



# KLE Technological University

Creating Value,  
Leveraging Knowledge

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING  
KLE TECHNOLOGICAL UNIVERSITY, HUBBALLI, INDIA

Minor Project Report

On

Intelligent Task Offloading in Private 5g Networks

submitted in partial fulfillment of the requirements for the award of the degree

of

Bachelor of Engineering

IN

COMPUTER SCIENCE AND ENGINEERING

*Submitted By*

Veeresh S Hiremath	01FE22BCS296
Rahul Pujari	01FE22BCS286
Manoj Pandekamat	01FE22BCS279
Anoop U Kadakol	01FE22BCS285

Under the guidance of

Dr. Narayan D G

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

2024-25



## SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

### CERTIFICATE

This is to certify that the project entitled “Intelligent Task Offloading in Private 5g Network” is a Bonafide work carried out by the student team Veeresh Hiremath - 01FE22BCS296, Rahul Pujari- 01FE22BCS286, Manoj Pandekamat - 01FE22BCS279, Anoop Kadakol- 01FE22BCS285 in partial fulfillment of the completion of the 6th semester B.E. course during the year 2024 – 2025. The project report has been approved as it satisfies the academic requirement concerning the project work prescribed for the above-mentioned course.

Dr. Narayan D G

Head, SoCSE  
Dr. Vijayalakshmi M.

#### External Viva-Voce

Name of the examiners

Signature with date

1 \_\_\_\_\_

\_\_\_\_\_

2 \_\_\_\_\_

\_\_\_\_\_

# Acknowledgement

We would like to thank our faculty and management for their professional guidance towards the completion of this work. We take this opportunity to thank Dr. Ashok Shettar, Pro-Chancellor, Dr. B.S Anami, Registrar, and Dr. P.G Tewari, Vice-Chancellor, Dr Meena S.M., Dean Academics, KLE Technological University, Hubballi, for their vision and support.

We also take this opportunity to thank Dr. Vijayalakshmi M, Professor and Head, SoCSE for having provided us direction and facilitated for enhancement of skills and academic growth.

We thank our guide Dr. Narayan D G, Professor and HOD SoCSE (AI), SoCSE for the constant guidance during interaction and reviews.

We extend our acknowledgment to the reviewers for critical suggestions and input. We also thank Project Co-ordinator Mr. Uday N.Kulkarni and Mr. Guruprasad Konnuramath for their support during the course of completion.

We express gratitude to our beloved parents for their constant encouragement and support.

Veeresh S Hiremath - 01fe22bcs296

Manoj Pandekamat - 01fe22bcs279

Anoop U Kadakol - 01fe22bcs285

Rahul S Pujari - 01fe22bcs286

# ABSTRACT

The evolution of 5G networks brings transformative capabilities such as ultra-low latency, high bandwidth, and reliable connectivity, fundamental requirements for emerging applications in smart healthcare, IoT, autonomous systems, and real-time analytics. Despite these advancements, exclusive reliance on centralized cloud infrastructure can lead to increased latency and network congestion. This work presents an intelligent task offloading framework within a 5G edge-cloud architecture to address these challenges. Utilizing Open5GS as the 5G core, srsRAN for radio access, and a USRP B210 software-defined radio as the base station, the system establishes direct communication between user devices and edge servers. At the edge layer, containerized services are orchestrated using Kubernetes for efficient resource management, while cloud services are hosted on OpenStack-based virtual machines to provide high computational capacity when needed. A Long Short-Term Memory (LSTM)-based machine learning model continuously monitors and predicts resource utilization at the edge. Based on these predictions, the system dynamically determines whether to process tasks locally on the edge or offload them to the cloud. This adaptive, data-driven strategy enhances system responsiveness, optimizes resource allocation, and reduces task execution delays. Results demonstrate that edge-based task execution significantly improves overall system performance, particularly for latency-sensitive applications

**Keywords:** *5G, edge computing, task offloading, Open5GS, srsRAN, Kubernetes, OpenStack, LSTM, resource optimization, low latency*

# CONTENTS

<b>Acknowledgement</b>	<b>3</b>
<b>ABSTRACT</b>	<b>i</b>
<b>CONTENTS</b>	<b>iii</b>
<b>LIST OF FIGURES</b>	<b>iv</b>
<b>1 INTRODUCTION</b>	<b>1</b>
1.1 Motivation . . . . .	2
1.2 Literature Survey . . . . .	2
1.3 Problem Statement . . . . .	4
1.4 Objectives of Proposed Work . . . . .	5
1.5 Organization of the Report . . . . .	5
<b>2 REQUIREMENT ANALYSIS</b>	<b>6</b>
2.1 Functional Requirements . . . . .	6
2.2 Non Functional Requirements . . . . .	6
2.3 Hardware Requirements . . . . .	7
2.4 Software Requirements . . . . .	7
<b>3 SYSTEM DESIGN</b>	<b>8</b>
3.1 Architecture Design . . . . .	8
3.2 Flowchart . . . . .	9
<b>4 IMPLEMENTATION</b>	<b>12</b>
4.1 5G Architecture Implementation Using srsRAN and Open5GS . . . . .	12
4.2 Edge Layer Implementation Using Kubernetes . . . . .	15
4.2.1 Collection of Dataset to Train LSTM Model . . . . .	15
4.2.2 LSTM-Based Intelligent Scheduler and Offloading Logic . . . . .	16
4.2.3 Threshold-Based Decision Making . . . . .	17
4.2.4 Edge Task Flow . . . . .	17
4.3 Cloud Layer Implementation Using OpenStack . . . . .	17
4.4 Summary of Implementation Flow . . . . .	18

<b>5</b>	<b>RESULTS AND DISCUSSIONS</b>	<b>19</b>
5.1	Experimental Setup . . . . .	19
5.2	Performance Analysis . . . . .	20
5.2.1	Performance Evaluation <b>With and Without MEC</b> . . . . .	20
5.2.2	Performance Evaluation by Varying Load on Edge Devices . . . . .	22
5.2.3	Performance Evaluation by Varying Load on Edge Nodes . . . . .	24
5.2.4	Performance Evaluation by Varying Load on Edge node and Edge Device	26
<b>6</b>	<b>CONCLUSION AND FUTURE SCOPE</b>	<b>28</b>

# LIST OF FIGURES

3.1	System design . . . . .	8
3.2	Process Flow of Intelligent Task Offloading in Private 5g Network. . . . .	10
4.1	srsRAN . . . . .	12
4.2	Open5gs dashboard with user registered . . . . .	13
4.3	Rewriting of Opencell Sim Card . . . . .	14
4.4	Sample dataset . . . . .	15
4.5	Scheduler selecting the edge nodes based on resource utilization . . . . .	16
4.6	Images Processed by Nodes are Stored in Cloud(Openstack). . . . .	18
5.1	Latency Comparison Across Tasks With and Without MEC Deployment . . .	21
5.2	Comparison of Average Latency with and without MEC deployment . . . . .	21
5.3	Comparison of Average Throughput with and without MEC deployment . . .	22
5.4	Task Offloading Latency Comparison Across Tasks With and Without Load on Edge Device . . . . .	23
5.5	Average Latency Comparison With and Without Load on Edge Device . . . .	23
5.6	Average Throughput Comparison With and Without Load on Edge Device . .	24
5.7	Latency per Request Comparison With and Without Load on Edge Nodes . .	25
5.8	Average Latency Comparison With and Without Load on Edge Nodes . . . .	25
5.9	Average Throughput Comparison With and Without Load on Edge Nodes . .	26
5.10	Latency per Request . . . . .	26

# Chapter 1

## INTRODUCTION

In the rapidly evolving digital landscape, next-generation mobile networks are expected to support latency-sensitive and compute-intensive applications such as natural disaster response, autonomous vehicles, and industrial IoT systems like drones. Traditional cloud-based computing models often fail to meet these stringent real-time requirements due to latency from distant data centers and network congestion. To address these challenges, Intelligent Task Offloading has emerged as a pivotal approach, enabling dynamic and adaptive distribution of computational tasks across edge and cloud resources based on real-time conditions.

This work implements a real-world 5G-enabled intelligent task offloading system that autonomously decides whether to process data locally at the edge or remotely in the cloud. The system architecture leverages an Open5GS-based 5G core and srsRAN radio access stack, deployed on an edge server, with a USRP B210 software-defined radio acting as the base station. A mobile device connects to this private 5G network and offloads data depending on computational context. Central to this system is an AI-driven LSTM model that predicts future CPU usage on the edge node, enabling proactive decisions. When predicted load surpasses a predefined threshold, tasks are seamlessly redirected to a cloud server hosted on an OpenStack virtual machine.

This intelligent approach ensures low latency and high availability. The system integrates real-time monitoring, predictive analytics, and virtualization to optimize task allocation based on factors like CPU load, memory, and disk utilization. This report details the design and implementation of the Smart Task Offloading system, emphasizing the role of AI in load prediction, orchestration across heterogeneous infrastructures, and performance metrics. The proposed solution presents a scalable, adaptive framework to meet the demands of modern mobile networks while maintaining optimal Quality of Service (QoS) and resource utilization.



## 1.1 Motivation

Next-generation mobile networks are expected to support applications with stringent requirements for low latency, high computational power, and real-time responsiveness. Use cases such as autonomous vehicles, augmented reality, and industrial IoT demand rapid data processing to ensure safety and efficiency. Relying solely on centralized cloud infrastructure introduces high latency and increases backhaul traffic due to the physical distance between user devices and cloud data centers. To overcome these challenges, edge computing places resources closer to the user, and the integration of 5G further enhances performance by offering high bandwidth and ultra-low latency communication.

To effectively utilize this edge-cloud architecture, intelligent task offloading has emerged as a key enabler. This approach dynamically assigns tasks to edge or cloud nodes based on current network load, device capability, and system performance predictions. Machine learning models like Long Short-Term Memory (LSTM) networks can forecast metrics such as CPU usage to make proactive decisions and avoid overload conditions. By combining real-time monitoring with predictive analytics, intelligent offloading optimizes resource usage, minimizes delays, and ensures smooth, scalable operation—laying the foundation for responsive and reliable mobile computing systems.

## 1.2 Literature Survey

The paper [1] investigates using Convolutional Neural Networks (CNNs) to classify source code and assign code kernels to optimal compute units. It compares CNNs with LSTM networks and other graph-based methods, focusing on preprocessing and hyper-parameter tuning. Token filtering (Tf-Idf) and different data preparation strategies are explored to improve model performance. Results show that CNNs outperform LSTMs and traditional approaches in both accuracy and speed. The best CNN model achieved up to 86.2% accuracy and a strong Matthews Correlation Coefficient on unbalanced datasets. Overall, the study demonstrates that well-tuned CNNs are highly effective for source code classification tasks.

The paper [2] presents a CNN-based model for classifying source code by algorithm and programming language. The model processes code as token sequences and uses convolutional layers to extract key features for accurate classification. It achieved 96.56% accuracy in algorithm identification and 98.12% in language detection, outperforming LSTM and traditional machine learning models. Precision and recall for most classes exceeded 95%. The approach remained robust across different datasets and code styles. Careful preprocessing and tokenization further boosted the model's performance, demonstrating CNNs' effectiveness for automated code classification tasks.

The paper [3] applies Proximal Policy Optimization (PPO), a reinforcement learning method, for MEC task offloading. Unlike DQN or heuristic models, PPO continuously learns from changing network conditions. It improves QoS for URLLC users by reducing processing time by 4% and achieves 26% energy savings for mMTC users. The model adapts better in 5G scenarios with high device density and reliability requirements.

The paper [4] develops a hybrid Artificial Neural Network (ANN) that predicts optimal edge nodes for task execution based on current resource availability. Clustering is performed using the elbow method and updated every 10 minutes. The model achieves a Mean Squared Error (MSE) of 0.568, outperforming Gradient Boosting's MSE of 0.865, using real-world IoT data from Santander, Spain.

The paper [5], develops a DQN-based model focused on heterogeneous task types and channel allocation in MEC systems. The approach adapts to real-time network changes and minimizes task completion latency. The simulation demonstrates 8% lower latency and 15% performance improvement over heuristic and rule-based methods. The method efficiently allocates tasks even in bandwidth-constrained or dynamically changing environments.

The paper [6] proposes an IoT-based system for smart home security and automation using NodeMCU, various sensors (motion, gas, temperature), and a mobile app for remote control. The system detects intrusions, gas leaks, and monitors temperature, sending alerts to users and allowing them to control appliances from anywhere. Communication is achieved via Wi-Fi, and the cloud platform enables real-time updates. The authors implemented and tested the system in a real home environment. Results showed the system was reliable, cost-effective, and easy to use for enhancing home safety and convenience.

The paper [7] Federated Learning-Assisted Task Offloading Based on Feature Matching and Caching addresses privacy and redundancy in collaborative Device-Edge-Cloud networks. It uses Federated Learning for decentralized decision-making and feature similarity-based caching to avoid repetitive task execution. The proposed model reduces delay by 15.6% and energy consumption by 18.2%, offering intelligent and privacy-preserving offloading mechanisms.

The paper [8] titled "Deep Reinforcement Learning for Task Offloading in Mobile Edge Computing Systems" by Ming Tang and Vincent W. S. Wong presents a model-free deep reinforcement learning (DRL) approach to optimize task offloading decisions in mobile edge computing (MEC) environments. The authors address the challenge of dynamic edge node loads and the need for decentralized decision-making by mobile devices. Their proposed algorithm incorporates advanced DRL techniques, including Long Short-Term Memory (LSTM),

dueling deep Q-networks (DQN), and double-DQN, to enhance learning stability and performance. Simulation results demonstrate that the algorithm significantly reduces the ratio of dropped tasks by 86.4% to 95.4% and decreases average task delay by 18.0% to 30.1% compared to existing methods. This approach enables mobile devices to make efficient offloading decisions without prior knowledge of task models or other devices' actions, thereby improving overall system performance in MEC scenarios.

The paper [9] presents energy-efficient task offloading strategy in 5G-enabled mobile edge computing (MEC) environments. The authors model the energy consumption from both computation and communication aspects, incorporating transmission scheduling over fronthaul and backhaul links. They formulate an optimization problem aimed at minimizing the total energy consumption under constraints such as computational capacity and service delay. To solve this, they propose an Artificial Fish Swarm Algorithm (AFSA)-based approach, which efficiently explores the solution space and ensures global convergence. The proposed model demonstrates significant energy savings and performance improvement over traditional methods, with simulations confirming the scheme's robustness and scalability in realistic 5G small-cell network scenarios.

The paper [10] propose a hybrid approach combining Deep Q-Networks (DQN) with Federated Learning (FL) to enhance task offloading and resource allocation in Wireless Powered Communication (WPC) MEC environments. DQN is used for offloading decisions, while FL enables decentralized learning across heterogeneous devices. The system shows up to 20% improvement in energy efficiency and reduced execution delays compared to centralized deep reinforcement learning methods. This approach is especially effective in dynamic network conditions with diverse device capabilities.

### 1.3 Problem Statement

To develop an intelligent data offloading in edge computing environments to optimize resource allocation, reduce latency.

Edge devices often face resource constraints and high latency due to inefficient task handling. Traditional offloading methods are not adaptive to changing network and resource conditions, leading to poor performance in real-time and latency-sensitive applications. To address this, there is a need for an intelligent task offloading system that can dynamically optimize resource

allocation and reduce latency, ensuring efficient execution and improved quality of service in edge computing environments.

## 1.4 Objectives of Proposed Work

1. Set up a private 5G network using Open5GS and srsRAN.
2. Set up edge and cloud environments using Kubernetes and OpenStack.
3. Design an intelligent task offloading algorithm that dynamically allocates tasks between edge and cloud nodes based on real-time results or predicted resource utilization
4. Evaluate the model.

## 1.5 Organization of the Report

The report is divided into six chapters. Chapter 1 introduces the concept of intelligent task offloading in 5G networks, along with the motivation, problem statement, and objectives. Chapter 2 covers the functional and non-functional requirements, and the hardware and software setup. Chapter 3 presents the system design and architecture. Chapter 4 describes the implementation using Open5GS, srsRAN, Kubernetes, and OpenStack, along with the LSTM-based scheduling logic. Chapter 5 provides performance evaluation results under different scenarios such as with and without MEC, and under varying load conditions. Chapter 6 concludes the work and outlines the future scope of the project.

# Chapter 2

## REQUIREMENT ANALYSIS

### 2.1 Functional Requirements

- The edge devices must be able to send data to the edge node servers.
- The scheduler should be able to decide whether the task should be executed locally or in the cloud environment.
- Edge and cloud nodes must be able to process the user's task and send the result back to the requesting user.
- The edge nodes must be able to upload data to the cloud storage in OpenStack.

### 2.2 Non Functional Requirements

- The throughput achieved during task execution at the edge should be increased by more than 50% compared to execution in the cloud.
- The latency measured when processing at the edge should be reduced by at least 30% compared to task execution in the cloud.

## 2.3 Hardware Requirements

- **RAM:** 32 GB
- **Storage:** 100 GB SSD or higher
- **Processor:** Multi-core CPU (Intel i7 or equivalent recommended)
- **Base Station Hardware:** USRP B210 Software Defined Radio

## 2.4 Software Requirements

- **Operating System:** Ubuntu 22.04 LTS
- **Programming Language:** Python
- **Development Tools:** Visual Studio Code, Docker
- **Virtualization Platform:** OpenStack (DevStack) 2025.1
- **Container Orchestration:** Kubernetes (MicroK8s)
- **Machine Learning Framework:** TensorFlow or PyTorch

# Chapter 3

## SYSTEM DESIGN

The system comprises of three layers i.e user equipments layer,corelayer, edge layer and cloud layer.This section will give you the working and connection between the systems.

### 3.1 Architecture Design

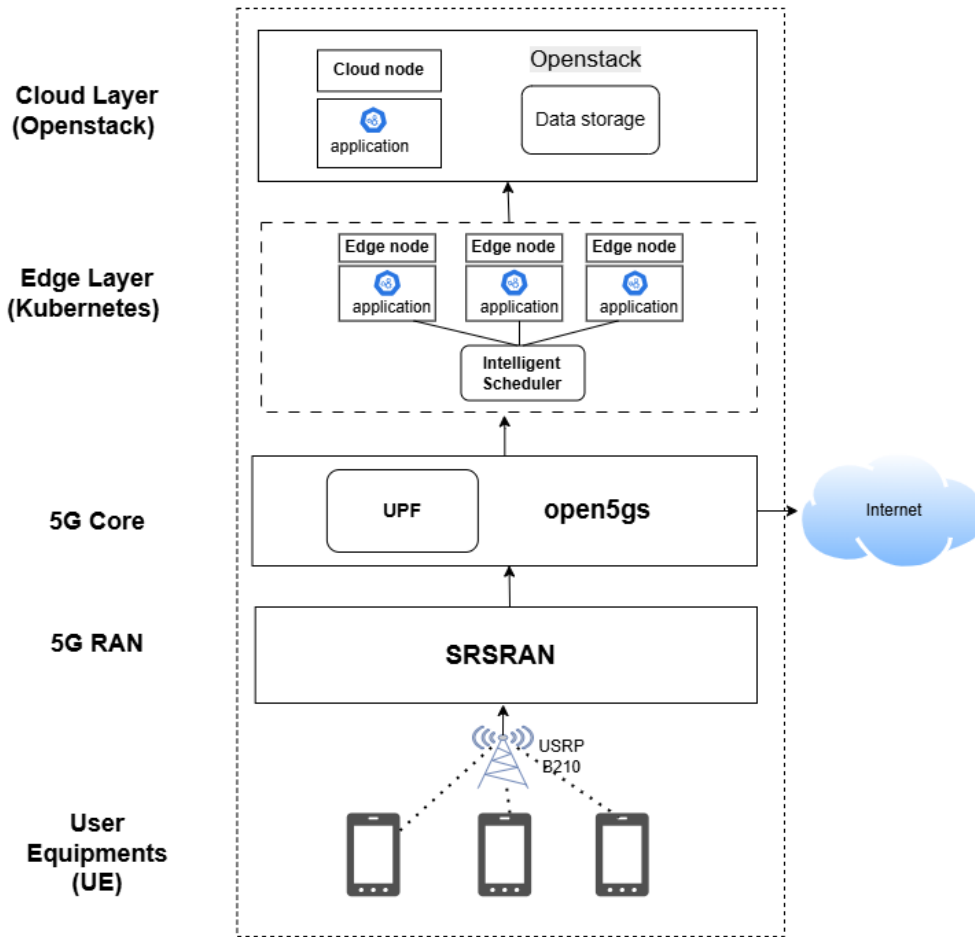


Figure 3.1: System design

The system model in Figure 3.1 illustrated represents the architecture of a 5G-enabled intelligent task offloading framework that integrates edge computing, machine learning, and cloud infrastructure. The architecture is structured into five distinct layers: the User Equipment

(UE) Layer, the Radio Access Network (RAN) Layer, the Core Layer, the Edge Layer, and the Cloud Layer.

User Equipment (UE) Layer: This layer consists of mobile devices (UEs) connected wirelessly via a 5G network. Communication is facilitated by a USRP B210 software-defined radio acting as the base station, which forms the Radio Access Network (RAN). These devices offload computation-intensive tasks to the network for processing.

Core Layer: The core layer is implemented using Open5GS as the 5G core network. Within Open5GS, key network functions such as the Session Management Function (SMF) and User Plane Function (UPF) manage routing, session handling, and packet forwarding. This layer establishes and maintains connectivity between UEs and the data processing layers, enabling seamless network registration and data flow.

Edge Layer (Kubernetes): Serving as an intermediate layer between the core and cloud, the edge layer comprises multiple edge nodes running containerized applications orchestrated by Kubernetes. These nodes provide computation near the user, significantly reducing latency. A real-time LSTM-based intelligent scheduler operates within this layer, continuously analyzing resource utilization metrics such as CPU and memory usage to determine optimal task placement. If the edge nodes have sufficient resources, tasks are executed locally to ensure low latency and faster response times. This dynamic, predictive offloading decision is a core feature of the system.

Cloud Layer (OpenStack): When the resource utilization of edge nodes exceeds a predefined threshold (30% CPU utilization), tasks are offloaded to cloud nodes. Data processed by both edge and cloud nodes are stored on virtual machines hosted on OpenStack. The cloud environment provides scalable computing power and storage for tasks requiring extensive processing or long-term archiving. Additionally, it acts as a backup in case of edge node failure.

## 3.2 Flowchart

The flowchart in Figure 3.2 represents the working of an intelligent task offloading system in a private 5G network. It highlights how user requests are dynamically routed between edge and cloud nodes. The offloading decision is driven by an LSTM-based scheduler that evaluates resource availability.



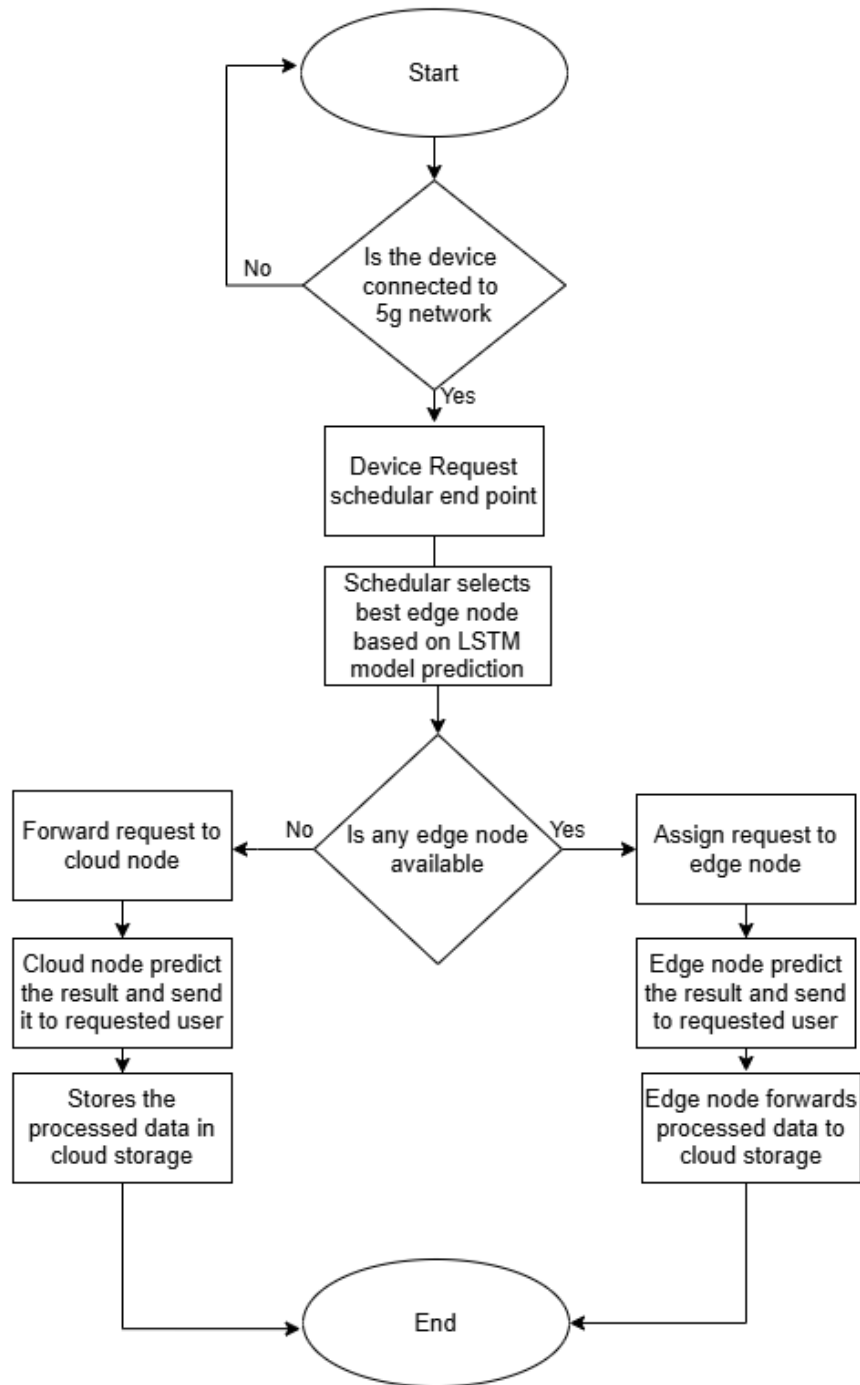


Figure 3.2: Process Flow of Intelligent Task Offloading in Private 5g Network.

Once the device is connected to the 5G network, it sends a task request to the scheduler endpoint. The scheduler analyzes current system metrics using a trained LSTM model to predict the future load on edge nodes. Based on this prediction, the scheduler determines whether any edge node has sufficient resources to handle the request. If a suitable edge node is available, the task is assigned to it for processing. The edge node then executes the task,

returns the result to the user, and forwards the processed data to cloud storage for persistence and future reference.

If all edge nodes are overloaded or unavailable, the scheduler redirects the request to a cloud node. The cloud node processes the task, sends the response back to the user, and stores the output in the cloud storage. This intelligent, predictive offloading mechanism ensures low latency, efficient resource utilization, and high system reliability by balancing the workload between local edge resources and the cloud.

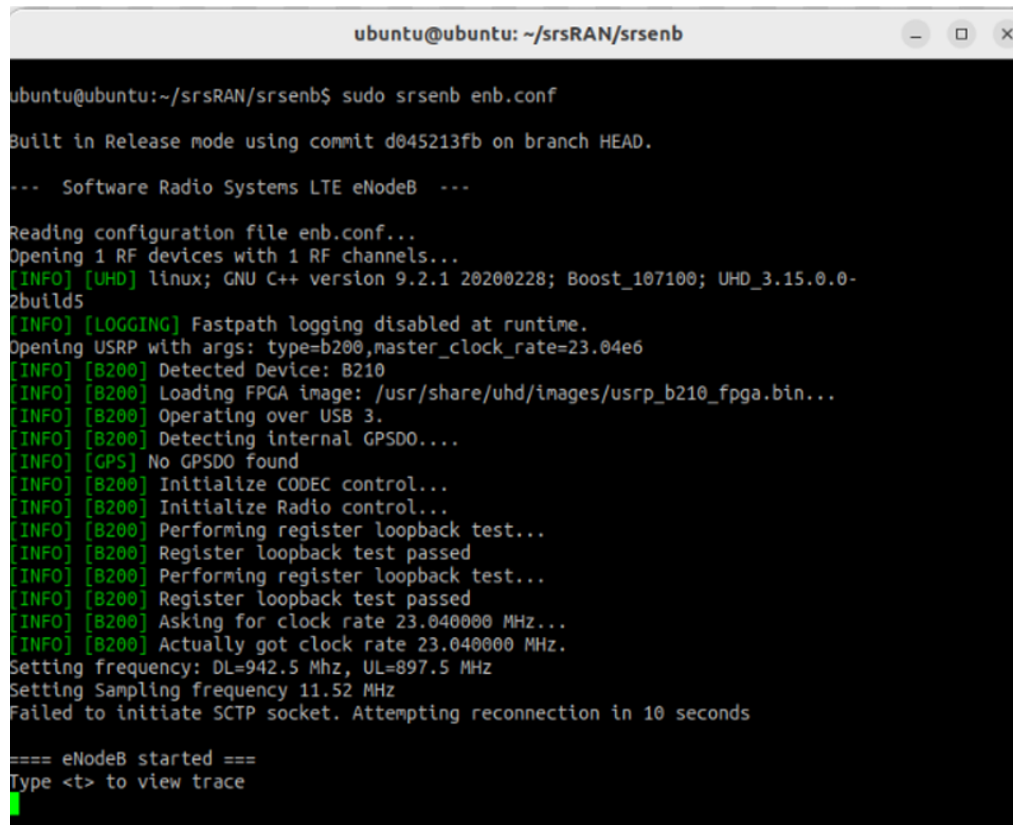
# Chapter 4

## IMPLEMENTATION

### 4.1 5G Architecture Implementation Using srsRAN and Open5GS

The foundation of the intelligent offloading system is built upon a private 5G network setup using **srsRAN** and **Open5GS**.

- **srsRAN** acts as the *Radio Access Network (RAN)* and interfaces with User Equipment (UE) such as smartphones via the **USRP B210** software-defined radio. It handles the physical and MAC layers and facilitates wireless communication with the 5G core.



```

ubuntu@ubuntu: ~/srsRAN/srsenb
ubuntu@ubuntu:~/srsRAN/srsenb$ sudo srsenb enb.conf
Built in Release mode using commit d045213fb on branch HEAD.

--- Software Radio Systems LTE eNodeB ---

Reading configuration file enb.conf...
Opening 1 RF devices with 1 RF channels...
[INFO] [UHD] linux; GNU C++ version 9.2.1 20200228; Boost_107100; UHD_3.15.0.0-2builds
[INFO] [LOGGING] Fastpath logging disabled at runtime.
Opening USRP with args: type=b200, master_clock_rate=23.04e6
[INFO] [B200] Detected Device: B210
[INFO] [B200] Loading FPGA image: /usr/share/uhd/images/usrp_b210_fpga.bin...
[INFO] [B200] Operating over USB 3.
[INFO] [B200] Detecting internal GPSDO....
[INFO] [GPS] No GPSDO found
[INFO] [B200] Initialize CODEC control...
[INFO] [B200] Initialize Radio control...
[INFO] [B200] Performing register loopback test...
[INFO] [B200] Register loopback test passed
[INFO] [B200] Performing register loopback test...
[INFO] [B200] Register loopback test passed
[INFO] [B200] Asking for clock rate 23.040000 MHz...
[INFO] [B200] Actually got clock rate 23.040000 MHz.
Setting frequency: DL=942.5 Mhz, UL=897.5 Mhz
Setting Sampling frequency 11.52 MHz
Failed to initiate SCTP socket. Attempting reconnection in 10 seconds

==== eNodeB started ====
Type <t> to view trace

```

Figure 4.1: srsRAN

Figure 4.1 illustrates the system design where the srsRAN components connect to the Open5GS core network using the USRP B210 as the software-defined radio interface. The USRP B210 facilitates realistic RF signal transmission between the User Equipment (UE) and the 5G Radio Access Network (RAN). This integration enables an end-to-end private 5G network simulation for task offloading and communication testing. The setup leverages open-source tools to provide flexibility and cost-effective experimentation in 5G environments.

- **Open5GS** functions as the *5G Core Network*, responsible for both control and user plane operations. It includes the following key components:

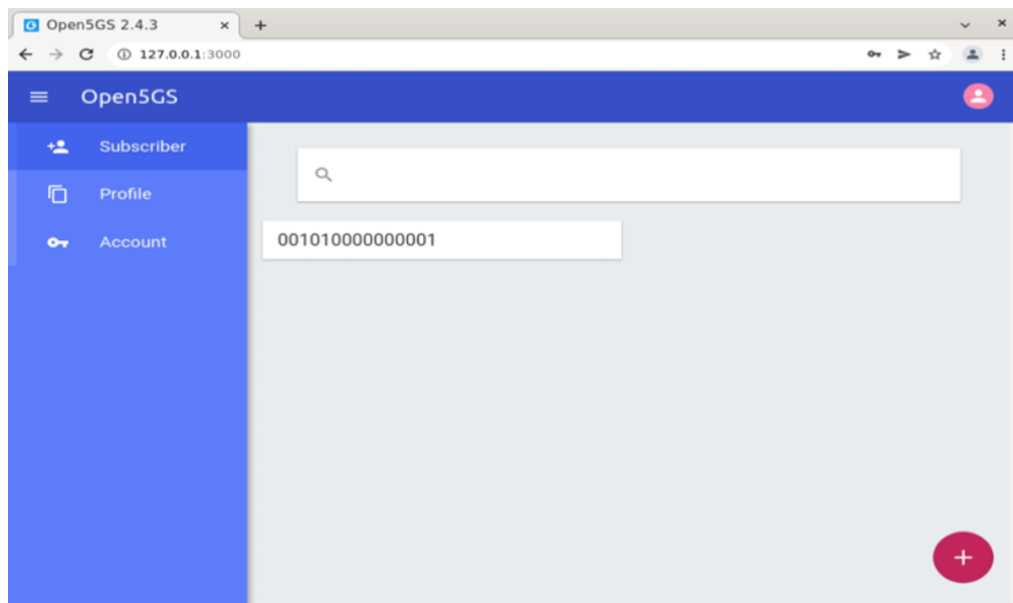


Figure 4.2: Open5gs dashboard with user registered

Figure 4.2 shows the Open5GS dashboard displaying a registered user within the 5G core network. This dashboard provides a real-time interface for monitoring user sessions, network status, and subscriber information. It is a key component for managing and visualizing network activities in the private 5G setup. Open5GS is widely used in research and development for its open-source core network functionalities.

- **NEF (Network Exposure Function)** – Provides secure APIs for third-party applications to access network services.
- **AUSF (Authentication Server Function)** – Authenticates users and devices trying to connect to the network.

- **UPF (User Plane Function)** – Manages data routing and forwarding between the user equipment and data networks.
  - **NRF (Network Repository Function)** – Maintains a registry of available network functions and their capabilities.
  - **PCF (Policy Control Function)** – Enforces policies for network resource usage, QoS, and charging.
  - **UDM (Unified Data Management)** – Stores and manages subscriber data and authentication information.
  - **SMF (Session Management Function)** – Manages session establishment, modification, and release, and allocates IP addresses.
  - **AMF (Access and Mobility Management Function)** – Handles device registration, connection, and mobility management.
  - **NSSF (Network Slice Selection Function)** – Selects appropriate network slices for different services and users.
- **OpenCells** Open5GS uses the OpenCells in Figure 4.3 SIM platform to register User Equipment (UE) by provisioning test SIM cards with specific IMSI (International Mobile Subscriber Identity), authentication keys, and network profiles compatible with the private 5G core. These programmable SIMs allow seamless integration with the Open5GS HSS/UDM, enabling user registration, authentication, and secure attachment to the network. This setup is especially valuable for testing and development, as it provides full control over SIM parameters and facilitates repeatable, customizable network experiments.

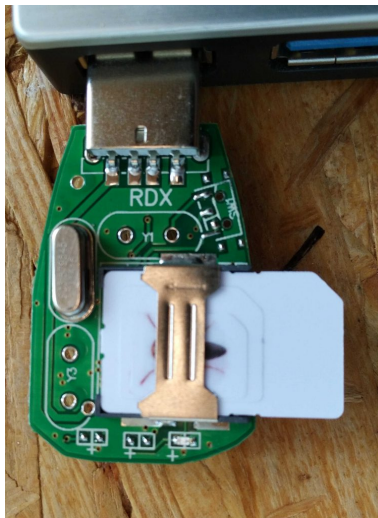


Figure 4.3: Rewriting of Opencell Sim Card

Together, this setup enables a fully operational standalone 5G network where UEs can send data to the edge network for further computation and intelligent decision-making.

## 4.2 Edge Layer Implementation Using Kubernetes

Once a user device is connected to the 5G network and transmits a data request, it reaches the **Edge Layer**, which consists of containerized **Edge Nodes** managed by **Kubernetes**.

Each edge node hosts lightweight applications capable of executing tasks offloaded from the UE. Kubernetes handles:

- Resource monitoring
- Service discovery
- Load-based scheduling

This decentralized edge setup ensures *low-latency and high-efficiency processing* close to the user, minimizing round-trip time compared to traditional cloud-based approaches.

### 4.2.1 Collection of Dataset to Train LSTM Model

	current_requests	cpu_percent	memory_percent	net_bytes_sent	net_bytes_recv	num_threads
0	1	48.4	39.9	774639	52353226	80
1	1	38.8	39.9	778815	52490615	60
2	1	72.1	40.0	782508	52644928	68
3	1	24.3	39.9	786906	52796831	60
4	1	17.4	40.1	793984	53307899	60
5	0	29.1	39.8	799740	53418895	50
6	0	71.2	39.8	809664	53696540	60

Figure 4.4: Sample dataset

The dataset in Figure 4.4 was used to train the LSTM model for intelligent task offloading decisions in an edge-cloud environment. It consists of 2,000 tuples, each collected at 10-second intervals, capturing real-time system behavior under varying loads. To simulate realistic conditions, the load on the edge device was varied randomly during data collection, reflecting diverse operating states of the system. The dataset includes features such as CPU usage (cpu\_percent), memory usage (memory\_percent), network bytes sent and received, number of active threads, and the number of current requests being handled by the system.

In Figure 4.4, each row represents a snapshot of system metrics. For instance, row 2 shows a high CPU usage (72.1%) while still handling a request, suggesting potential overload. This kind of temporal data helps the LSTM model learn patterns and predict when to offload tasks to maintain optimal performance.

### 4.2.2 LSTM-Based Intelligent Scheduler and Offloading Logic

At the core of the offloading decision is an **Intelligent Scheduler**, which integrates a **Long Short-Term Memory (LSTM)** model. This model continuously monitors the *CPU utilization* of each edge node.

The **LSTM model** is trained using historical CPU usage data collected from the edge servers under various workloads. By learning temporal patterns, it can *predict CPU utilization* for the next time interval with high accuracy.

The scheduler collects various metrics such as CPU, memory, and disk utilization every 30 seconds and predicts the resource usage for the next interval. It selects an edge node with the lowest predicted utilization. If no edge nodes are below the threshold, the request is offloaded to a cloud node for processing.

```
[Health Check] No backend under threshold; will use cloud fallback
[Predict] Forwarding to: http://10.2.2.114:8000
INFO: 10.9.1.161:65110 - "POST /predict HTTP/1.1" 200 OK
[Predict] Forwarding to: http://10.2.2.114:8000
INFO: 10.9.1.161:54654 - "POST /predict HTTP/1.1" 200 OK
[Predict] Forwarding to: http://10.2.2.114:8000
INFO: 10.9.1.161:55136 - "POST /predict HTTP/1.1" 200 OK
[Predict] Forwarding to: http://10.2.2.114:8000
INFO: 10.9.1.161:44840 - "POST /predict HTTP/1.1" 200 OK
[Predict] Forwarding to: http://10.2.2.114:8000
INFO: 10.9.1.161:26918 - "POST /predict HTTP/1.1" 200 OK
[http://10.9.1.161:30081] CPU: 0.9020, MEM: -0.1386, Score: 0.7634
[http://10.9.1.161:30082] CPU: -1.8442, MEM: 0.1051, Score: -1.7391
[http://10.9.1.161:30083] CPU: 0.8468, MEM: 0.6275, Score: 1.4742
[Health Check] Best backend selected: http://10.9.1.161:30082 (Score: -1.7391)
[Predict] Forwarding to: http://10.9.1.161:30082
INFO: 10.9.1.161:42622 - "POST /predict HTTP/1.1" 200 OK
[Predict] Forwarding to: http://10.9.1.161:30082
INFO: 10.9.1.161:65234 - "POST /predict HTTP/1.1" 200 OK
[Predict] Forwarding to: http://10.9.1.161:30082
```

Figure 4.5: Scheduler selecting the edge nodes based on resource utilization

The scheduler dynamically selects one of the under-utilized edge nodes to handle incoming user requests. If all edge nodes exceed the utilization threshold, the system triggers a fallback to a cloud backend. As shown in Figure 4.5, once any edge node's CPU and memory usage drops below the threshold, the scheduler re-evaluates and reassigns requests to that node. The health check component regularly monitors all nodes, computes scores based on CPU

and memory metrics, and identifies the best backend node (e.g., IP 10.9.1.161:30082) for optimal performance.

### 4.2.3 Threshold-Based Decision Making

The LSTM model's predictions determine whether a task is processed at the edge or offloaded to the cloud:

- If the **predicted CPU usage**  $\leq 30\%$ , the task is processed locally at the edge node.
- If the **predicted CPU usage of all edge nodes is**  $> 30\%$ , the system proactively **offloads the task to the cloud layer** to avoid overloading and maintain real-time responsiveness.

This **30% threshold** is a configurable parameter that balances *latency* and *resource utilization*, ensuring optimal system performance.

### 4.2.4 Edge Task Flow

1. The scheduler node accepts the request.
2. Kubernetes schedules the task on an available edge node.
3. Upon completion, results are returned to the user.
4. The data (e.g., images or logs) is uploaded to the cloud storage.

## 4.3 Cloud Layer Implementation Using OpenStack

For tasks that cannot be handled at the edge due to predicted high CPU load, the system utilizes the **Cloud Layer**. This layer consists of OpenStack nodes where offloaded tasks are executed and data from both edge and cloud nodes is stored.

This multi-tiered setup ensures **fault tolerance, scalability, and efficiency**. It allows for a flexible division of labor between the edge and cloud, adapting dynamically based on real-time system status and predictions from the LSTM model.



```

root@ln-1:/home/debian/express-app/images# ls
1747223803418-a.jpeg
root@ln-1:/home/debian/express-app/images# ls
1747223803418-a.jpeg 1747223882993-a.jpeg
root@ln-1:/home/debian/express-app/images# ls^C
root@ln-1:/home/debian/express-app/images# lsof -t :8080
COMMAND PID USER FD TYPE DEVICE SIZE/OFF NODE NAME
node 1821 root 19u IPv6 26540 0t0 TCP *:http-alt (LISTEN)
root@ln-1:/home/debian/express-app/images# cd ..
root@ln-1:/home/debian/express-app# node index.js
Server running on http://localhost:8080
root@ln-1:/home/debian/express-app# node index.js
Server running on http://localhost:8080
root@ln-1:/home/debian/express-app# cd images/
root@ln-1:/home/debian/express-app/images# ls
1747223803418-a.jpeg 1747228261186-a.jpeg 1747229382438-cat.jpg 1747230797027-a.jpeg
1747223882993-a.jpeg 1747228964327-test.jpg 1747229469410-a.jpeg 1747230012205-a.jpeg
1747228253417-a.jpeg 1747229201111-a.jpeg 1747229502673-a.jpeg 1747230014740-a.jpeg
root@ln-1:/home/debian/express-app/images# rm *
root@ln-1:/home/debian/express-app/images# ls
1747230868766-test.jpg 1747230869751-a.jpeg
root@ln-1:/home/debian/express-app/images#

```

Figure 4.6: Images Processed by Nodes are Stored in Cloud(Openstack).

In Figure 4.6 The images processed by the selected edge nodes and cloud node are stored in a cloud storage managed by **OpenStack**. A Debian-based virtual machine (VM) deployed within the OpenStack environment acts as the storage node. This cloud-based storage ensures reliability, scalability, and secure retention of processed image data. Utilizing OpenStack allows efficient management of virtualized resources and guarantees high availability of stored results.

## 4.4 Summary of Implementation Flow

1. UE connects to the 5G network (srsRAN + Open5GS).
2. The data/task is routed to the edge via the UPF.
3. The LSTM model predicts CPU usage on edge nodes.
4. Based on the 30% threshold:
  - If underutilized  $\Rightarrow$  process at edge.
  - If overloaded  $\Rightarrow$  offload to cloud.
5. The result is returned to the UE via the edge core.
6. The processed data is stored in the cloud storage.

# Chapter 5

## RESULTS AND DISCUSSIONS

### 5.1 Experimental Setup

For the performance testing of our system, the following setup has been used, as summarized in Table 5.1.

Table 5.1: Hardware and Software Tools Used in the Experimental Setup

Environment/Tool	Version
Ubuntu	v22.04
srsRAN	v23.11
Open5GS	v2.7.5
Base Station	USRP B210
SIM Cards	OpenCell SIMs
OpenStack (DevStack)	2025.1 (Epoxy)
Kubernetes	MicroK8s (v1.33)

In our implementation of intelligent task offloading within a private 5G network, we used several key tools and platforms to set up, manage, and monitor the environment. srsRAN (v23.11) was used as the 5G Radio Access Network (RAN) to provide connectivity between the user devices (UEs) and the core network. The USRP B210, a software-defined radio, was employed to handle real RF signals, giving us realistic wireless communication for testing. Open5GS (v2.7.5) was set up on Ubuntu and acted as the 5G core network, managing functions such as mobility, sessions, and the user plane to ensure 5G connectivity was established properly.

To create a cloud environment for offloaded tasks, we used DevStack, a development version of OpenStack based on the 2025.1 “Epoxy” release. This cloud platform is designed to run tasks that are moved away from the edge when edge resources are fully used. For the edge layer, we deployed MicroK8s (v1.33), a small but powerful Kubernetes distribution that runs

containerized services. MicroK8s hosts the intelligent task scheduling components, including an LSTM-based model that makes decisions about task placement.

Together, these tools (as detailed in Table 5.1) form the main part of our intelligent task offloading testbed. This setup allows us to dynamically manage and move tasks between the edge and the cloud depending on resource availability, ensuring that our private 5G network runs efficiently and adapts to varying computational demands.

## 5.2 Performance Analysis

The results are analyzed under different scenarios, each designed to evaluate the performance of intelligent task offloading under varying load conditions. The analysis focuses on a single evaluation parameter across different algorithms. We measured delay with MEC and without MEC 5.2.1 and Specifically, we consider three scenarios: (i) load applied on the Edge devices (user equipment) 5.2.2, (ii) load applied solely on the edge nodes (Kubernetes pods), 5.2.3 and (iii) load applied on both the user devices and edge nodes simultaneously 5.2.4. These scenarios allow us to assess how effectively the system adapts to dynamic workload distributions and to evaluate the behavior of the LSTM-based scheduler under real-time offloading conditions. The detailed analysis for each scenario is presented below.

### 5.2.1 Performance Evaluation With and Without MEC

Figure 5.1 presents the latency comparison across 10 task instances, with and without Mobile Edge Computing (MEC) deployment. For all 10 instances, the latency with MEC is significantly lower, indicating faster task processing due to edge-localized computation. Without MEC, higher latency values occur consistently, reflecting the overhead of offloading tasks to distant servers. These results clearly demonstrate the performance advantages of MEC in reducing task offloading delays across multiple instances.

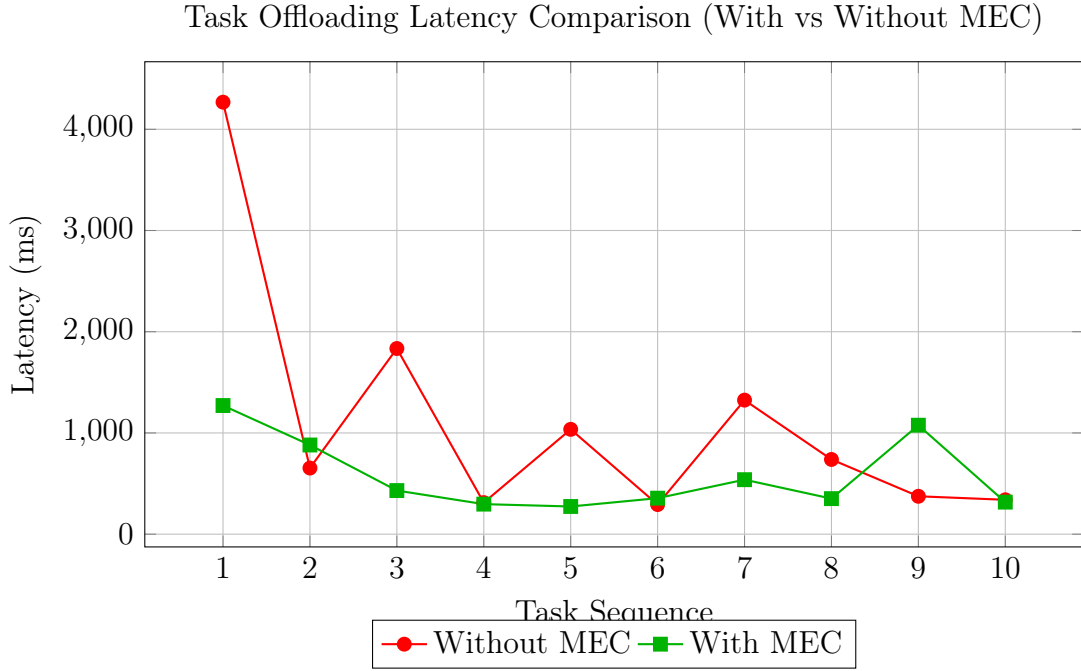


Figure 5.1: Latency Comparison Across Tasks With and Without MEC Deployment

Figure 5.2 compares the average latency between scenarios with and without MEC deployment. The average latency with MEC is significantly lower at 579.09 ms, compared to 1117.13 ms without MEC. This highlights the effectiveness of MEC in reducing processing delays by bringing computation closer to the edge.

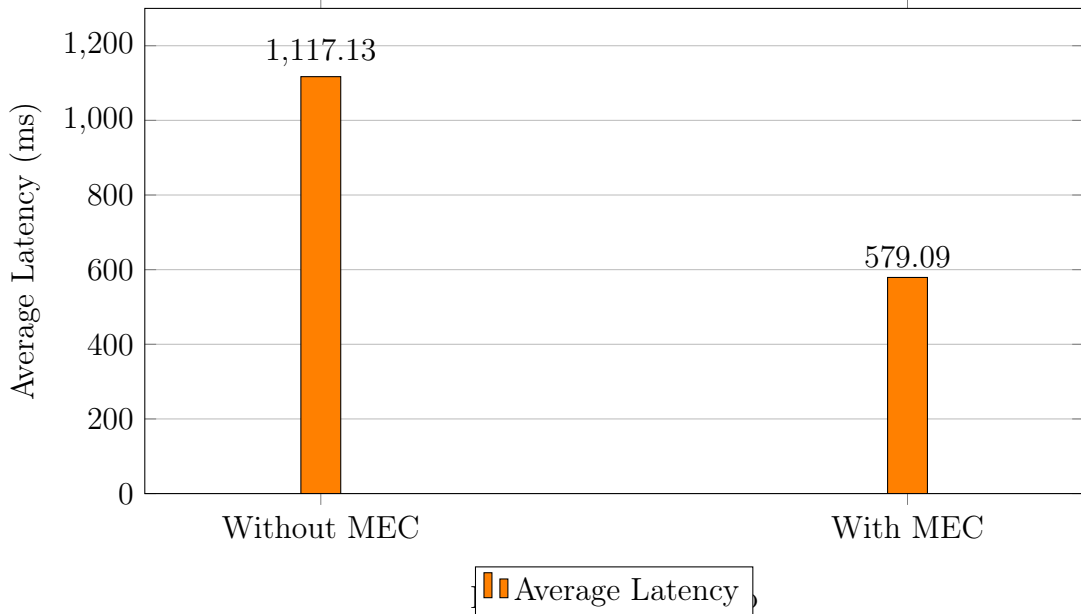


Figure 5.2: Comparison of Average Latency with and without MEC deployment

Figure 5.3 shows the average throughput comparison between deployments with and with-

out MEC. Throughput nearly doubles with MEC deployment, increasing from 1.71 Gbps to 3.31 Gbps. This demonstrates MEC's capability to enhance data transmission efficiency by processing closer to the source.

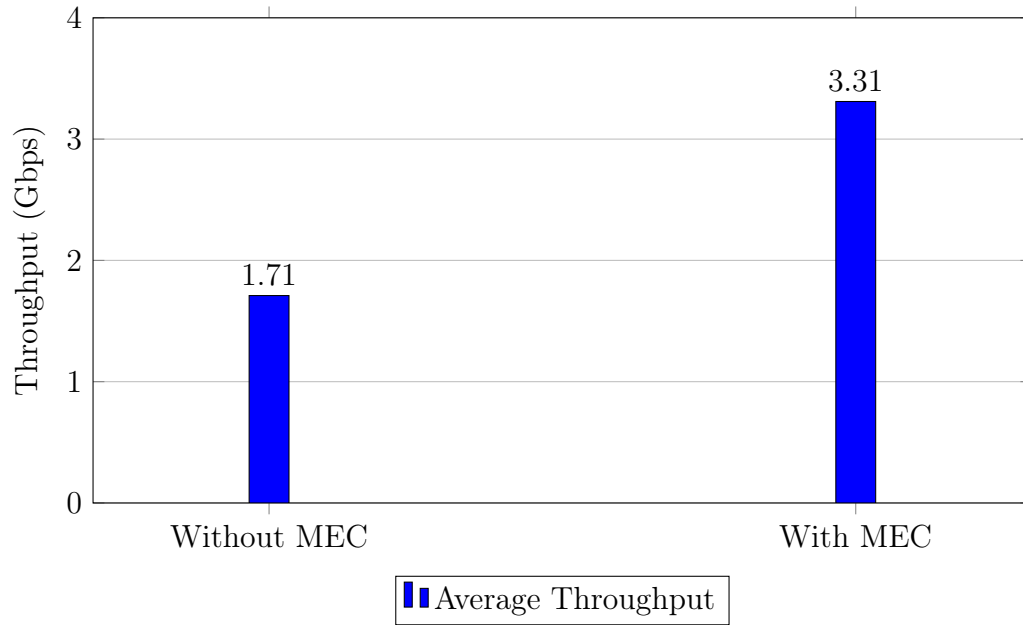


Figure 5.3: Comparison of Average Throughput with and without MEC deployment

### 5.2.2 Performance Evaluation by Varying Load on Edge Devices

Figure 5.4 compares task offloading latency across 10 instances with and without load on the edge device. Latency significantly increases when the edge device is under load, with some tasks experiencing delays over 5000 ms. Without load, latency remains consistently lower, typically under 1000 ms. This highlights how device load adversely impacts offloading performance and response times.

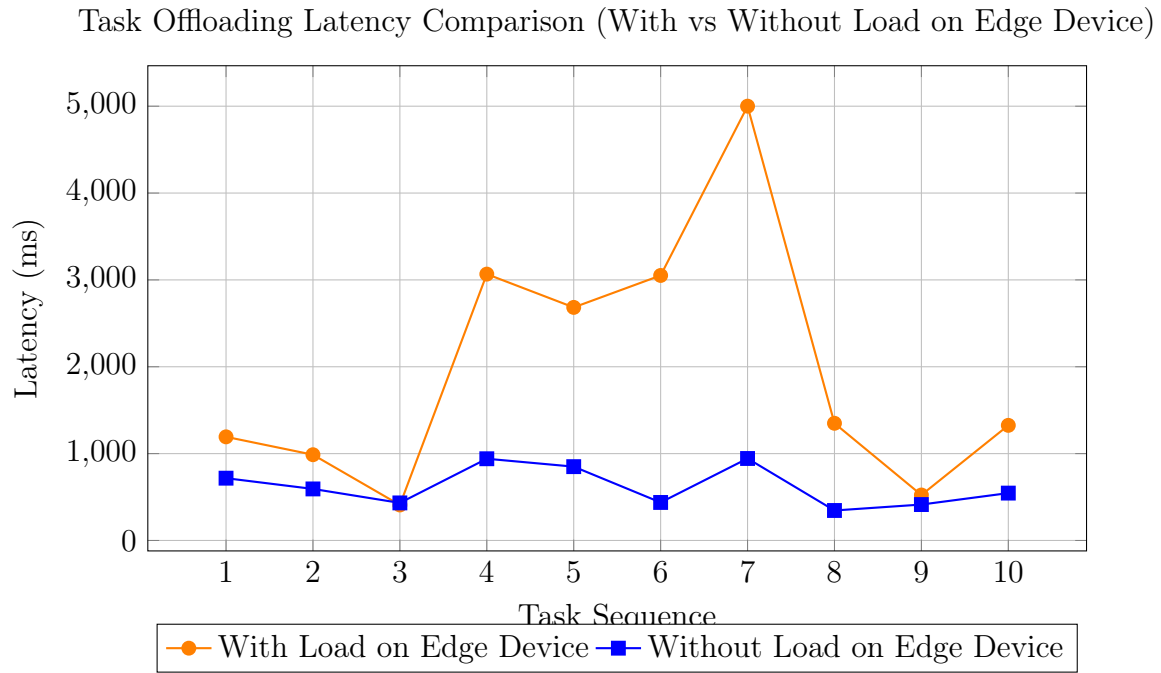


Figure 5.4: Task Offloading Latency Comparison Across Tasks With and Without Load on Edge Device

Figure 5.5 shows the average latency comparison with and without load on the edge device. The latency is significantly higher when the edge device is under load, reaching nearly 1959 ms. Without load, the average latency reduces substantially to about 622 ms, demonstrating the impact of device load on performance.

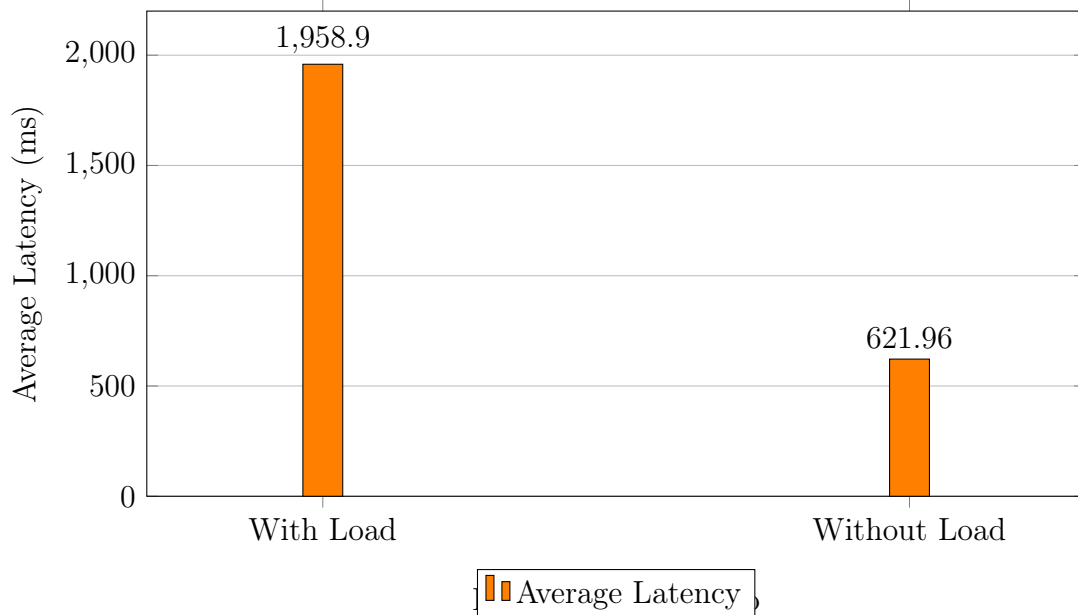


Figure 5.5: Average Latency Comparison With and Without Load on Edge Device

Figure 5.6 illustrates the average throughput comparison with and without load on the edge device. Throughput significantly decreases to 0.98 Mbps under load conditions. Without load, the throughput improves markedly to 3.08 Mbps, highlighting the effect of edge device load on data transmission performance.

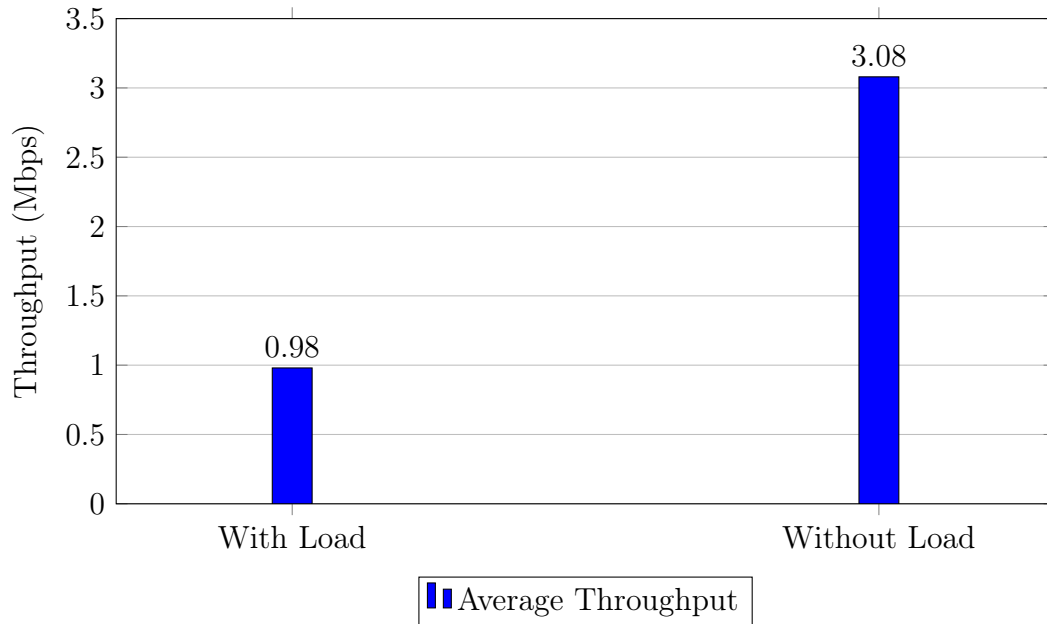


Figure 5.6: Average Throughput Comparison With and Without Load on Edge Device

### 5.2.3 Performance Evaluation by Varying Load on Edge Nodes

Figure 5.7 compares upload latency per request across 10 instances with and without load on the edge node. Latency values are consistently higher under load, reaching up to 3104.47 ms. Without load, latency remains stable and low, around 700–800 ms. This highlights the significant impact of edge node load on upload latency performance.

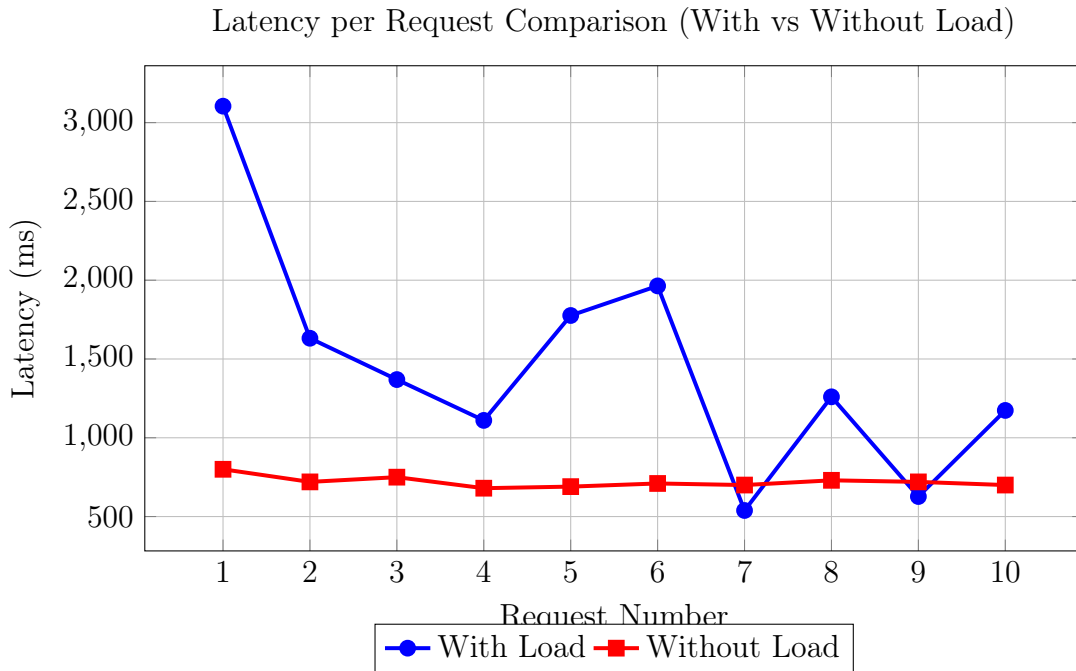


Figure 5.7: Latency per Request Comparison With and Without Load on Edge Nodes

Figure 5.8 shows the average latency comparison with and without load on the edge node. Latency nearly doubles under load, increasing from 720 ms to 1455.48 ms. This demonstrates the substantial impact of edge node load on system responsiveness.

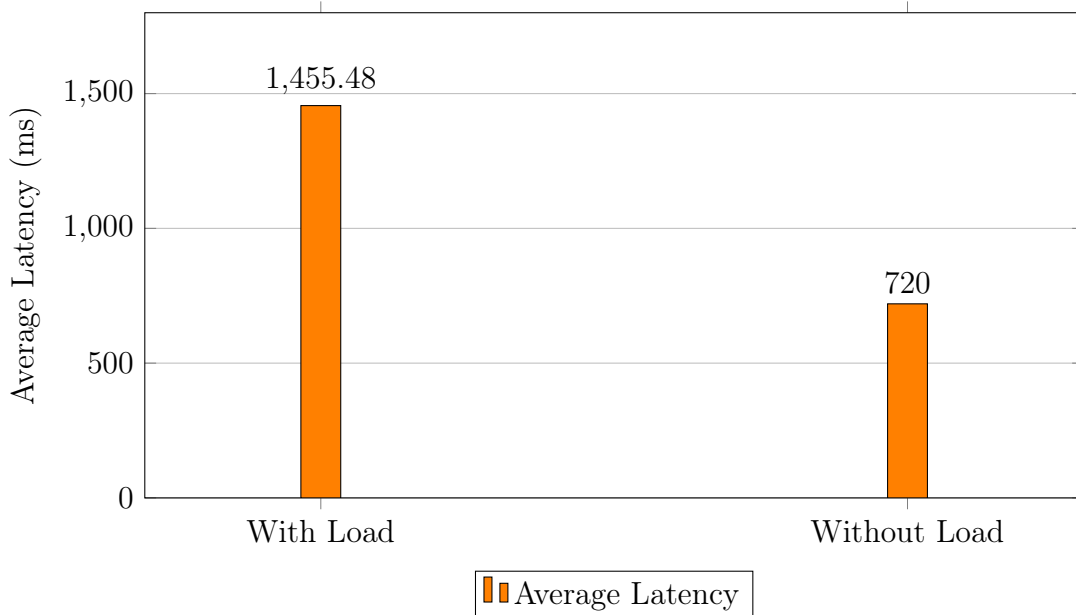


Figure 5.8: Average Latency Comparison With and Without Load on Edge Nodes

Figure 5.9 compares average upload throughput with and without load on the edge node. Throughput decreases from 1.85 Mbps without load to 1.32 Mbps under load. This highlights



the negative impact of edge node load on data transmission efficiency.

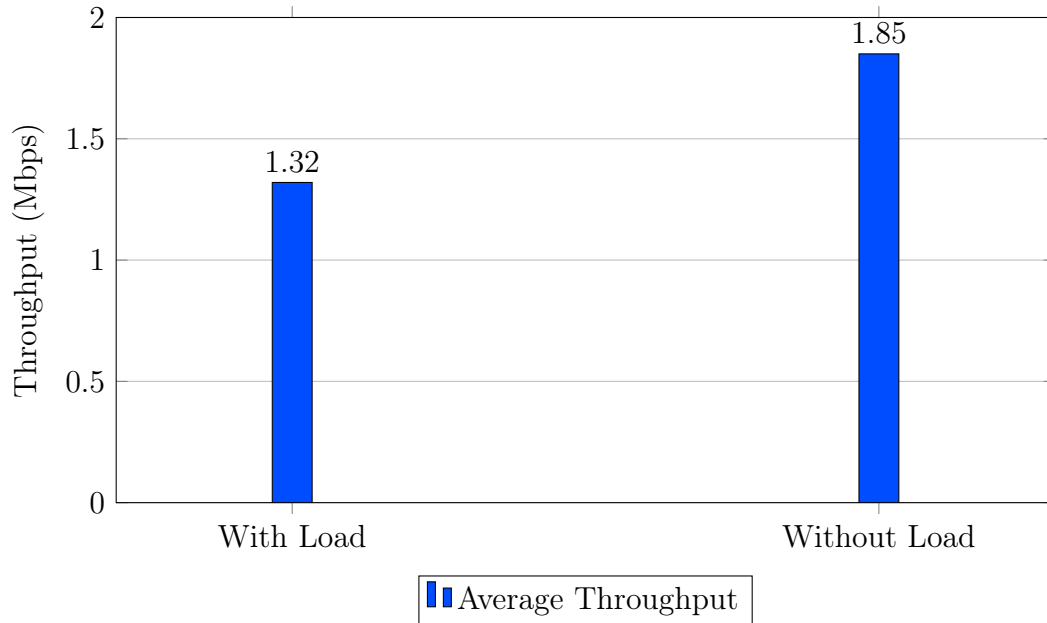


Figure 5.9: Average Throughput Comparison With and Without Load on Edge Nodes

#### 5.2.4 Performance Evaluation by Varying Load on Edge node and Edge Device

Figure 5.10 shows the upload latency for 10 individual requests under load conditions. Latency values vary significantly, ranging from a high of 5853.45 ms (Request 3) to a low of 449.55 ms (Request 10).

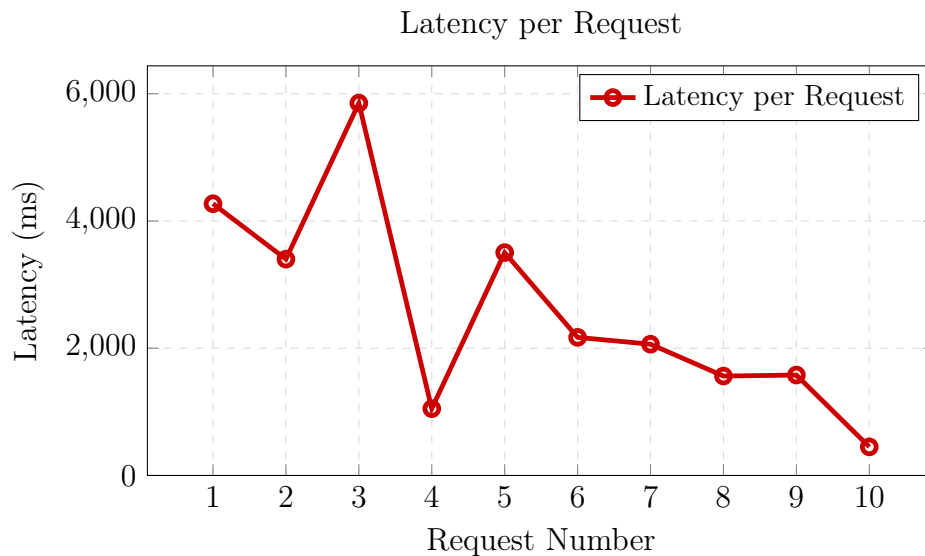


Figure 5.10: Latency per Request

Table 5.2: Average Latency and Throughput by Varying Load on Both Edge Nodes and Edge Devices

<b>Metric</b>	<b>Value</b>
Average Latency (ms)	2590.33
Average Throughput (Mbps)	0.74

Table 5.2 presents the average latency and throughput measured under varying load conditions on both edge nodes and edge devices. The latency, recorded at approximately 2590.33 ms, reflects the total time taken for task offloading and execution under load. The throughput, observed at 0.74 Mbps, indicates the data transmission rate during this process. These values help in assessing the performance impact when system resources are stressed.

# Chapter 6

## CONCLUSION AND FUTURE SCOPE

This work focused on the implementation of a 5G Offloading System designed to improve network efficiency by dynamically redistributing user traffic across available network resources. Using a 5G environment built on Open5GS and srsRAN, the system demonstrates how traffic offloading can reduce congestion, reduce latency, and enhance overall Quality of Service (QoS) for end-users. The setup enabled the redirection of high-load traffic from one gNodeB or UPF to another based on real-time traffic conditions, showcasing the potential of intelligent offloading in handling the massive data demands typical in 5G networks. While the system performed effectively in the testbed environment, it is clear that further validation is needed in more realistic and larger-scale deployments to fully assess its robustness, adaptability, and performance benefits.

Future enhancements could include integrating Multi-access Edge Computing (MEC) by altering the UPF to forward the traffic to the outsource internet, further reduce latency for time-sensitive applications, supporting handover-aware offloading to maintain session continuity, and incorporating support for 5G core slicing to manage diverse service types more efficiently. Additionally, expanding the system to handle more complex topologies and the private autonomous machine control system in the large scale industry.

# REFERENCES

- [1] G. Carvalho, B. Cabral, V. Pereira, and J. Bernardino. Computation offloading in edge computing environments using artificial intelligence techniques. *Engineering Applications of Artificial Intelligence*, 95:103840, 2020.
- [2] P. Choi and J. Kwak. A survey on mobile edge computing for deep learning. In *2023 International Conference on Information Networking (ICOIN)*, pages 652–655, 2023.
- [3] A. Ebrahimi and F. Afghah. Intelligent task offloading: Advanced mec task offloading and resource management in 5g networks. In *2025 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1–6, 2025.
- [4] R. Hamadi, A. Khanfor, H. Ghazzai, and Y. Massoud. A hybrid artificial neural network for task offloading in mobile edge computing. In *2022 IEEE 65th International Midwest Symposium on Circuits and Systems (MWSCAS)*, pages 1–4, 2022.
- [5] T. Jiang, Z. Chen, Z. Zhao, M. Feng, and J. Zhou. Deep-reinforcement-learning-based task offloading and resource allocation in mobile edge computing network with heterogeneous tasks. *IEEE Internet of Things Journal*, 12(8):10899–10906, 2025.
- [6] T. Nishio and R. Yonetani. Client selection for federated learning with heterogeneous resources in mobile edge. In *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, pages 1–7, 2019.
- [7] J. Tang, S. Wang, S. Yang, Y. Xiang, and Z. Zhou. Federated learning-assisted task offloading based on feature matching and caching in collaborative device-edge-cloud networks. *IEEE Transactions on Mobile Computing*, 23(12):12061–12079, 2024.
- [8] M. Tang and V. W. Wong. Deep reinforcement learning for task offloading in mobile edge computing systems. *IEEE Transactions on Mobile Computing*, 21(6):1985–1997, 2022.
- [9] L. Yang, H. Zhang, M. Li, J. Guo, and H. Ji. Mobile edge computing empowered energy efficient task offloading in 5g. *IEEE Transactions on Vehicular Technology*, 67(7):6398–6409, 2018.
- [10] L. Zang, X. Zhang, and B. Guo. Federated deep reinforcement learning for online task offloading and resource allocation in wpc-mec networks. *IEEE Access*, 10:9856–9867, 2022.