.2f}%")

# Plot Memory Usage Distribution
plt.figure(figsize=(8, 6))
plt.hist(df['Memory Usage'], bins=30, color='orange', alpha=0.7)
plt.title('Memory Usage Distribution')
plt.xlabel('Memory Usage (%)')
plt.ylabel('Frequency')
plt.show()
```

```
count    335.000000
mean      92.217910
std        4.033294
min       77.600000
25%       89.800000
50%       93.600000
75%       95.150000
max       98.300000
Name: Memory Usage, dtype: float64
Average Memory Usage: 92.22%
Minimum Memory Usage: 77.60%
Maximum Memory Usage: 98.30%
```

Figure 25: Calculating Memory Usage Performance

```
In [164]: # Disk I/O Statistics
disk_io_stats = df['Disk I/O'].describe()
print(disk_io_stats)

average_disk_io = df['Disk I/O'].mean()
min_disk_io = df['Disk I/O'].min()
max_disk_io = df['Disk I/O'].max()

print(f"Average Disk I/O: {average_disk_io:.2f}%")
print(f"Minimum Disk I/O: {min_disk_io:.2f}%")
print(f"Maximum Disk I/O: {max_disk_io:.2f}%")

# Plot Disk I/O Distribution
plt.figure(figsize=(8, 6))
plt.hist(df['Disk I/O'], bins=30, color='purple', alpha=0.7)
plt.title('Disk I/O Distribution')
plt.xlabel('Disk I/O (MB/s)')
plt.ylabel('Frequency')
plt.show()

count    3.350000e+02
mean     2.755400e+12
std      7.918501e+10
min      2.581573e+12
25%      2.771347e+12
50%      2.797162e+12
75%      2.804576e+12
max      2.819065e+12
Name: Disk I/O, dtype: float64
Average Disk I/O: 2755400384210.91%
Minimum Disk I/O: 2581573134848.00%
Maximum Disk I/O: 2819065078784.00%
```

Figure 26: Calculating Disk I/O Performance

```
In [165]: # Download Bandwidth Statistics
download_bandwidth_stats = df['Download Bandwidth'].describe()
print("Download Bandwidth Statistics:\n", download_bandwidth_stats)

# Upload Bandwidth Statistics
upload_bandwidth_stats = df['Upload Bandwidth'].describe()
print("Upload Bandwidth Statistics:\n", upload_bandwidth_stats)

average_download_bandwidth = df['Download Bandwidth'].mean()
min_download_bandwidth = df['Download Bandwidth'].min()
max_download_bandwith = df['Download Bandwidth'].max()

print(f"Average Download Bandwidth: {average_download_bandwidth:.2f}%")
print(f"Minimum Download Bandwidth: {min_download_bandwidth:.2f}%")
print(f"Maximum Download Bandwidth: {max_download_bandwith:.2f}%")

average_upload_bandwidth = df['Upload Bandwidth'].mean()
min_upload_bandwidth = df['Upload Bandwidth'].min()
max_upload_bandwith = df['Upload Bandwidth'].max()

print(f"Average Upload Bandwidth: {average_upload_bandwidth:.2f}%")
print(f"Minimum Upload Bandwidth: {min_upload_bandwidth:.2f}%")
print(f"Maximum Upload Bandwidth: {max_upload_bandwith:.2f}%")

# Plot Download and Upload Bandwidth Distributions
plt.figure(figsize=(10, 6))

plt.subplot(1, 2, 1)
plt.hist(df['Download Bandwidth'], bins=30, color='green', alpha=0.7)
plt.title('Download Bandwidth Distribution')
plt.xlabel('Download Bandwidth (Mbps)')
plt.ylabel('Frequency')

plt.subplot(1, 2, 2)
plt.hist(df['Upload Bandwidth'], bins=30, color='blue', alpha=0.7)
plt.title('Upload Bandwidth Distribution')
plt.xlabel('Upload Bandwidth (Mbps)')
plt.ylabel('Frequency')

plt.tight_layout()
plt.show()
```

Figure 27: Calculating download and upload bandwidth Performance

```
In [166]: # Jitter Statistics
jitter_stats = df['Jitter'].describe()
print(jitter_stats)

average_jitter = df['Jitter'].mean()
min_jitter = df['Jitter'].min()
max_jitter = df['Jitter'].max()

print(f"Average Jitter: {average_jitter:.2f}%")
print(f"Minimum Jitter: {min_jitter:.2f}%")
print(f"Maximum Jitter: {max_jitter:.2f}%")

# Plot Jitter Distribution
plt.figure(figsize=(8, 6))
plt.hist(df['Jitter'], bins=30, color='red', alpha=0.7)
plt.title('Jitter Distribution')
plt.xlabel('Jitter (ms)')
plt.ylabel('Frequency')
plt.show()

count     335.000000
mean       15.067707
std       110.593895
min         0.000000
25%         2.153993
50%         3.103912
75%         4.681945
max      1594.493230
Name: Jitter, dtype: float64
Average Jitter: 15.07%
Minimum Jitter: 0.00%
Maximum Jitter: 1594.49%
```

Figure 28: Calculating Jitter Performance

5.1.2 Model Training

A. Random Forest

In [134]:

```
# Import necessary libraries
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.impute import SimpleImputer
import numpy as np
import pandas as pd

# Step 1: Impute missing values in X_train, X_test, y_train, y_test

# Impute missing values in X_train and X_test using SimpleImputer (mean strategy)
imputer = SimpleImputer(strategy='mean')
X_train_imputed = imputer.fit_transform(X_train)  # Impute X_train
X_test_imputed = imputer.transform(X_test)        # Impute X_test

# Impute missing values in y_train and y_test (if any)
y_train = y_train.fillna(y_train.mean())  # Replace NaN values in y_train with the mean
y_test = y_test.fillna(y_test.mean())     # Replace NaN values in y_test with the mean

# Step 2: Train the Random Forest model
model = RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(X_train_imputed, y_train)  # Train the model using imputed data

# Step 3: Make predictions on the imputed test set
y_pred = model.predict(X_test_imputed)

# Step 4: Calculate Evaluation Metrics
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
rmse = mean_squared_error(y_test, y_pred, squared=False)  # RMSE is the square root of MSE
r2 = r2_score(y_test, y_pred)

# Step 5: Print the Evaluation Results
print(f"Mean Squared Error (MSE): {mse}")
print(f"Mean Absolute Error (MAE): {mae}")
print(f"Root Mean Squared Error (RMSE): {rmse}")
print(f"R² Score: {r2}")

# Step 6: (Optional) Check for NaN values in predictions (y_pred) (very rare, but just in case)
print("NaN values in y_pred:", np.isnan(y_pred).sum())
```

Figure 29: Random Forest Training Code

In [135]:

```
import matplotlib.pyplot as plt

# Scatter plot of actual vs predicted QoE scores
plt.scatter(y_test, y_pred, color='blue', label='Predicted')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', linewidth=2, label='Perfect Fit')  # Perfect predic
plt.xlabel('Actual QoE Score')
plt.ylabel('Predicted QoE Score')
plt.title('Actual vs Predicted QoE Score (Random Forest)')
plt.legend()
plt.show()
```

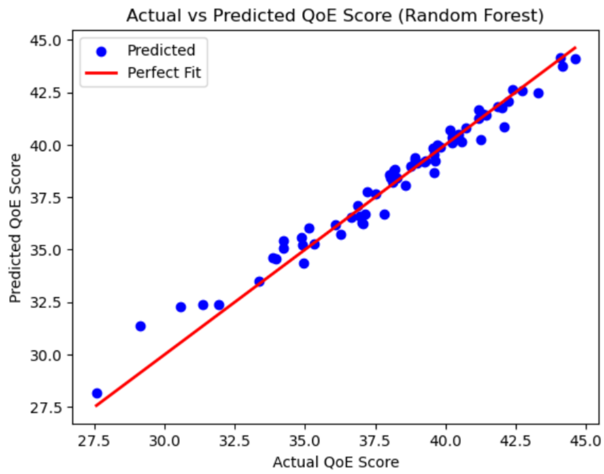

Figure 30: Plot Random Forest Model

B. Gradient Boosting Regressor : XGBoost

```
In [137]: #Gradient Boosting Regressor:

In [138]: from sklearn.impute import SimpleImputer

          # Initialize the imputer
          imputer = SimpleImputer(strategy='mean')  # You can also use 'median' if appropriate

          # Apply the imputer to the training and test sets
          X_train_imputed = imputer.fit_transform(X_train)
          X_test_imputed = imputer.transform(X_test)

          # Now train the Gradient Boosting Regressor using the imputed data
          gbr_model.fit(X_train_imputed, y_train)
          y_pred_gbr = gbr_model.predict(X_test_imputed)

In [139]: from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

          # Calculate evaluation metrics
          mse_gbr = mean_squared_error(y_test, y_pred_gbr)
          mae_gbr = mean_absolute_error(y_test, y_pred_gbr)
          rmse_gbr = mean_squared_error(y_test, y_pred_gbr, squared=False)  # RMSE is the square root of MSE
          r2_gbr = r2_score(y_test, y_pred_gbr)

          # Print the results
          print(f"Mean Squared Error (MSE): {mse_gbr}")
          print(f"Mean Absolute Error (MAE): {mae_gbr}")
          print(f"Root Mean Squared Error (RMSE): {rmse_gbr}")
          print(f"R² Score: {r2_gbr}")

          Mean Squared Error (MSE): 0.14523394316374988
          Mean Absolute Error (MAE): 0.2869169674143648
          Root Mean Squared Error (RMSE): 0.38109571391416863
          R² Score: 0.988361435684305
```

Figure 31: Code for XGBoost Model

```
In [140]: #Visual XGBoost

In [141]: import matplotlib.pyplot as plt

          # Scatter plot of actual vs predicted QoE scores
          plt.scatter(y_test, y_pred_gbr, color='blue')
          plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', linewidth=2)  # Perfect prediction line
          plt.xlabel('Actual QoE Score')
          plt.ylabel('Predicted QoE Score')
          plt.title('Actual vs Predicted QoE Score (Gradient Boosting)')
          plt.show()
```

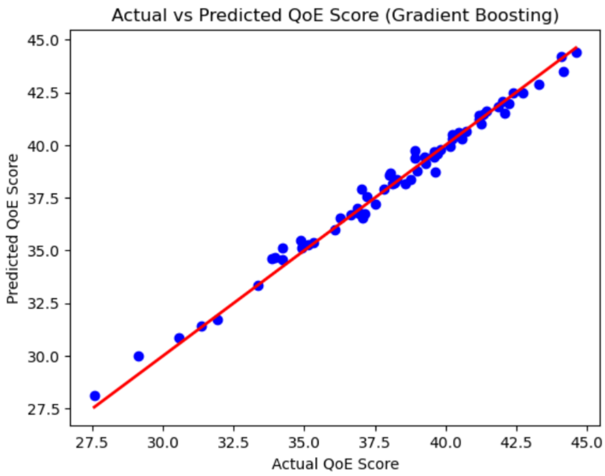

Figure 32: Plot xgboost

C. Support Vector Regression : SVR

```
In [ ]: #Training the SVR model:
```

```
In [144]: from sklearn.svm import SVR
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Since SVR performs better when the data is scaled, we'll scale the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_imputed)  # Scale the imputed training data
X_test_scaled = scaler.transform(X_test_imputed)        # Scale the imputed test data

# Initialize the SVR model
svr_model = SVR(kernel='rbf')  # You can also try other kernels like 'linear' or 'poly'

# Train the model
svr_model.fit(X_train_scaled, y_train)

# Make predictions on the test set
y_pred_svr = svr_model.predict(X_test_scaled)

# Calculate evaluation metrics for SVR
mse_svr = mean_squared_error(y_test, y_pred_svr)
mae_svr = mean_absolute_error(y_test, y_pred_svr)
rmse_svr = mean_squared_error(y_test, y_pred_svr, squared=False)
r2_svr = r2_score(y_test, y_pred_svr)

# Print the evaluation results for SVR
print(f"SVR - Mean Squared Error (MSE): {mse_svr}")
print(f"SVR - Mean Absolute Error (MAE): {mae_svr}")
print(f"SVR - Root Mean Squared Error (RMSE): {rmse_svr}")
print(f"SVR - R² Score: {r2_svr}")

SVR - Mean Squared Error (MSE): 1.7721803987677869
SVR - Mean Absolute Error (MAE): 0.7885500420066415
SVR - Root Mean Squared Error (RMSE): 1.3312326613961163
SVR - R² Score: 0.8579833673811527
```

Figure 33: Code for SVR Model

```
In [146]: #Hypertunning SVR Model
```

```
In [147]: from sklearn.model_selection import GridSearchCV

# Define the parameter grid for SVR
param_grid = {
    'C': [0.1, 1, 10],
    'epsilon': [0.01, 0.1, 1],
    'kernel': ['rbf', 'linear', 'poly']
}

# Use GridSearchCV to find the best hyperparameters
grid_search_svr = GridSearchCV(SVR(), param_grid, cv=5, scoring='r2')
grid_search_svr.fit(X_train_scaled, y_train)

# Get the best parameters
print("Best parameters for SVR:", grid_search_svr.best_params_)

# Evaluate the model with the best parameters
y_pred_svr_tuned = grid_search_svr.predict(X_test_scaled)
mse_svr_tuned = mean_squared_error(y_test, y_pred_svr_tuned)
r2_svr_tuned = r2_score(y_test, y_pred_svr_tuned)
print(f"Tuned SVR - MSE: {mse_svr_tuned}, R² Score: {r2_svr_tuned}")

Best parameters for SVR: {'C': 10, 'epsilon': 0.01, 'kernel': 'linear'}
Tuned SVR - MSE: 0.0018504607367699662, R² Score: 0.9998517102419076
```

```
In [148]: # Use the best estimator from GridSearchCV for cross-validation
cv_scores_svr_tuned = cross_val_score(grid_search_svr.best_estimator_, X_train_scaled, y_train, cv=5, scoring='r2')
print(f"Cross-validated R² scores (Tuned SVR): {cv_scores_svr_tuned}")
print(f"Mean Cross-validated R² score (Tuned SVR): {cv_scores_svr_tuned.mean()}")

Cross-validated R² scores (Tuned SVR): [0.9999974  0.99999875 0.99999957 0.99999884 0.99989786]
Mean Cross-validated R² score (Tuned SVR): 0.9999784834310752
```

Figure 34: Hypertunning SVR Model

```
In [ ]: #SVR Visualization: Actual vs Predicted QoE Scores
```

```
In [149]: import matplotlib.pyplot as plt

# Scatter plot of actual vs predicted QoE scores (for Tuned SVR)
plt.scatter(y_test, y_pred_svr_tuned, color='blue', label='Predicted')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', linewidth=2, label='Perfect Fit')  # Perfect pred
plt.xlabel('Actual QoE Score')
plt.ylabel('Predicted QoE Score')
plt.title('Actual vs Predicted QoE Score (Tuned SVR)')
plt.legend()
plt.show()
```

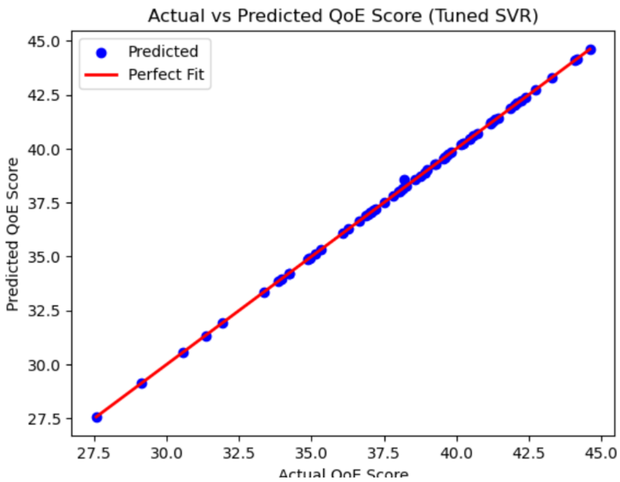

Figure 35: Plot graph for SVR Model

D. Neural Network

```
In [ ]: #Train the Neural Network (MLPRegressor)
```

```
In [150]: from sklearn.neural_network import MLPRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Initialize the Neural Network model (Multi-Layer Perceptron Regressor)
# You can experiment with different hidden layers and neuron sizes
nn_model = MLPRegressor(hidden_layer_sizes=(100, 50), max_iter=1000, random_state=42)

# Train the Neural Network model
nn_model.fit(X_train_scaled, y_train)

# Make predictions on the test set
y_pred_nn = nn_model.predict(X_test_scaled)

# Calculate evaluation metrics for Neural Network
mse_nn = mean_squared_error(y_test, y_pred_nn)
mae_nn = mean_absolute_error(y_test, y_pred_nn)
rmse_nn = mean_squared_error(y_test, y_pred_nn, squared=False)
r2_nn = r2_score(y_test, y_pred_nn)

# Print the evaluation results for Neural Network
print(f"Neural Network - Mean Squared Error (MSE): {mse_nn}")
print(f"Neural Network - Mean Absolute Error (MAE): {mae_nn}")
print(f"Neural Network - Root Mean Squared Error (RMSE): {rmse_nn}")
print(f"Neural Network - R² Score: {r2_nn}")

Neural Network - Mean Squared Error (MSE): 0.26922036638723024
Neural Network - Mean Absolute Error (MAE): 0.3128992613871213
Neural Network - Root Mean Squared Error (RMSE): 0.5188644971350711
Neural Network - R² Score: 0.9784255768242832
```

Figure 36: Code for Neural Network

In [ ]:
```
#Visualize Neural Network
```

In [151]:
```
import matplotlib.pyplot as plt

# Scatter plot of actual vs predicted QoE scores (Neural Network)
plt.scatter(y_test, y_pred_nn, color='blue', label='Predicted')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', linewidth=2, label='Perfect Fit')
plt.xlabel('Actual QoE Score')
plt.ylabel('Predicted QoE Score')
plt.title('Actual vs Predicted QoE Score (Neural Network)')
plt.legend()
plt.show()
```

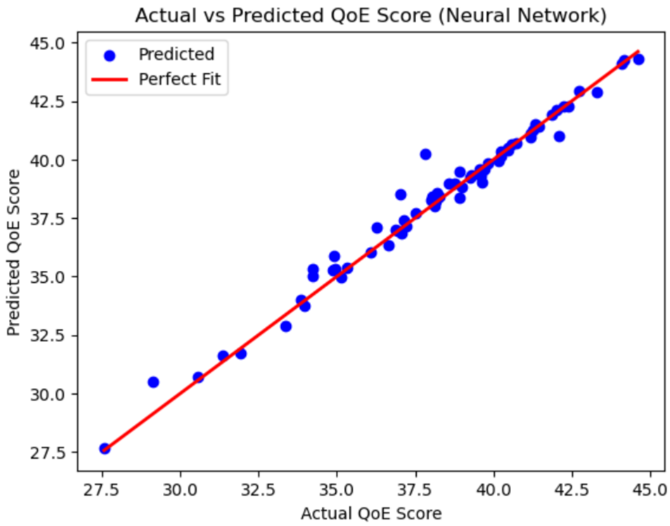

Figure 37: Visualize Neural Network

5.2 Dataset collected using Wire-shark : Calculating Performance Metrics using Google Colab

5.2.1 Calculating Performance Metrics :

A. Average RTT and Average Jitter Calcualtion on wifi(5G) network:

```
import pandas as pd

# Load the CSV file
wireshark_data = pd.read_csv('wireshark_network.csv', engine='python', on_bad_lines='skip')

# Function to calculate RTT
def calculate_rtt(df):
    rtt_values = []
    seq_to_time = {}

    for i, row in df.iterrows():
        # Ensure 'Info' field is treated as a string
        info = str(row['Info'])

        # Capture sequence numbers and their corresponding times
        if "Seq=" in info:
            try:
                seq_number = info.split("Seq=")[1].split()[0]
                seq_to_time[seq_number] = float(row['Time'])
            except (IndexError, ValueError):
                # Skip if there are issues with the parsing or conversion
                continue

        # Match ACK numbers with their corresponding sequence numbers to calculate RTT
        if "Ack=" in info:
            try:
                ack_number = info.split("Ack=")[1].split()[0]
                if ack_number in seq_to_time:
                    rtt = float(row['Time']) - seq_to_time[ack_number]
                    rtt_values.append(rtt)
            except (IndexError, ValueError):
                # Skip if there are issues with the parsing or conversion
                continue

    return rtt_values
```

```
[45]
# Function to calculate Jitter
def calculate_jitter(df):
    jitter_values = []
    previous_time = None

    for i, row in df.iterrows():
        try:
            current_time = float(row['Time'])

            if previous_time is not None:
                jitter = abs(current_time - previous_time)
                jitter_values.append(jitter)

            previous_time = current_time
        except ValueError:
            # Skip if there are issues with time conversion
            continue

    return jitter_values

# Calculate RTT and Jitter
rtt_values = calculate_rtt(wireshark_data)
jitter_values = calculate_jitter(wireshark_data)

# Calculate average RTT and Jitter with safe division
average_rtt_wifi = sum(rtt_values) / len(rtt_values) if rtt_values else 0
average_jitter_wifi = sum(jitter_values) / len(jitter_values) if jitter_values else 0

print(f"Average RTT Wifi: {average_rtt_wifi:.6f} seconds")
print(f"Average Jitter Wifi: {average_jitter_wifi:.6f} seconds")
```

```
Average RTT Wifi: 0.227748 seconds
Average Jitter Wifi: 0.000063 seconds
```

Figure 38: Code to Calculate Avg RTT and jitter on 5G network dataset

B. Average RTT and Average Jitter Calcualtion on Mobile Hotspot(4G) network

```python
import pandas as pd

# Load the CSV file
wireshark_data = pd.read_csv('wireshark_mobHotspot.csv', engine='python', on_bad_lines='skip')

# Function to calculate RTT
def calculate_rtt(df):
    rtt_values = []
    seq_to_time = {}

    for i, row in df.iterrows():
        # Ensure 'Info' field is treated as a string
        info = str(row['Info'])

        # Capture sequence numbers and their corresponding times
        if "Seq=" in info:
            try:
                seq_number = info.split("Seq=")[1].split()[0]
                seq_to_time[seq_number] = float(row['Time'])
            except (IndexError, ValueError):
                # Skip if there are issues with the parsing or conversion
                continue

        # Match ACK numbers with their corresponding sequence numbers to calculate RTT
        if "Ack=" in info:
            try:
                ack_number = info.split("Ack=")[1].split()[0]
                if ack_number in seq_to_time:
                    rtt = float(row['Time']) - seq_to_time[ack_number]
                    rtt_values.append(rtt)
            except (IndexError, ValueError):
                # Skip if there are issues with the parsing or conversion
                continue

    return rtt_values
```

```python
# Function to calculate Jitter
def calculate_jitter(df):
    jitter_values = []
    previous_time = None

    for i, row in df.iterrows():
        try:
            current_time = float(row['Time'])

            if previous_time is not None:
                jitter = abs(current_time - previous_time)
                jitter_values.append(jitter)

            previous_time = current_time
        except ValueError:
            # Skip if there are issues with time conversion
            continue

    return jitter_values

# Calculate RTT and Jitter
rtt_values = calculate_rtt(wireshark_data)
jitter_values = calculate_jitter(wireshark_data)

# Calculate average RTT and Jitter with safe division
average_rtt_mob = sum(rtt_values) / len(rtt_values) if rtt_values else 0
average_jitter_mob = sum(jitter_values) / len(jitter_values) if jitter_values else 0

print(f"Average RTT Mob: {average_rtt_mob:.6f} seconds")
print(f"Average Jitter Mob: {average_jitter_mob:.6f} seconds")
```

```
Average RTT Mob: 2.121424 seconds
Average Jitter Mob: 0.000541 seconds
```

Figure 39: ode to Calculate Avg RTT and jitter on 4G network dataset

C. Throughput Calculation for both network related dataset

```python
import pandas as pd

# Function to calculate throughput
def calculate_throughput(df, packet_size_col, timestamp_col):
    # Total bytes transferred
    total_bytes = df[packet_size_col].sum()

    # Total time in seconds (max - min timestamp)
    duration = df[timestamp_col].max() - df[timestamp_col].min()

    # Convert duration to seconds and calculate throughput in Mbps
    # Throughput = (Total bytes * 8 bits per byte) / (Duration in seconds * 1e6 to convert to Mbps)
    throughput_mbps = (total_bytes * 8) / (duration * 1e6)
    return throughput_mbps

# Assuming the datasets are already loaded as wifi_data and mobile_data
wifi_throughput = calculate_throughput(wifi_data, 'Length', 'Time')
mobile_throughput = calculate_throughput(mobile_data, 'Length', 'Time')

# Print throughput results
print(f"WiFi Throughput: {wifi_throughput:.2f} Mbps")
print(f"Mobile Hotspot Throughput: {mobile_throughput:.2f} Mbps")
```

```
WiFi Throughput: 153.24 Mbps
Mobile Hotspot Throughput: 16.80 Mbps
```

Figure 40: Code to Calculate throughput for both network dataset

D. Packet Inter-arrival Time Calculation for both network related dataset

```python
[36] def calculate_interarrival_time(df, timestamp_col):
        # Calculate inter-arrival times by finding the difference between consecutive packet times
        df['InterarrivalTime'] = df[timestamp_col].diff()

        # Summary statistics for inter-arrival time
        interarrival_stats = df['InterarrivalTime'].describe()
        return interarrival_stats

    # Example: Calculate inter-arrival times for WiFi and Mobile Hotspot
    wifi_interarrival = calculate_interarrival_time(wifi_data, 'Time')
    mobile_interarrival = calculate_interarrival_time(mobile_data, 'Time')

    # Print results
    print("WiFi Packet Inter-arrival Time Stats:\n", wifi_interarrival)
    print("Mobile Hotspot Packet Inter-arrival Time Stats:\n", mobile_interarrival)
```

Figure 41: Code to calculate Packet Inter-arrival Time for both network related dataset

E. TCP Retransmissions Calcualtion for both network related dataset

```python
[37] def count_tcp_retransmissions(df):
        # Identify retransmissions using the 'Info' column for strings like 'Retransmission'
        retransmissions = df[df['Info'].str.contains('Retransmission', case=False, na=False)]

        # Count number of retransmissions
        retransmission_count = retransmissions.shape[0]

        return retransmission_count

    # Example: Count retransmissions for WiFi and Mobile Hotspot
    wifi_retransmissions = count_tcp_retransmissions(wifi_data)
    mobile_retransmissions = count_tcp_retransmissions(mobile_data)

    # Print results
    print(f"WiFi TCP Retransmissions: {wifi_retransmissions}")
    print(f"Mobile Hotspot TCP Retransmissions: {mobile_retransmissions}")
```

```
WiFi TCP Retransmissions: 14356
Mobile Hotspot TCP Retransmissions: 289
```

Figure 42: TCP Retransmissions Calculation for both network related dataset

F. RTT Variability (Standard Deviation) Calculation for both network related dataset

```
[39] def calculate_tcp_rtt(df):
         # Filter for TCP packets (if needed)
         tcp_packets = df[df['Protocol'] == 'TCP']

         # Filter for SYN and SYN-ACK packets
         syn_packets = tcp_packets[tcp_packets['Info'].str.contains('SYN', case=False)]
         ack_packets = tcp_packets[tcp_packets['Info'].str.contains('ACK', case=False)]

         # Ensure packets are sorted by time
         df_sorted = tcp_packets.sort_values(by='Time')

         # Create a column to store calculated RTT
         df['RTT'] = None

         # Iterate through the SYN packets and find the corresponding ACK
         for idx, syn_row in syn_packets.iterrows():
             # Find the ACK packet for the same TCP stream
             ack_row = ack_packets[ack_packets['Source'] == syn_row['Destination']]
             if not ack_row.empty:
                 # Calculate RTT as the time difference between SYN and ACK
                 rtt = ack_row['Time'].values[0] - syn_row['Time']
                 df.at[idx, 'RTT'] = rtt

         return df

     # Apply to your dataset
     wifi_data_with_rtt = calculate_tcp_rtt(wifi_data)
     mobile_data_with_rtt = calculate_tcp_rtt(mobile_data)

     # Now you can calculate RTT statistics
     wifi_rtt_stddev = calculate_rtt_stddev(wifi_data_with_rtt, 'RTT')
     mobile_rtt_stddev = calculate_rtt_stddev(mobile_data_with_rtt, 'RTT')

     print(f"WiFi RTT Standard Deviation: {wifi_rtt_stddev:.4f} seconds")
     print(f"Mobile Hotspot RTT Standard Deviation: {mobile_rtt_stddev:.4f} seconds")

     WiFi RTT Standard Deviation: 25.1514 seconds
     Mobile Hotspot RTT Standard Deviation: 108.5104 seconds
```

Figure 43: RTT Variability (Standard Deviation) Calculation for both network related dataset

5.2.2 Visualization for all the metrics calculated above for Network Dataset

```python
import matplotlib.pyplot as plt
import numpy as np

# Assuming you have the dynamically computed values for Wi-Fi and Mobile Hotspot from your previous steps

# Example values for the metrics (these should be replaced by the actual dynamic values you've computed):
wifi_metrics = [average_rtt_wifi, average_jitter_wifi, wifi_throughput, wifi_interarrival['mean'], wifi_interarrival['std'], wifi_retransmissions, wifi_rtt_stddev]
mobile_metrics = [average_rtt_mob, average_jitter_mob, mobile_throughput, mobile_interarrival['mean'], mobile_interarrival['std'], mobile_retransmissions, mobile_rtt_stddev]

# Define labels for the metrics
labels = ['Average RTT (s)', 'Average Jitter (s)', 'Throughput (Mbps)', 'Inter-arrival Time (Mean)', 'Inter-arrival Time (Std)', 'TCP Retransmissions', 'RTT Std Dev']

# Set up the plot
x = np.arange(len(labels))  # the label locations
width = 0.35  # the width of the bars

fig, ax = plt.subplots(figsize=(10, 6))
rects1 = ax.bar(x - width/2, wifi_metrics, width, label='Wi-Fi')
rects2 = ax.bar(x + width/2, mobile_metrics, width, label='Mobile Hotspot')

# Add some text for labels, title and custom x-axis tick labels, etc.
ax.set_xlabel('Metrics')
ax.set_ylabel('Values')
ax.set_title('Wi-Fi vs Mobile Hotspot Metrics')
ax.set_xticks(x)
ax.set_xticklabels(labels, rotation=45, ha="right")
ax.legend()

# Attach a text label above each bar in *rects*, displaying its height.
def autolabel(rects):
    """Attach a text label above each bar in *rects*, displaying its height."""
    for rect in rects:
        height = rect.get_height()
        ax.annotate('{}'.format(round(height, 2)),
                    xy=(rect.get_x() + rect.get_width() / 2, height),
                    xytext=(0, 3),  # 3 points vertical offset
```

```python
# Attach a text label above each bar in *rects*, displaying its height.
def autolabel(rects):
    """Attach a text label above each bar in *rects*, displaying its height."""
    for rect in rects:
        height = rect.get_height()
        ax.annotate('{}'.format(round(height, 2)),
                    xy=(rect.get_x() + rect.get_width() / 2, height),
                    xytext=(0, 3),  # 3 points vertical offset
                    textcoords="offset points",
                    ha='center', va='bottom')

autolabel(rects1)
autolabel(rects2)

fig.tight_layout()
plt.show()
```

Figure 44: Code for Visualization for all the metrics calculated above for Network Dataset

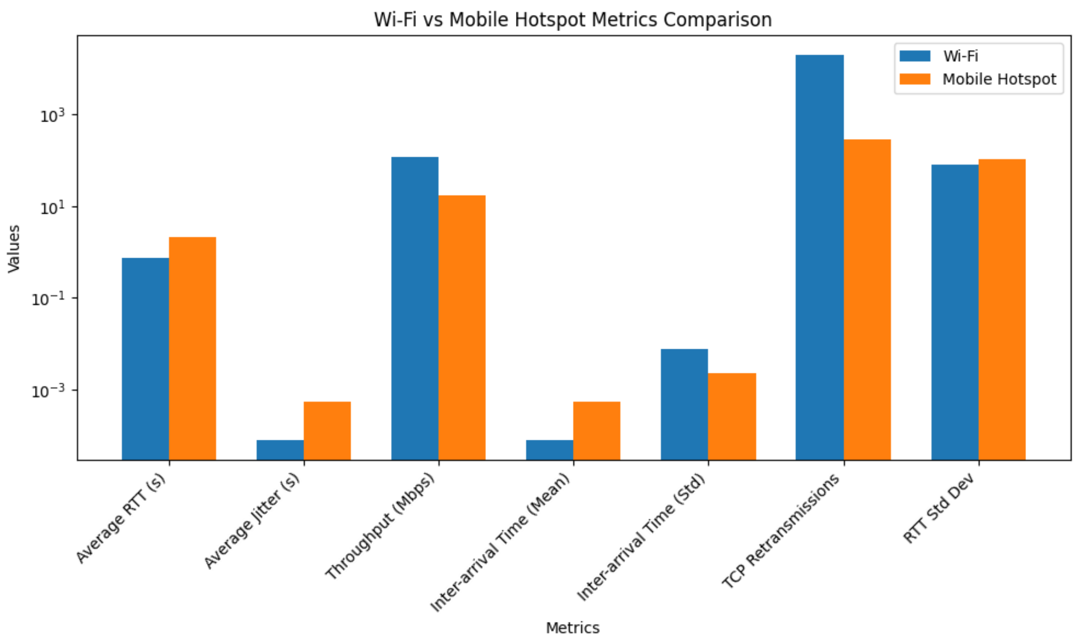

Figure 45: Graph of visualization

6 Processing Dataset and Model Training on Combined Dataset

6.1 Processing Dataset and calculating performance metrics

This research have tow combined dataset as follow:

- Wifi-combined : Game-Metrics Dataset + Wireshark-Network(5G) Dataset
- Mobile-combined : Game-Metrics Dataset + Wireshark-MobHotspot(4G) Dataset

6.1.1 Read Dataset

```
In [122]: #Read Dataset

In [123]: import pandas as pd
          import boto3
          from io import StringIO

          # Initialize S3 client
          s3_client = boto3.client('s3')

          # Define S3 bucket names and file paths
          network_metrics_bucket = 'fortnite-s3-bucket'
          wifi_file = 'wireshark_network.csv'
          mobile_file = 'wireshark_mobHotspot.csv'

          # Function to read CSV from S3
          def read_s3_csv(bucket, file):
              obj = s3_client.get_object(Bucket=bucket, Key=file)
              data = obj['Body'].read().decode('utf-8')
              return pd.read_csv(StringIO(data))

          # Step 1: Read the Wi-Fi and Mobile Hotspot datasets
          wifi_data = read_s3_csv(network_metrics_bucket, wifi_file)
          mobile_data = read_s3_csv(network_metrics_bucket, mobile_file)

          # Step 2: Check the first few rows of the datasets to see if they differ
          print("Wi-Fi Dataset Head:")
          print(wifi_data.head())

          print("\nMobile Hotspot Dataset Head:")
          print(mobile_data.head())

          # Step 3: Check if the datasets are identical
          if wifi_data.equals(mobile_data):
              print("\nThe Wi-Fi and Mobile Hotspot datasets are identical.")
          else:
              print("\nThe Wi-Fi and Mobile Hotspot datasets are NOT identical.")
```

Figure 46: Code to load both dataset

```
Wi-Fi Dataset Head:
   No.      Time                                  Source  \
0    1  0.000000                     2001:7c8:c5:c00a::2
1    2  0.000000                     2001:7c8:c5:c00a::2
2    3  0.000000                     2001:7c8:c5:c00a::2
3    4  0.000000                     2001:7c8:c5:c00a::2
4    5  0.000012  2a02:8084:20c2:5780:2def:7b7d:1c79:b8bf

                               Destination Protocol  Length  \
0  2a02:8084:20c2:5780:2def:7b7d:1c79:b8bf      TCP      74
1  2a02:8084:20c2:5780:2def:7b7d:1c79:b8bf      TCP      74
2  2a02:8084:20c2:5780:2def:7b7d:1c79:b8bf      TCP      74
3  2a02:8084:20c2:5780:2def:7b7d:1c79:b8bf      TCP      74
4                      2001:7c8:c5:c00a::2      TCP    1066

                                         Info
0    8080  >  20107 [ACK] Seq=1 Ack=1 Win=6506 Len=0
1    8080  >  20103 [ACK] Seq=1 Ack=1 Win=9910 Len=0
2  8080  >  20103 [ACK] Seq=1 Ack=1441 Win=9933 L...
3  8080  >  20103 [ACK] Seq=1 Ack=2881 Win=9955 L...
4  20107  >  8080 [PSH, ACK] Seq=58785 Ack=1 Win=...
```

Figure 47: Wifi(5G) Dataset Head(first few rows from dataset)

```
Mobile Hotspot Dataset Head:
   No.      Time          Source     Destination Protocol  Length  \
0    1  0.000000  74.125.193.100  192.168.86.242      UDP      70
1    2  0.000000  74.125.193.100  192.168.86.242      UDP      69
2    3  0.000000  74.125.193.100  192.168.86.242      UDP      70
3    4  0.000164  74.125.193.100  192.168.86.242      UDP      70
4    5  0.000270  192.168.86.242  74.125.193.100      UDP    1292

                     Info
0    443  >  58048 Len=28
1    443  >  58048 Len=27
2    443  >  58048 Len=28
3    443  >  58048 Len=28
4  58048  >  443 Len=1250
```

Figure 48: Mobile(4G) Dataset Head(first few rows from dataset)

```
In [124]: print('game_metrics')
          print(game_metrics.head())
          print(game_metrics['Time'].describe())
          print('wifiData time descirbe')
          print(wifi_data['Time'].describe())
          print('mobData time descirbe')
          print(mobile_data['Time'].describe())
```

```
game_metrics
                  Time  CPU Usage  Memory Usage  GPU Usage  Network Latency  \
0 2024-08-13 21:35:14        4.7          93.7       23.0        12.349606
1 2024-08-13 21:35:43        5.9          93.9       43.0        14.088631
2 2024-08-13 21:36:08        1.2          94.1       44.0        14.773130
3 2024-08-13 21:36:52        2.6          94.2       23.0        12.058258
4 2024-08-13 21:37:21        6.0          93.6       24.0        13.025284

        Disk I/O    Jitter  Download Bandwidth  Upload Bandwidth
0  2581573134848  1.333475          567.579356         49.733278
1  2581912073728  2.987087          331.067366         49.434823
2  2582229181952  1.558423          522.446390         49.794932
3  2582377055232  1.792192          494.318620         49.976185
4  2582576309760  3.298640          191.838909         50.464365
count                              335
mean     2024-08-14 15:21:17.385074688
min                2024-08-13 21:35:14
25%                2024-08-14 19:52:51
50%                2024-08-14 20:29:38
75%                2024-08-14 21:02:03
max                2024-08-14 21:33:03
Name: Time, dtype: object
wifiData time descirbe
count    8.147494e+06
mean     1.185516e+03
std      1.410971e+03
min      0.000000e+00
25%      1.300070e+02
50%      4.024866e+02
75%      2.050613e+03
max      5.023966e+03
Name: Time, dtype: float64
```

Figure 49: code to load game metrics dataset and its output

6.1.2 Combine wifi (5G) and Mobile(4G) Dataset with Game metrics

```
In [128]: #Merging Dataset

In [129]: # import boto3
          import pandas as pd
          from io import StringIO

          # Initialize S3 client
          s3_client = boto3.client('s3')

          # Define S3 bucket names and file paths
          game_metrics_bucket = 'fortnite-data-bucket'
          game_metrics_file = 'game_metrics_refined.csv'

          network_metrics_bucket = 'fortnite-s3-bucket'
          wifi_file = 'wireshark_network.csv'
          mobile_file = 'wireshark_mobHotspot.csv'

          # Function to read CSV from S3
          def read_s3_csv(bucket, file):
              obj = s3_client.get_object(Bucket=bucket, Key=file)
              data = obj['Body'].read().decode('utf-8')
              return pd.read_csv(StringIO(data))

          # Step 1: Read the datasets
          game_metrics = read_s3_csv(game_metrics_bucket, game_metrics_file)
          wifi_data = read_s3_csv(network_metrics_bucket, wifi_file)
          mobile_data = read_s3_csv(network_metrics_bucket, mobile_file)

          # Step 2: Convert 'Time' in game_metrics to datetime
          game_metrics['Time'] = pd.to_datetime(game_metrics['Time'], errors='coerce')

          # Step 3: Convert 'Time' in Wireshark data to relative datetime (adjust based on your capture start time)
          capture_start_time = pd.to_datetime('2024-08-12 03:15:00')  # Adjust this as needed
          wifi_data['Time'] = capture_start_time + pd.to_timedelta(wifi_data['Time'], unit='s')
          mobile_data['Time'] = capture_start_time + pd.to_timedelta(mobile_data['Time'], unit='s')

          # Step 4: Keep the 'Time' column and filter only numeric columns for resampling
          wifi_numeric = wifi_data.select_dtypes(include=['number'])
          mobile_numeric = mobile_data.select_dtypes(include=['number'])

          # Add back the 'Time' column for resampling
```

```
# Step 4: Keep the 'Time' column and filter only numeric columns for resampling
wifi_numeric = wifi_data.select_dtypes(include=['number'])
mobile_numeric = mobile_data.select_dtypes(include=['number'])

# Add back the 'Time' column for resampling
wifi_numeric['Time'] = wifi_data['Time']
mobile_numeric['Time'] = mobile_data['Time']

# Step 5: Resample the Wireshark data based on a specific time granularity (e.g., 1 minute)
wifi_resampled = wifi_numeric.set_index('Time').resample('1T').mean().reset_index()
mobile_resampled = mobile_numeric.set_index('Time').resample('1T').mean().reset_index()

# Step 6: Merge game metrics with Wi-Fi data
combined_wifi_data = pd.merge_asof(game_metrics.sort_values('Time'),
                                   wifi_resampled.sort_values('Time'),
                                   on='Time',
                                   direction='nearest',
                                   tolerance=pd.Timedelta('1T'))

# Step 7: Merge game metrics with Mobile Hotspot data
combined_mobile_data = pd.merge_asof(game_metrics.sort_values('Time'),
                                     mobile_resampled.sort_values('Time'),
                                     on='Time',
                                     direction='nearest',
                                     tolerance=pd.Timedelta('1T'))

# Step 8: Save the combined datasets as new CSV files in S3
csv_buffer_wifi = StringIO()
csv_buffer_mobile = StringIO()

combined_wifi_data.to_csv(csv_buffer_wifi, index=False)
combined_mobile_data.to_csv(csv_buffer_mobile, index=False)

# Save to S3
s3_client.put_object(Bucket=network_metrics_bucket, Key='combined_wifi_game_metrics.csv', Body=csv_buffer_wifi.getvalue())
s3_client.put_object(Bucket=network_metrics_bucket, Key='combined_mobile_game_metrics.csv', Body=csv_buffer_mobile.getvalue())

# Print confirmation
print("Combined Wi-Fi and Game Metrics saved successfully as 'combined_wifi_game_metrics.csv'")
print("Combined Mobile Hotspot and Game Metrics saved successfully as 'combined_mobile_game_metrics.csv'")
```

Figure 50: Code to Merge dataset

```python
In [130]: #Load the combined datasets and check the first few rows
import boto3
import pandas as pd
from io import StringIO

# Initialize S3 client
s3_client = boto3.client('s3')

# Define S3 bucket name and file paths
network_metrics_bucket = 'fortnite-s3-bucket'
wifi_combined_file = 'combined_wifi_game_metrics.csv'
mobile_combined_file = 'combined_mobile_game_metrics.csv'

# Function to read CSV from S3
def read_s3_csv(bucket, file):
    obj = s3_client.get_object(Bucket=bucket, Key=file)
    data = obj['Body'].read().decode('utf-8')
    return pd.read_csv(StringIO(data))

# Step 1: Read the combined datasets from S3
wifi_combined = read_s3_csv(network_metrics_bucket, wifi_combined_file)
mobile_combined = read_s3_csv(network_metrics_bucket, mobile_combined_file)

# Step 2: Check the first few rows of the datasets
print("Wi-Fi Combined Dataset Head:")
print(wifi_combined.head())

print("\nMobile Hotspot Combined Dataset Head:")
print(mobile_combined.head())
```

Figure 51: Code to display Merged Dataset

6.1.3 Check Missing Values

```python
In [132]: # Check for missing values in Wi-Fi dataset
print("\nMissing Values in Wi-Fi Combined Dataset:")
print(wifi_combined.isnull().sum())

# Check for missing values in Mobile Hotspot dataset
print("\nMissing Values in Mobile Hotspot Combined Dataset:")
print(mobile_combined.isnull().sum())
```

Figure 52: Check Missing Values

6.1.4 Let's examine how different metrics correlate with each other in both datasets. This will help us identify potential relationships between variables.

```python
In [136]: import seaborn as sns
import matplotlib.pyplot as plt

# Drop non-numeric columns (e.g., 'Time')
wifi_combined_numeric = wifi_combined.select_dtypes(include=['float64', 'int64'])
mobile_combined_numeric = mobile_combined.select_dtypes(include=['float64', 'int64'])

# Correlation matrix for Wi-Fi combined dataset
plt.figure(figsize=(10, 8))
sns.heatmap(wifi_combined_numeric.corr(), annot=True, cmap='coolwarm')
plt.title('Correlation Matrix - Wi-Fi Combined Dataset')
plt.show()

# Correlation matrix for Mobile Hotspot combined dataset
plt.figure(figsize=(10, 8))
sns.heatmap(mobile_combined_numeric.corr(), annot=True, cmap='coolwarm')
plt.title('Correlation Matrix - Mobile Hotspot Combined Dataset')
plt.show()
```

Figure 53: Check Correlation between different Metrics

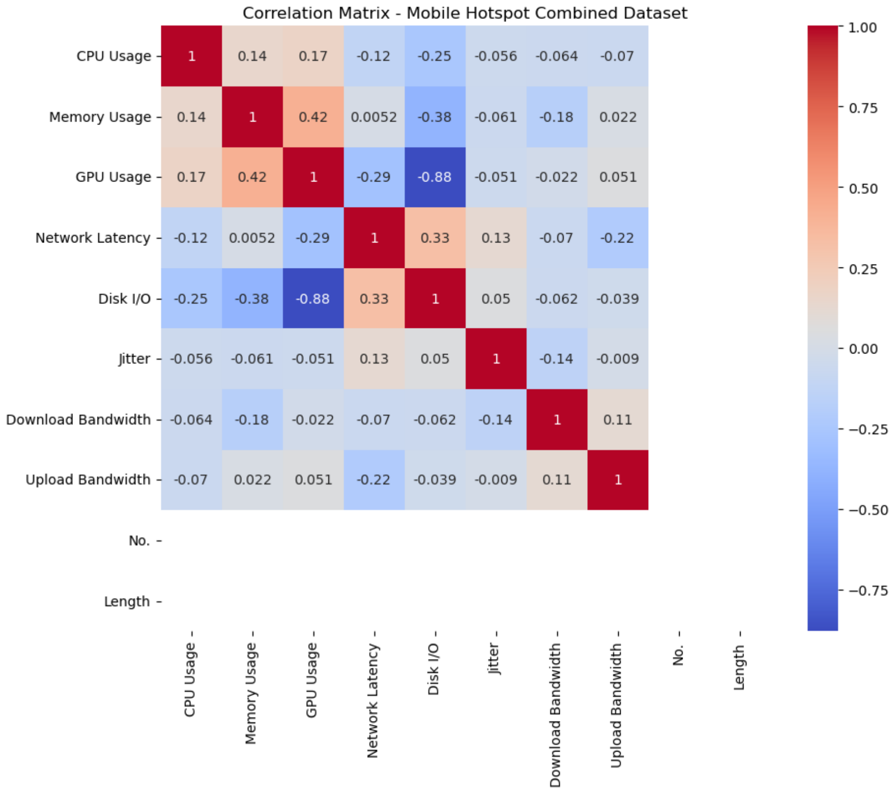

Figure 55: Correlation Matrix for 4G combined dataset

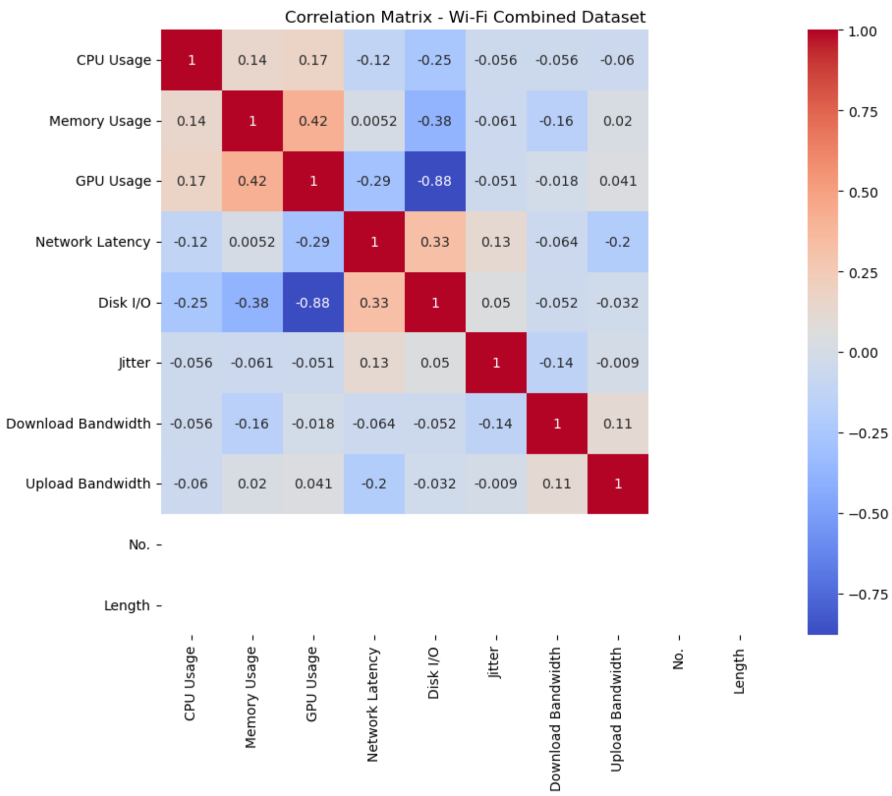

Figure 54: Correlation Matrix for wifi combined Dataset

6.1.5 Plot all Performance Metric graph for Wifi(5G) vs Mobile(4G) Network

```
In [137]: import matplotlib.pyplot as plt
          import seaborn as sns

          # Define the figure size and subplots (5 rows, 2 columns)
          fig, axes = plt.subplots(5, 2, figsize=(14, 18))

          # Plot 1: CPU Usage for Wi-Fi and Mobile Hotspot
          sns.histplot(wifi_combined['CPU Usage'], kde=True, bins=30, ax=axes[0, 0])
          axes[0, 0].set_title('CPU Usage - Wi-Fi')
          sns.histplot(mobile_combined['CPU Usage'], kde=True, bins=30, ax=axes[0, 1])
          axes[0, 1].set_title('CPU Usage - Mobile Hotspot')

          # Plot 2: Memory Usage for Wi-Fi and Mobile Hotspot
          sns.histplot(wifi_combined['Memory Usage'], kde=True, bins=30, ax=axes[1, 0])
          axes[1, 0].set_title('Memory Usage - Wi-Fi')
          sns.histplot(mobile_combined['Memory Usage'], kde=True, bins=30, ax=axes[1, 1])
          axes[1, 1].set_title('Memory Usage - Mobile Hotspot')

          # Plot 3: GPU Usage for Wi-Fi and Mobile Hotspot
          sns.histplot(wifi_combined['GPU Usage'], kde=True, bins=30, ax=axes[2, 0])
          axes[2, 0].set_title('GPU Usage - Wi-Fi')
          sns.histplot(mobile_combined['GPU Usage'], kde=True, bins=30, ax=axes[2, 1])
          axes[2, 1].set_title('GPU Usage - Mobile Hotspot')

          # Plot 4: Network Latency for Wi-Fi and Mobile Hotspot
          sns.histplot(wifi_combined['Network Latency'], kde=True, bins=30, ax=axes[3, 0])
          axes[3, 0].set_title('Network Latency - Wi-Fi')
          sns.histplot(mobile_combined['Network Latency'], kde=True, bins=30, ax=axes[3, 1])
          axes[3, 1].set_title('Network Latency - Mobile Hotspot')

          # Plot 5: Jitter for Wi-Fi and Mobile Hotspot
          sns.histplot(wifi_combined['Jitter'], kde=True, bins=30, ax=axes[4, 0])
          axes[4, 0].set_title('Jitter - Wi-Fi')
          sns.histplot(mobile_combined['Jitter'], kde=True, bins=30, ax=axes[4, 1])
          axes[4, 1].set_title('Jitter - Mobile Hotspot')

          # Adjust layout to avoid overlapping
          plt.tight_layout()

          # Show the plot
          plt.show()
```

Figure 56: Wifi(5G) vs Mobile(4G) Network Performance Metrics

6.1.6 Feature Scaling and Create new features.

```
In [140]: #Feature Scaling
          #We will use StandardScaler from sklearn to standardize the features.
```

```
In [141]: from sklearn.preprocessing import StandardScaler

          # Select numeric columns for scaling
          numeric_columns = ['CPU Usage', 'Memory Usage', 'GPU Usage', 'Network Latency',
                             'Disk I/O', 'Jitter', 'Download Bandwidth', 'Upload Bandwidth']

          # Initialize the scaler
          scaler = StandardScaler()

          # Apply scaling on Wi-Fi combined dataset
          wifi_combined_scaled = wifi_combined.copy()
          wifi_combined_scaled[numeric_columns] = scaler.fit_transform(wifi_combined[numeric_columns])

          # Apply scaling on Mobile Hotspot combined dataset
          mobile_combined_scaled = mobile_combined.copy()
          mobile_combined_scaled[numeric_columns] = scaler.fit_transform(mobile_combined[numeric_columns])
```

```
In [142]: # Create new features for both datasets

          # Network efficiency: Download Bandwidth / Upload Bandwidth
          wifi_combined_scaled['Network Efficiency'] = wifi_combined['Download Bandwidth'] / wifi_combined['Upload Bandwidth']
          mobile_combined_scaled['Network Efficiency'] = mobile_combined['Download Bandwidth'] / mobile_combined['Upload Bandwidth']

          # Latency-to-Jitter ratio: Network Latency / Jitter
          wifi_combined_scaled['Latency to Jitter'] = wifi_combined['Network Latency'] / wifi_combined['Jitter']
          mobile_combined_scaled['Latency to Jitter'] = mobile_combined['Network Latency'] / mobile_combined['Jitter']

          # Resource-to-Performance ratio: CPU Usage / Download Bandwidth
          wifi_combined_scaled['Resource to Performance'] = wifi_combined['CPU Usage'] / wifi_combined['Download Bandwidth']
          mobile_combined_scaled['Resource to Performance'] = mobile_combined['CPU Usage'] / mobile_combined['Download Bandwidth']

          # Preview the new features
          print("Wi-Fi Combined Dataset with new features:\n", wifi_combined_scaled.head())
          print("\nMobile Hotspot Combined Dataset with new features:\n", mobile_combined_scaled.head())
```

Figure 57: Feature Scaling and creating new features

```
In [144]: #correlations between these newly created features
```

```
In [145]: import seaborn as sns
          import matplotlib.pyplot as plt

          # Plot the distribution of the new features for Wi-Fi dataset
          plt.figure(figsize=(14, 8))

          plt.subplot(2, 2, 1)
          sns.histplot(wifi_combined_scaled['Network Efficiency'], kde=True, bins=30)
          plt.title('Network Efficiency - Wi-Fi')

          plt.subplot(2, 2, 2)
          sns.histplot(wifi_combined_scaled['Latency to Jitter'], kde=True, bins=30)
          plt.title('Latency to Jitter - Wi-Fi')

          plt.subplot(2, 2, 3)
          sns.histplot(wifi_combined_scaled['Resource to Performance'], kde=True, bins=30)
          plt.title('Resource to Performance - Wi-Fi')

          plt.tight_layout()
          plt.show()

          # Now, do the same for the Mobile Hotspot dataset
          plt.figure(figsize=(14, 8))

          plt.subplot(2, 2, 1)
          sns.histplot(mobile_combined_scaled['Network Efficiency'], kde=True, bins=30)
          plt.title('Network Efficiency - Mobile Hotspot')

          plt.subplot(2, 2, 2)
          sns.histplot(mobile_combined_scaled['Latency to Jitter'], kde=True, bins=30)
          plt.title('Latency to Jitter - Mobile Hotspot')

          plt.subplot(2, 2, 3)
          sns.histplot(mobile_combined_scaled['Resource to Performance'], kde=True, bins=30)
          plt.title('Resource to Performance - Mobile Hotspot')

          plt.tight_layout()
          plt.show()
```

Figure 58: Correlation between newly created features

6.2 Model Training

6.2.1 Random Forest

```
In [148]: #1. Random Forest
```

```
In [149]: import pandas as pd
import numpy as np
import boto3
from io import StringIO
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.preprocessing import StandardScaler

# Initialize S3 client
s3_client = boto3.client('s3')

# Define S3 bucket and file paths
bucket_name = 'fortnite-s3-bucket'
wifi_file = 'combined_wifi_game_metrics.csv'
mobile_file = 'combined_mobile_game_metrics.csv'

# Function to read CSV file from S3
def read_s3_csv(bucket, file):
    obj = s3_client.get_object(Bucket=bucket, Key=file)
    data = obj['Body'].read().decode('utf-8')
    return pd.read_csv(StringIO(data))

# Step 1: Load both combined datasets from S3 (Wi-Fi and Mobile Hotspot)
wifi_combined = read_s3_csv(bucket_name, wifi_file)
mobile_combined = read_s3_csv(bucket_name, mobile_file)

# Step 2: Handle NaN values in numeric columns only
numeric_cols_wifi = wifi_combined.select_dtypes(include=[np.number]).columns
numeric_cols_mobile = mobile_combined.select_dtypes(include=[np.number]).columns

wifi_combined[numeric_cols_wifi] = wifi_combined[numeric_cols_wifi].fillna(wifi_combined[numeric_cols_wifi].mean())
mobile_combined[numeric_cols_mobile] = mobile_combined[numeric_cols_mobile].fillna(mobile_combined[numeric_cols_mobile].mean())

# Step 3: Prepare features (X) and target (y)
target_column = 'Network Latency'
features = [col for col in numeric_cols_wifi if col != target_column]
X_wifi, y_wifi = wifi_combined[features], wifi_combined[target_column]
X_mobile, y_mobile = mobile_combined[features], mobile_combined[target_column]

X_train_wifi, X_test_wifi, y_train_wifi, y_test_wifi = train_test_split(X_wifi, y_wifi, test_size=0.2, random_state=42)
X_train_mobile, X_test_mobile, y_train_mobile, y_test_mobile = train_test_split(X_mobile, y_mobile, test_size=0.2, random_state=4

# Step 4: Scale the features
scaler = StandardScaler()
X_train_wifi = scaler.fit_transform(X_train_wifi)
X_test_wifi = scaler.transform(X_test_wifi)
X_train_mobile = scaler.fit_transform(X_train_mobile)
X_test_mobile = scaler.transform(X_test_mobile)

# Step 5: Train Random Forest Model
random_forest = RandomForestRegressor(n_estimators=100, random_state=42)

# Train on Wi-Fi data
random_forest.fit(X_train_wifi, y_train_wifi)
y_pred_wifi = random_forest.predict(X_test_wifi)

# Calculate metrics for Wi-Fi
mse_wifi = mean_squared_error(y_test_wifi, y_pred_wifi)
mae_wifi = mean_absolute_error(y_test_wifi, y_pred_wifi)
rmse_wifi = np.sqrt(mse_wifi)
r2_wifi = r2_score(y_test_wifi, y_pred_wifi)

print(f"Wi-Fi Random Forest Results: MSE: {mse_wifi:.4f}, MAE: {mae_wifi:.4f}, RMSE: {rmse_wifi:.4f}, R2: {r2_wifi:.4f}")

# Train on Mobile Hotspot data
random_forest.fit(X_train_mobile, y_train_mobile)
y_pred_mobile = random_forest.predict(X_test_mobile)

# Calculate metrics for Mobile Hotspot
mse_mobile = mean_squared_error(y_test_mobile, y_pred_mobile)
mae_mobile = mean_absolute_error(y_test_mobile, y_pred_mobile)
rmse_mobile = np.sqrt(mse_mobile)
r2_mobile = r2_score(y_test_mobile, y_pred_mobile)

print(f"Mobile Hotspot Random Forest Results: MSE: {mse_mobile:.4f}, MAE: {mae_mobile:.4f}, RMSE: {rmse_mobile:.4f}, R2: {r2_mobi
```

Figure 59: Training Random Forest Model

6.2.2 XGBoost Model Training

```
In [169]: #XGBoost Model
```

```
In [170]: import xgboost as xgb
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import matplotlib.pyplot as plt

# Step 1: Initialize the XGBoost model
xgb_model = xgb.XGBRegressor(objective='reg:squarederror', random_state=42)

# Step 2: Define the parameter grid for GridSearchCV
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [3, 6, 9],
    'learning_rate': [0.01, 0.1, 0.2],
    'subsample': [0.7, 0.8, 1.0],
    'colsample_bytree': [0.7, 0.8, 1.0],
    'gamma': [0, 0.1, 0.2]
}

# Step 3: Perform GridSearchCV
grid_search_xgb = GridSearchCV(estimator=xgb_model, param_grid=param_grid,
                               scoring='neg_mean_squared_error', cv=3, verbose=1)

# Fit the model on the Wi-Fi dataset
grid_search_xgb.fit(X_train_wifi_scaled, y_train_wifi)

# Step 4: Best parameters and model
best_params_xgb = grid_search_xgb.best_params_
best_xgb_wifi = grid_search_xgb.best_estimator_

# Step 5: Predict on the test data
y_pred_wifi_xgb = best_xgb_wifi.predict(X_test_wifi_scaled)

# Step 6: Evaluate the model
mse_wifi_xgb = mean_squared_error(y_test_wifi, y_pred_wifi_xgb)
mae_wifi_xgb = mean_absolute_error(y_test_wifi, y_pred_wifi_xgb)
rmse_wifi_xgb = mse_wifi_xgb ** 0.5
r2_wifi_xgb = r2_score(y_test_wifi, y_pred_wifi_xgb)
```

```
In [171]: # Print the results
print(f"Best Hyperparameters for Wi-Fi dataset (XGBoost): {best_params_xgb}")
print(f"Wi-Fi XGBoost Results: MSE: {mse_wifi_xgb:.4f}, MAE: {mae_wifi_xgb:.4f}, RMSE: {rmse_wifi_xgb:.4f}, R2: {r2_wifi_xgb:.4f}

# Step 7: Plot feature importances
importances_xgb = best_xgb_wifi.feature_importances_
feature_importances_xgb = pd.Series(importances_xgb, index=X_train_wifi.columns)
feature_importances_xgb.sort_values(ascending=False).plot(kind='bar', figsize=(10, 6))
plt.title("XGBoost Feature Importance - Wi-Fi Dataset")
plt.show()
```

```
Best Hyperparameters for Wi-Fi dataset (XGBoost): {'colsample_bytree': 1.0, 'gamma': 0, 'learning_rate': 0.1, 'max_depth': 9,
'n_estimators': 200, 'subsample': 0.7}
Wi-Fi XGBoost Results: MSE: 85.2439, MAE: 4.1361, RMSE: 9.2328, R2: -0.3435
```

Figure 60: Training XGBoost Model

A. Further tuning of the XGBoost model by adjusting the parameter grid and focusing on more fine-grained control over the parameters

```
In [172]: import xgboost as xgb
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import matplotlib.pyplot as plt

# Step 1: Initialize the XGBoost model
xgb_model = xgb.XGBRegressor(objective='reg:squarederror', random_state=42)

# Step 2: Define the refined parameter grid for GridSearchCV
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [6, 9, 12],
    'learning_rate': [0.01, 0.05, 0.1],
    'subsample': [0.7, 0.9, 1.0],
    'colsample_bytree': [0.7, 1.0],
    'gamma': [0, 0.1, 0.2],
    'min_child_weight': [1, 5, 10],
    'reg_alpha': [0, 0.01, 0.1]  # L1 regularization
}

# Step 3: Perform GridSearchCV
grid_search_xgb = GridSearchCV(estimator=xgb_model, param_grid=param_grid,
                               scoring='neg_mean_squared_error', cv=3, verbose=1, n_jobs=-1)

# Fit the model on the Wi-Fi dataset
grid_search_xgb.fit(X_train_wifi_scaled, y_train_wifi)

# Step 4: Best parameters and model
best_params_xgb = grid_search_xgb.best_params_
best_xgb_wifi = grid_search_xgb.best_estimator_

# Step 5: Predict on the test data
y_pred_wifi_xgb = best_xgb_wifi.predict(X_test_wifi_scaled)

# Step 6: Evaluate the model
mse_wifi_xgb = mean_squared_error(y_test_wifi, y_pred_wifi_xgb)
mae_wifi_xgb = mean_absolute_error(y_test_wifi, y_pred_wifi_xgb)
rmse_wifi_xgb = mse_wifi_xgb ** 0.5
r2_wifi_xgb = r2_score(y_test_wifi, y_pred_wifi_xgb)

# Print the results
print(f"Best Hyperparameters for Wi-Fi dataset (XGBoost): {best_params_xgb}")
print(f"Wi-Fi XGBoost Results after further tuning: MSE: {mse_wifi_xgb:.4f}, MAE: {mae_wifi_xgb:.4f}, RMSE: {rmse_wifi_xgb:.4f},

# Step 7: Plot feature importances
importances_xgb = best_xgb_wifi.feature_importances_
feature_importances_xgb = pd.Series(importances_xgb, index=X_train_wifi.columns)
feature_importances_xgb.sort_values(ascending=False).plot(kind='bar', figsize=(10, 6))
plt.title("XGBoost Feature Importance - Wi-Fi Dataset (After Further Tuning)")
plt.show()
```

Figure 61: Tuned XGBoost : Applying GridSearchCV

6.2.3 Support Vector Regression (SVR) using paratmer grid

```
In [175]: from sklearn.svm import SVR
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline

# Step 1: Define a pipeline with StandardScaler and SVR
pipe_svr = Pipeline([
    ('scaler', StandardScaler()),  # Scaling is essential for SVR
    ('svr', SVR())
])

# Step 2: Define parameter grid for tuning SVR
param_grid_svr = {
    'svr__kernel': ['linear', 'rbf', 'poly'],  # Different kernel types
    'svr__C': [0.1, 1, 10, 100],  # Regularization parameter
    'svr__epsilon': [0.01, 0.1, 0.2],  # Epsilon for margin of error
    'svr__gamma': ['scale', 'auto']  # Kernel coefficient for 'rbf' and 'poly'
}

# Step 3: Perform GridSearchCV for SVR
grid_search_svr = GridSearchCV(estimator=pipe_svr, param_grid=param_grid_svr,
                               scoring='neg_mean_squared_error', cv=3, verbose=2, n_jobs=-1)

# Step 4: Fit the model on the Wi-Fi dataset
grid_search_svr.fit(X_train_wifi_scaled, y_train_wifi)

# Step 5: Get the best model and predict on the test data
best_svr = grid_search_svr.best_estimator_
y_pred_wifi_svr = best_svr.predict(X_test_wifi_scaled)

# Step 6: Evaluate the model
mse_wifi_svr = mean_squared_error(y_test_wifi, y_pred_wifi_svr)
mae_wifi_svr = mean_absolute_error(y_test_wifi, y_pred_wifi_svr)
rmse_wifi_svr = mse_wifi_svr ** 0.5
r2_wifi_svr = r2_score(y_test_wifi, y_pred_wifi_svr)

# Print the results
print(f"Best Hyperparameters for Wi-Fi dataset (SVR): {grid_search_svr.best_params_}")
print(f"Wi-Fi SVR Results: MSE: {mse_wifi_svr:.4f}, MAE: {mae_wifi_svr:.4f}, RMSE: {rmse_wifi_svr:.4f}, R2: {r2_wifi_svr:.4f}")
```

Figure 62: SVR Model Training

6.2.4 Neural Network

```
In [ ]: #Neural Network
```

```
In [66]: # Import required libraries
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.regularizers import l2
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import numpy as np
import matplotlib.pyplot as plt

# Step 1: Create the neural network model with Dropout and L2 regularization
def create_nn_model(input_dim):
    model = Sequential()
    # Input layer + first hidden layer
    model.add(Dense(64, input_dim=input_dim, activation='relu', kernel_regularizer=l2(0.01)))
    model.add(Dropout(0.2))  # Dropout layer to prevent overfitting
    # Second hidden layer
    model.add(Dense(32, activation='relu', kernel_regularizer=l2(0.01)))
    model.add(Dropout(0.2))  # Dropout layer to prevent overfitting
    # Output layer for regression
    model.add(Dense(1))

    # Step 2: Compile the model with a reduced learning rate
    model.compile(optimizer=Adam(learning_rate=0.0001), loss='mean_squared_error', metrics=['mae'])

    return model

# Step 3: Scale the data (Neural networks perform better with scaled input)
scaler = StandardScaler()
X_train_wifi_scaled = scaler.fit_transform(X_train_wifi)
X_test_wifi_scaled = scaler.transform(X_test_wifi)

# Step 4: Create and train the model
input_dim = X_train_wifi_scaled.shape[1]
model_nn = create_nn_model(input_dim)

# Step 5: Implement early stopping to prevent overfitting
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)
```


```
# Step 5: Implement early stopping to prevent overfitting
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)
```


```
# Train the model with early stopping
history = model_nn.fit(X_train_wifi_scaled, y_train_wifi,
                       validation_split=0.2,
                       epochs=100,
                       batch_size=32,
                       verbose=1,
                       callbacks=[early_stopping])

# Step 6: Predict on the test data
y_pred_wifi_nn = model_nn.predict(X_test_wifi_scaled)

# Step 7: Evaluate the model
mse_wifi_nn = mean_squared_error(y_test_wifi, y_pred_wifi_nn)
mae_wifi_nn = mean_absolute_error(y_test_wifi, y_pred_wifi_nn)
rmse_wifi_nn = np.sqrt(mse_wifi_nn)
r2_wifi_nn = r2_score(y_test_wifi, y_pred_wifi_nn)

# Print results
print(f"Wi-Fi Neural Network Results: MSE: {mse_wifi_nn:.4f}, MAE: {mae_wifi_nn:.4f}, RMSE: {rmse_wifi_nn:.4f}, R2: {r2_wifi_nn:.

# Step 8: Plotting the training loss and validation loss
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()

# Step 9: Plotting the training MAE and validation MAE
plt.plot(history.history['mae'], label='Train MAE')
plt.plot(history.history['val_mae'], label='Validation MAE')
plt.title('Mean Absolute Error')
plt.xlabel('Epoch')
plt.ylabel('MAE')
plt.legend()
plt.show()
```

Figure 63: Neural Network Model Training

A. Adding more layer to tune Neural Network for improving model's accuracy

```
In [183]: # Import required libraries
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.regularizers import l2
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import numpy as np
import matplotlib.pyplot as plt

# Step 1: Create a neural network model with more layers, Dropout, and L2 regularization
def create_deep_nn_model(input_dim):
    model = Sequential()
    # Input layer + first hidden layer
    model.add(Dense(128, input_dim=input_dim, activation='relu', kernel_regularizer=l2(0.01)))
    model.add(Dropout(0.3))  # Dropout layer to prevent overfitting
    # Second hidden layer
    model.add(Dense(64, activation='relu', kernel_regularizer=l2(0.01)))
    model.add(Dropout(0.3))  # Dropout layer to prevent overfitting
    # Third hidden layer
    model.add(Dense(32, activation='relu', kernel_regularizer=l2(0.01)))
    model.add(Dropout(0.2))  # Dropout layer to prevent overfitting
    # Fourth hidden layer
    model.add(Dense(16, activation='relu', kernel_regularizer=l2(0.01)))
    model.add(Dropout(0.2))  # Dropout layer to prevent overfitting
    # Output layer for regression
    model.add(Dense(1))

    # Step 2: Compile the model with a reduced learning rate
    model.compile(optimizer=Adam(learning_rate=0.0001), loss='mean_squared_error', metrics=['mae'])

    return model

# Step 3: Scale the data (Neural networks perform better with scaled input)
scaler = StandardScaler()
X_train_wifi_scaled = scaler.fit_transform(X_train_wifi)
X_test_wifi_scaled = scaler.transform(X_test_wifi)
# Step 4: Create and train the model
input_dim = X_train_wifi_scaled.shape[1]
model_nn = create_deep_nn_model(input_dim)

# Step 5: Implement early stopping to prevent overfitting
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)

# Train the model with early stopping
history = model_nn.fit(X_train_wifi_scaled, y_train_wifi,
                       validation_split=0.2,
                       epochs=150,
                       batch_size=32,
                       verbose=1,
                       callbacks=[early_stopping])

# Step 6: Predict on the test data
y_pred_wifi_nn = model_nn.predict(X_test_wifi_scaled)

# Step 7: Evaluate the model
mse_wifi_nn = mean_squared_error(y_test_wifi, y_pred_wifi_nn)
mae_wifi_nn = mean_absolute_error(y_test_wifi, y_pred_wifi_nn)
rmse_wifi_nn = np.sqrt(mse_wifi_nn)
r2_wifi_nn = r2_score(y_test_wifi, y_pred_wifi_nn)

# Print results
print(f"Wi-Fi Neural Network Results (with more layers): MSE: {mse_wifi_nn:.4f}, MAE: {mae_wifi_nn:.4f}, RMSE: {rmse_wifi_nn:.4f}

# Step 8: Plotting the training loss and validation loss
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()

# Step 9: Plotting the training MAE and validation MAE
plt.plot(history.history['mae'], label='Train MAE')
plt.plot(history.history['val_mae'], label='Validation MAE')
plt.title('Mean Absolute Error')
plt.xlabel('Epoch')
plt.ylabel('MAE')
```

Figure 64: Adding more layer to tune Neural Network for improving model's accuracy

6.2.5 Comparing all the trained Model

```
In [191]: import matplotlib.pyplot as plt
          import seaborn as sns
          import pandas as pd

          # Data for model comparison
          model_data = {
              'Model': ['Random Forest', 'SVR (Refined)', 'XGBoost', 'Neural Network', 'Complex Neural Network'],
              'MSE': [18.3747, 30.9906, 86.8246, 29.2626, 26.2754],
              'MAE': [2.9667, 4.0076, 4.0761, 3.8804, 3.4884],
              'RMSE': [4.2866, 5.5669, 9.3180, 5.4095, 5.1260],
              'R² Score': [0.7104, 0.5116, -0.3685, 0.5388, 0.5859]
          }

          # Create a DataFrame
          df = pd.DataFrame(model_data)

          # Set the figure size
          plt.figure(figsize=(12, 8))

          # Plot MSE, MAE, RMSE, and R² in subplots
          fig, axs = plt.subplots(2, 2, figsize=(14, 10))

          # Plot MSE
          sns.barplot(x='Model', y='MSE', data=df, ax=axs[0, 0])
          axs[0, 0].set_title('Mean Squared Error (MSE)')
          axs[0, 0].tick_params(axis='x', rotation=45)

          # Plot MAE
          sns.barplot(x='Model', y='MAE', data=df, ax=axs[0, 1])
          axs[0, 1].set_title('Mean Absolute Error (MAE)')
          axs[0, 1].tick_params(axis='x', rotation=45)

          # Plot RMSE
          sns.barplot(x='Model', y='RMSE', data=df, ax=axs[1, 0])
          axs[1, 0].set_title('Root Mean Squared Error (RMSE)')
          axs[1, 0].tick_params(axis='x', rotation=45)

          # Plot R² Score
          sns.barplot(x='Model', y='R² Score', data=df, ax=axs[1, 1])
          axs[1, 1].set_title('R² Score')
          axs[1, 1].tick_params(axis='x', rotation=45)

          # Adjust layout
          plt.tight_layout()

          # Show the plots
          plt.show()
```

Figure 65: Model comparison

6.3 Changing Target variable to Jitter

6.3.1 Training all models with new target variable jitter

```
In [198]: from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from xgboost import XGBRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.model_selection import train_test_split, cross_val_score
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam
from sklearn.preprocessing import StandardScaler

# Step 1: Set 'Jitter' as the target variable (from previous step)
X_wifi = wifi_combined.drop(['Jitter', 'Time'], axis=1)  # Dropping Jitter (target) and Time (for now)
y_wifi = wifi_combined['Jitter']

# Step 2: Train-test split
X_train_wifi, X_test_wifi, y_train_wifi, y_test_wifi = train_test_split(X_wifi, y_wifi, test_size=0.2, random_state=42)

# Step 3: Standardize the data (For SVR and Neural Network)
scaler = StandardScaler()
X_train_wifi_scaled = scaler.fit_transform(X_train_wifi)
X_test_wifi_scaled = scaler.transform(X_test_wifi)

# Step 4: Define evaluation function to store results
model_performance = pd.DataFrame(columns=['Model', 'MSE', 'MAE', 'RMSE', 'R² Score', 'Cross-Validated R²'])

def store_results(model_name, y_true, y_pred, model, X_train, y_train):
    mse = mean_squared_error(y_true, y_pred)
    mae = mean_absolute_error(y_true, y_pred)
    rmse = mse ** 0.5
    r2 = r2_score(y_true, y_pred)
    # Perform cross-validation to calculate the cross-validated R² score
    cross_val_r2 = cross_val_score(model, X_train, y_train, cv=5, scoring='r2').mean() if model else 'Not applicable'

    # Store the results in the DataFrame
    model_performance.loc[len(model_performance)] = [model_name, mse, mae, rmse, r2, cross_val_r2]

# Step 5: Train models

# Random Forest
def train_random_forest(X_train, y_train, X_test, y_test):
    rf = RandomForestRegressor(n_estimators=100, random_state=42)
    rf.fit(X_train, y_train)
    y_pred = rf.predict(X_test)
    store_results('Random Forest', y_test, y_pred, rf, X_train, y_train)

# SVR
def train_svr(X_train, y_train, X_test, y_test):
    svr = SVR(kernel='rbf', C=100, epsilon=0.2, gamma='auto')
    svr.fit(X_train, y_train)
    y_pred = svr.predict(X_test)
    store_results('SVR', y_test, y_pred, svr, X_train, y_train)

# XGBoost
def train_xgboost(X_train, y_train, X_test, y_test):
    xgb_model = XGBRegressor(objective='reg:squarederror', random_state=42)
    xgb_model.fit(X_train, y_train)
    y_pred = xgb_model.predict(X_test)
    store_results('XGBoost', y_test, y_pred, xgb_model, X_train, y_train)

# Neural Network
def create_nn_model(input_dim):
    model = Sequential()
    model.add(Dense(64, input_dim=input_dim, activation='relu'))
    model.add(Dropout(0.3))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(1))  # Output layer
    model.compile(optimizer=Adam(learning_rate=0.001), loss='mean_squared_error', metrics=['mae'])
    return model

def train_neural_network(X_train, y_train, X_test, y_test):
    input_dim = X_train.shape[1]
    model_nn = create_nn_model(input_dim)
    model_nn.fit(X_train, y_train, epochs=100, batch_size=32, verbose=1, validation_split=0.2)
    y_pred = model_nn.predict(X_test).flatten()
    # Since cross_val_score doesn't work directly with Keras models, we skip cross-validation for Neural Networks
    store_results('Neural Network', y_test, y_pred, None, X_train, y_train)

# Step 6: Train all models with 'Jitter' as the target
train_random_forest(X_train_wifi, y_train_wifi, X_test_wifi, y_test_wifi)
train_svr(X_train_wifi_scaled, y_train_wifi, X_test_wifi_scaled, y_test_wifi)
train_xgboost(X_train_wifi, y_train_wifi, X_test_wifi, y_test_wifi)
train_neural_network(X_train_wifi_scaled, y_train_wifi, X_test_wifi_scaled, y_test_wifi)

# Step 7: Display the stored results
print(model_performance)
```

Figure 66: All Model Trained on Jitter as Target Variable

6.4 Changing Target variable back to Network latency

6.4.1 Training all Model again on Network Latency as target variable

Now after all the HyperTunning, GridSearchCv, changing target variable, and add more layer , we will once again train all the model to improve the accuracy.

A. Working on Wifi(5G) combined Dataset

```
In [199]: import pandas as pd
import matplotlib.pyplot as plt

# Step 1: Drop 'Jitter' from the dataset
wifi_combined_no_jitter = wifi_combined.drop(['Jitter'], axis=1)

# Step 2: Select a new target variable (for example, 'Network Latency')
new_target_variable = 'Network Latency'

# Step 3: Prepare the new features (X) and the new target (y)
X_wifi_new = wifi_combined_no_jitter.drop([new_target_variable, 'Time'], axis=1)  # Drop the new target and 'Time'
y_wifi_new = wifi_combined_no_jitter[new_target_variable]

# Step 4: Train-test split
from sklearn.model_selection import train_test_split
X_train_wifi_new, X_test_wifi_new, y_train_wifi_new, y_test_wifi_new = train_test_split(X_wifi_new, y_wifi_new, test_size=0.2, ra

# Step 5: Standardize the data (For SVR and Neural Network)
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train_wifi_scaled_new = scaler.fit_transform(X_train_wifi_new)
X_test_wifi_scaled_new = scaler.transform(X_test_wifi_new)

# Step 6: Create a DataFrame to store results
model_performance = pd.DataFrame(columns=['Model', 'MSE', 'MAE', 'RMSE', 'R²'])

# Step 7: Redefine the model training functions and store results

# Random Forest
from sklearn.ensemble import RandomForestRegressor
def train_random_forest(X_train, y_train, X_test, y_test):
    rf = RandomForestRegressor(n_estimators=100, random_state=42)
    rf.fit(X_train, y_train)
    y_pred = rf.predict(X_test)
    store_results('Random Forest', y_test, y_pred)

# SVR
from sklearn.svm import SVR
def train_svr(X_train, y_train, X_test, y_test):
    svr = SVR(kernel='rbf', C=100, epsilon=0.2, gamma='auto')
    svr.fit(X_train, y_train)
    y_pred = svr.predict(X_test)
    store_results('SVR', y_test, y_pred)

# XGBoost
from xgboost import XGBRegressor
def train_xgboost(X_train, y_train, X_test, y_test):
    xgb_model = XGBRegressor(objective='reg:squarederror', random_state=42)
    xgb_model.fit(X_train, y_train)
    y_pred = xgb_model.predict(X_test)
    store_results('XGBoost', y_test, y_pred)

# Neural Network
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam

def create_nn_model(input_dim):
    model = Sequential()
    model.add(Dense(64, input_dim=input_dim, activation='relu'))
    model.add(Dropout(0.3))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(1))  # Output layer
    model.compile(optimizer=Adam(learning_rate=0.001), loss='mean_squared_error', metrics=['mae'])
    return model

def train_neural_network(X_train, y_train, X_test, y_test):
    input_dim = X_train.shape[1]
    model_nn = create_nn_model(input_dim)
    model_nn.fit(X_train, y_train, epochs=100, batch_size=32, verbose=1, validation_split=0.2)
    y_pred = model_nn.predict(X_test).flatten()
    store_results('Neural Network', y_test, y_pred)

# Step 8: Evaluation and store function
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

def store_results(model_name, y_true, y_pred):
    mse = mean_squared_error(y_true, y_pred)
    mae = mean_absolute_error(y_true, y_pred)
    rmse = mse ** 0.5
    r2 = r2_score(y_true, y_pred)
    model_performance.loc[len(model_performance)] = [model_name, mse, mae, rmse, r2]

# Step 9: Train the models with the new target ('Network Latency')
train_random_forest(X_train_wifi_new, y_train_wifi_new, X_test_wifi_new, y_test_wifi_new)
train_svr(X_train_wifi_scaled_new, y_train_wifi_new, X_test_wifi_scaled_new, y_test_wifi_new)
train_xgboost(X_train_wifi_new, y_train_wifi_new, X_test_wifi_new, y_test_wifi_new)
train_neural_network(X_train_wifi_scaled_new, y_train_wifi_new, X_test_wifi_scaled_new, y_test_wifi_new)

# Step 10: Display the stored results
print(model_performance)

# Step 11: Visualization of the model comparison
model_performance.set_index('Model', inplace=True)
model_performance.plot(kind='bar', figsize=(10, 6))
plt.title("Model Performance Comparison")
plt.ylabel("Error / R²")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

Figure 67: Training all Model on Network Latency as target variable on Wifi(5G) Combined Dataset

Visualization of all model against performance metrics (MSE, MAE, RSME,R-squared)

```
In [200]: # Assuming the model_performance DataFrame is already populated

          import matplotlib.pyplot as plt
          import seaborn as sns

          # Step 1: Reset the index for easier plotting
          model_performance.reset_index(inplace=True)

          # Step 2: Visualization setup
          plt.figure(figsize=(14, 8))

          # Step 3: Plot MSE comparison
          plt.subplot(2, 2, 1)
          sns.barplot(x='Model', y='MSE', data=model_performance)
          plt.title('Mean Squared Error (MSE)')
          plt.xticks(rotation=45)

          # Step 4: Plot MAE comparison
          plt.subplot(2, 2, 2)
          sns.barplot(x='Model', y='MAE', data=model_performance)
          plt.title('Mean Absolute Error (MAE)')
          plt.xticks(rotation=45)

          # Step 5: Plot RMSE comparison
          plt.subplot(2, 2, 3)
          sns.barplot(x='Model', y='RMSE', data=model_performance)
          plt.title('Root Mean Squared Error (RMSE)')
          plt.xticks(rotation=45)

          # Step 6: Plot R² comparison
          plt.subplot(2, 2, 4)
          sns.barplot(x='Model', y='R²', data=model_performance)
          plt.title('R² Score')
          plt.xticks(rotation=45)

          # Step 7: Layout adjustments and show plot
          plt.tight_layout()
          plt.show()
```

Figure 68: Visualization of all model against performance metrics : 5G Dataset

B. Working on Mobile(4G) combined Dataset

```
In [201]: import pandas as pd
import matplotlib.pyplot as plt

# Step 1: Drop 'Jitter' from the mobile_combined dataset
mobile_combined_no_jitter = mobile_combined.drop(['Jitter'], axis=1)

# Step 2: Select a new target variable (for example, 'Network Latency')
new_target_variable = 'Network Latency'

# Step 3: Prepare the new features (X) and the new target (y)
X_mobile_new = mobile_combined_no_jitter.drop([new_target_variable, 'Time'], axis=1)  # Drop the new target and 'Time'
y_mobile_new = mobile_combined_no_jitter[new_target_variable]

# Step 4: Train-test split
from sklearn.model_selection import train_test_split
X_train_mobile_new, X_test_mobile_new, y_train_mobile_new, y_test_mobile_new = train_test_split(X_mobile_new, y_mobile_new, test_

# Step 5: Standardize the data (For SVR and Neural Network)
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train_mobile_scaled_new = scaler.fit_transform(X_train_mobile_new)
X_test_mobile_scaled_new = scaler.transform(X_test_mobile_new)

# Step 6: Create a DataFrame to store results
model_performance_mobile = pd.DataFrame(columns=['Model', 'MSE', 'MAE', 'RMSE', 'R²'])

# Step 7: Redefine the model training functions and store results for mobile_combined

# Random Forest
from sklearn.ensemble import RandomForestRegressor
def train_random_forest_mobile(X_train, y_train, X_test, y_test):
    rf = RandomForestRegressor(n_estimators=100, random_state=42)
    rf.fit(X_train, y_train)
    y_pred = rf.predict(X_test)
    store_results_mobile('Random Forest', y_test, y_pred)

# SVR
from sklearn.svm import SVR
def train_svr_mobile(X_train, y_train, X_test, y_test):
    svr = SVR(kernel='rbf', C=100, epsilon=0.2, gamma='auto')
    svr.fit(X_train, y_train)
    y_pred = svr.predict(X_test)
    store_results_mobile('SVR', y_test, y_pred)

# XGBoost
from xgboost import XGBRegressor
def train_xgboost_mobile(X_train, y_train, X_test, y_test):
    xgb_model = XGBRegressor(objective='reg:squarederror', random_state=42)
    xgb_model.fit(X_train, y_train)
    y_pred = xgb_model.predict(X_test)
    store_results_mobile('XGBoost', y_test, y_pred)

# Neural Network
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam

def create_nn_model(input_dim):
    model = Sequential()
    model.add(Dense(64, input_dim=input_dim, activation='relu'))
    model.add(Dropout(0.3))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(1))  # Output layer
    model.compile(optimizer=Adam(learning_rate=0.001), loss='mean_squared_error', metrics=['mae'])
    return model

def train_neural_network_mobile(X_train, y_train, X_test, y_test):
    input_dim = X_train.shape[1]
    model_nn = create_nn_model(input_dim)
    model_nn.fit(X_train, y_train, epochs=100, batch_size=32, verbose=1, validation_split=0.2)
    y_pred = model_nn.predict(X_test).flatten()
    store_results_mobile('Neural Network', y_test, y_pred)

# Step 8: Evaluation and store function for mobile_combined
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

def store_results_mobile(model_name, y_true, y_pred):
    mse = mean_squared_error(y_true, y_pred)
    mae = mean_absolute_error(y_true, y_pred)
    rmse = mse ** 0.5
    r2 = r2_score(y_true, y_pred)
    model_performance_mobile.loc[len(model_performance_mobile)] = [model_name, mse, mae, rmse, r2]

# Step 9: Train the models with the new target ('Network Latency') for mobile_combined
train_random_forest_mobile(X_train_mobile_new, y_train_mobile_new, X_test_mobile_new, y_test_mobile_new)
train_svr_mobile(X_train_mobile_scaled_new, y_train_mobile_new, X_test_mobile_scaled_new, y_test_mobile_new)
train_xgboost_mobile(X_train_mobile_new, y_train_mobile_new, X_test_mobile_new, y_test_mobile_new)
train_neural_network_mobile(X_train_mobile_scaled_new, y_train_mobile_new, X_test_mobile_scaled_new, y_test_mobile_new)

# Step 10: Display the stored results for mobile_combined
print(model_performance_mobile)

# Step 11: Visualization of the model comparison for mobile_combined
model_performance_mobile.set_index('Model', inplace=True)
model_performance_mobile.plot(kind='bar', figsize=(10, 6))
plt.title("Mobile Combined Model Performance Comparison")
plt.ylabel("Error / R²")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

Figure 69: Training all Model on Network Latency as target variable on Mobile(4G) Combined Dataset

```python
In [205]: import pandas as pd
import matplotlib.pyplot as plt

# Assuming `model_performance_wifi` and `model_performance_mobile` are DataFrames for Wi-Fi and Mobile

# Step 1: Check if 'Model' column exists and add it if necessary
if 'Model' not in model_performance.columns:
    model_performance['Model'] = ['Random Forest', 'SVR', 'XGBoost', 'Neural Network']

if 'Model' not in model_performance_mobile.columns:
    model_performance_mobile['Model'] = ['Random Forest', 'SVR', 'XGBoost', 'Neural Network']

# Step 2: Set 'Model' as index for both DataFrames
model_performance.set_index('Model', inplace=True)
model_performance_mobile.set_index('Model', inplace=True)

# Step 3: Plot a comparison of Wi-Fi Combined vs Mobile Combined metrics
fig, axes = plt.subplots(2, 2, figsize=(14, 10))

# MSE comparison
model_performance['MSE'].plot(kind='bar', ax=axes[0, 0], color='b', alpha=0.6, position=0, width=0.4, label='Wi-Fi')
model_performance_mobile['MSE'].plot(kind='bar', ax=axes[0, 0], color='r', alpha=0.6, position=1, width=0.4, label='Mobile')
axes[0, 0].set_title('MSE Comparison')
axes[0, 0].set_ylabel('MSE')
axes[0, 0].legend()

# MAE comparison
model_performance['MAE'].plot(kind='bar', ax=axes[0, 1], color='b', alpha=0.6, position=0, width=0.4, label='Wi-Fi')
model_performance_mobile['MAE'].plot(kind='bar', ax=axes[0, 1], color='r', alpha=0.6, position=1, width=0.4, label='Mobile')
axes[0, 1].set_title('MAE Comparison')
axes[0, 1].set_ylabel('MAE')
axes[0, 1].legend()

# RMSE comparison
model_performance['RMSE'].plot(kind='bar', ax=axes[1, 0], color='b', alpha=0.6, position=0, width=0.4, label='Wi-Fi')
model_performance_mobile['RMSE'].plot(kind='bar', ax=axes[1, 0], color='r', alpha=0.6, position=1, width=0.4, label='Mobile')
axes[1, 0].set_title('RMSE Comparison')
axes[1, 0].set_ylabel('RMSE')
axes[1, 0].legend()

# R² comparison
model_performance['R²'].plot(kind='bar', ax=axes[1, 1], color='b', alpha=0.6, position=0, width=0.4, label='Wi-Fi')
model_performance_mobile['R²'].plot(kind='bar', ax=axes[1, 1], color='r', alpha=0.6, position=1, width=0.4, label='Mobile')
axes[1, 1].set_title('R² Score Comparison')
axes[1, 1].set_ylabel('R² Score')
axes[1, 1].legend()

# Adjust layout and show the plot
plt.tight_layout()
plt.show()
```

Figure 70: Visualization of all model against performance metrics : 4G Dataset

These are all the steps carried during this research for the implementation. After implementing and training Model results were evaluated and compared.

References