Sudan University
of Science & Technology
College of Engineering
Electronics Engineering Department

Adaptive Federated Learning-Based Resource Management in Mobile Edge Computing for Machine Type Communication in 5G Network

A Research Submitted in Partial Fulfillment of the Requirements for the Degree of B.Sc. in Electronics Engineering

Prepared By:

Asia Easa Yousuf Ahmed
Esraa Yahya Zakaria Hassan
Omnia Ahmed Abd Elgadir Ali
Ruaa Saadeldin Abdelhameed Mohamed
Yusra Yahya Zakaria Hassan

Supervised By: Prof. Dr. Rashid A. SAEED

Sept 2026

بسم الله الرحمن الرحيم

﴿ يَرفَعِ اللَّهُ الَّذينَ آمَنوا مِنكُم وَالَّذينَ أوتُوا العِلمَ دَرَجاتٍ وَاللَّهُ بِما تَعمَلونَ خَبيرٌ ﴾

[المجادلة: 11]

DECLARATION

We, Asia Easa Yousuf Ahmed, Esraa Yahya Zakaria Hassan, Omnia Ahmed Abd Elgadir Ali, Ruaa Saadeldin Abdelhameed Mohamed, and Yusra Yahya Zakaria Hassan, hereby declare that the research work titled: "Adaptive Federated Learning-based Resource Management in Mobile Edge Computing for Machine Type communication in 5G Network". This research work, submitted to the Electronics Engineering Department, College of Engineering, Sudan University of Science & Technology, in partial fulfillment of the requirements for the degree of Bachelor in Electronics Engineering, represents our original work. We affirm that this research has not been submitted previously, in whole or in part, for any other degree or academic award.

Supervised By: Prof. Dr. Rashid A. Saeed

Prepared By:

Asia Easa Yousuf Ahmed
Esraa Yahya Zakaria Hassan
Omnia Ahmed Abd Elgadir Ali
Ruaa Saadeldin Abdelhameed Mohamed
Yusra Yahya Zakaria Hassan

DEDICATION

This research is dedicated to our beloved parents, whose love, guidance, and unwavering support have been our greatest source of strength. We also dedicate this work to our friends and family, who encouraged us and stood by us throughout this journey. Your faith in us has made this achievement possible.

ACKNOWLEDGMENT

We would like to express our sincere gratitude to Prof.Dr. Rashid A. Saeed for his continuous guidance, valuable feedback, and dedicated supervision throughout the development of this research. Our appreciation also extends to the Sudan University of Science and Technology – College of Engineering, Electronics Engineering Department, for providing the academic support and resources that enabled the successful completion of this work.

We are deeply grateful to our families, whose love, encouragement, and patience have been our strength throughout this journey. Their unwavering support has been essential in helping us overcome challenges and remain motivated.

We also extend our heartfelt thanks to our friends, who stood by us, offered support, and shared this academic journey with kindness and positivity. Their encouragement has been truly appreciated.

ABSTRACT

With the rapid expansion of intelligent devices and the growing demand for real-time data processing, massive volumes of data are now generated and stored at the network edge rather than in centralized clouds. To ensure data privacy while enabling collaborative model training, Federated Learning (FL) has emerged as a promising paradigm that allows devices to train local models using private data and share only model parameters with a central server. However, due to limited wireless bandwidth and constrained energy resources in mobile devices, it is impractical for FL to perform model updates and aggregation across all participating devices simultaneously. Furthermore, selecting appropriate devices for each training round is a critical challenge that directly impacts resource utilization, energy efficiency, and communication latency. In 5G-enabled Mobile Edge Computing (MEC) environments supporting Machine Type Communication (MTC), these challenges are further amplified by device heterogeneity and dynamic network conditions, making efficient coordination and resource management essential for maintaining performance and scalability. To address these issues, we propose an adaptive resource management framework for MEC-assisted FL that integrates intelligent device selection and dynamic resource allocation using a Double Deep Q-Network (DDQN). The device selection process is modeled as a Markov Decision Process (MDP), allowing the DDQN to make dynamic and experience-driven decisions that balance energy consumption, computational load, and bandwidth utilization under varying network and device conditions. By combining

5th generation network (5GN) capabilities with FL on the MNIST dataset, the framework achieves an intelligent, low-latency, and energy-efficient edge learning environment—paving the way for scalable, privacy-preserving, and real-time distributed Artificial Intelligence (AI) systems for MTC applications.

الخلاصة

مع التوسع السريع في انتشار الأجهزة الذكية، والطلب المتزايد على معالجة البيانات في الوقت الفعلي، أصبحت كميات هائلة من البيانات تُولَّد وتُخزَّن عند حافة الشبكة بدلًا من تخزينها في السحابة المركزية. ولضمان خصوصية البيانات مع تمكين التدريب التعاوني للنماذج، برز التعلّم الاتحادي (FL) بوصفه نموذجاً واعداً يتيح للأجهزة تدريب نماذج محلية باستخدام بياناتها الخاصة، ومشاركة معلمات النماذج فقط مع الخادم المركزي. ومع ذلك، ونظرًا لمحدودية عرض النطاق الترددي اللاسلكي وقيود موارد الطاقة في الأجهزة المحمولة، فإن تنفيذ تحديثات النماذج وعمليات التجميع عبر جميع الأجهزة المشاركة في الوقت نفسه يُعد أمرًا غير عملي. وعلاوة على ذلك، يمثل اختيار الأجهزة المناسبة لكل جولة تدريب تحديًا أساسيًا، إذ يؤثر بصورة مباشرة في كفاءة استخدام الموارد، وكفاءة استهلاك الطاقة، وزمن الاتصال. وفي بيئات الحوسبة الطرفية المتنقلة (MEC) المدعومة بشبكات الجيل الخامس (5G) والتي تدعم اتصالات الآلات (MTC)، تزداد هذه التحديات تعقيدًا نتيجة تباين قدرات الأجهزة وديناميكية ظروف الشبكة، مما يجعل التنسيق الفعّال وإدارة الموارد أمرين ضروريين للحفاظ على الأداء وقابلية التوسع. ولمعالجة هذه المشكلات، نقترح إطارًا متكيفًا لإدارة الموارد في التعلّم الاتحادي المدعوم بالحوسبة الطرفية المتنقلة (MEC)، يدمج بين الاختيار الذكي للأجهزة والتخصيص الديناميكي للموارد باستخدام خوارزمية DDQN. وقد جرى نمذجة عملية اختيار الأجهزة باعتبارها عملية قرار ماركوف (MDP)، مما يتيح لخوارزمية DDQN اتخاذ قرارات ديناميكية قائمة على الخبرة، تحقق توازنًا بين استهلاك الطاقة، والحمل الحاسوبي، واستخدام عرض النطاق الترددي، في ظل اختلاف ظروف الشبكة والأجهزة وتغيرها. ومن خلال دمج إمكانات شبكات الجيل الخامس (5G) مع التعلّم الاتحادي وتطبيقه على مجموعة بيانات MNIST، يحقق الإطار المقترح بيئة تعلّم طرفية ذكية تتميز بانخفاض زمن الاستجابة وكفاءة استهلاك الطاقة، مما يمهّد الطريق نحو أنظمة ذكاء اصطناعي موزعة قابلة للتوسع، وتحافظ على خصوصية البيانات، وتدعم المعالجة في الوقت الفعلي لتطبيقات اتصالات الآلات (MTC).

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER ONE

INTRODUCTION

1.1 Introduction

The emergence of fifth generation (5G) networks has ushered in a new era of connectivity, promising unprecedented speed, reliability, and support for diverse applications, including Machine Type Communication (MTC). MTC is characterized by massive device connectivity, ultra-reliable low-latency communication, and energy-efficient operations, making it a cornerstone of modern Internet of Things (IoT) applications [1]. However, these features also pose significant challenges in terms of resource management, scalability, and maintaining the quality of service in dynamic network environments [2, 3].

Mobile Edge Computing (MEC) has emerged as a transformative solution by bringing computational resources closer to the edge of the network, reducing latency and offloading processing tasks from centralized cloud systems [4]. While MEC significantly enhances system performance, its integration with MTC demands efficient resource management to handle the diverse capabilities and constraints of edge devices [5]. Traditional resource management approaches fall short in addressing the dynamic resource availability, device heterogeneity, and massive data traffic generated by MTC devices [6].

Federated Learning (FL), a decentralized machine learning paradigm, has the potential to address these challenges. By allowing devices to train machine learning models while keeping data locally, FL ensures data privacy, reduces bandwidth usage, and distributes computational workloads. However, existing FL implementations in MEC environments

are not fully optimized for the unique demands of MTC applications. Specifically, they struggle to adapt to fluctuating network conditions, diverse device capabilities, and the massive scale of MTC deployments [7].

This research proposes an Adaptive FL based Resource Management Framework tailored for MEC environments to address these challenges. By leveraging advanced techniques such as Deep Reinforcement Learning (DRL), the framework dynamically optimizes resource allocation and device selection in MEC systems, ensuring energy efficiency, minimizing latency, and enhancing scalability. The proposed solution not only addresses the constraints of MTC environments but also aligns with the privacy-preserving principles of FL, enabling efficient and reliable machine learning for 5G-enabled IoT ecosystems [8].

This study aims to bridge the gap between MEC and FL for MTC applications by introducing an innovative framework that adapts to the dynamic nature of 5G networks. By validating the framework through simulation tools and performance metrics such as latency, energy efficiency, and resource utilization, this research seeks to contribute to the broader goal of scalable and privacy-preserving edge computing in the era of 5G.

1.2 Problem Statement

The rapid proliferation of MTC in 5G networks has introduced substantial challenges for MEC systems, particularly in managing resources efficiently under massive connectivity and low-latency requirements [9]. MTC applications generate many simultaneous device connections, each demanding quick and reliable data transmission, which increases the burden on MEC infrastructures. This results in difficulties related to scalability, energy consumption, and maintaining consistent quality of service. In this context, efficient resource management becomes a critical factor in ensuring that computational tasks are properly offloaded from the core network to the edge, improving system responsiveness, and overall network performance [6].

However, existing resource management strategies in MEC often fail to cope with the dynamic and heterogeneous nature of 5G networks. Devices differ in energy capacity, processing power, and communication bandwidth, while network conditions fluctuate rapidly. As a result, static or rule-based resource allocation mechanisms tend to cause inefficient computation, unbalanced energy consumption, and delayed task execution in large-scale MTC environments [10]. To overcome these challenges, FL has emerged as a promising paradigm that enables distributed model training across multiple IoT devices without sharing raw data, thus enhancing privacy and scalability. Nevertheless, most existing FL frameworks deployed in MEC are not adaptive to the dynamic variations of device resources and network states. This lack of adaptiveness leads to

degraded model performance, high communication overhead, and suboptimal energy usage, especially in MTC scenarios involving diverse devices and data distributions [6, 10].

Therefore, this research addresses the problem of developing an adaptive FL based resource management framework for MEC in 5G networks, specifically designed to meet the demands of MTC applications. The proposed framework focuses on dynamically optimizing resource allocation based on device states and network conditions, ensuring fairness, scalability, and energy efficiency across heterogeneous IoT environments [11].

1.3 Aim and Objectives

The aim of this research is to develop an adaptive and efficient FL-based resource management framework tailored for MEC environments. The proposed framework seeks to optimize energy consumption, minimize training delays, and enhance scalability while addressing the dynamic resource constraints, device heterogeneity, and massive connectivity demands of MTC applications in 5G networks. To achieve this aim, the following objectives are outlined:

1. To study the related works and identify the research gaps in resource management for MEC in 5G networks, focusing on adaptive FL learning approaches for MTC applications.

2. To develop dynamic resource management solutions for MEC in 5G networks, aimed at optimizing resource allocation such as energy consumption and reducing latency to support efficient solutions for many devices in MTC environment [6].

3. To integrate MEC and FL to design a framework that optimizes resource allocation in 5G networks for MTC, ensuring improved system performance, reduced costs, and maintaining privacy through collaborative device learning [10].

4. To validate and verify the developed framework using simulation tools and performance metrics like latency, energy efficiency, and resource utilization, ensuring the framework's effectiveness in realistic MTC environments.

1.4 Scope

This research focuses on the development of an adaptive FL based resource management framework tailored for MEC environments in 5G networks, specifically targeting MTC applications. It includes addressing resource allocation challenges, such as energy consumption, bandwidth utilization, and training latency [8]. Leveraging MEC to enable low-latency data processing and 5G networks leads to facilitating seamless connectivity and massive device support. In addition, developing and evaluating adaptive devices selection algorithms using DRL, specifically

Double Deep Q-Network (DDQN), to dynamically optimize resource management and ensure fairness across devices [12]. Using Modified National Institute of Standards and Technology dataset (MNIST) for training and validation, with simulations conducted in a controlled MEC-5G environment to assess system performance under diverse Network conditions.

Evaluating the proposed framework based on energy efficiency, latency, resource utilization, and model accuracy to ensure its practical applicability in real-world scenarios. It is limited to simulation-based evaluations and does not involve real-time hardware implementations or physical device testing. The scope is constrained by the availability of preprocessed MNIST dataset and computational resources for model training and evaluation. It ensures a focused approach toward addressing key challenges in resource management for MEC-enabled FL systems, providing a foundation for further research and potential real-world integration.

1.5 Methodology

This research aims to explore efficient resource management for IoT devices in constrained FL systems by employing a DRL approach integrated with MEC within a 5G network environment. The proposed methodology follows a systematic process that begins with dataset selection and preparation, followed by model development, 5G

integration, federated training, optimizing device resource usage and energy consumption, and finally model evaluation [10].

In this phase, publicly available datasets are curated MNIST dataset chosen as the primary dataset for this study. The MNIST dataset is preprocessed and organized to meet the requirements of FL experiments. The selection process prioritizes dataset diversity, scale, and suitability for training machine learning models under resource-constrained conditions. In this work, the dataset is obtained directly from the PyTorch torchvision library, which provides standardized and pre-structured versions of MNIST commonly used in deep learning research [13].

Next, the global learning model is designed and implemented using Python, where libraries such as PyTorch are utilized to construct a model capable of supporting distributed training across heterogeneous wireless devices. At this stage, the model architecture is constructed to support distributed learning and efficient aggregation of locally trained parameters. This stage focuses on building a robust base model that can be continuously updated through communication between the edge server and the devices [14]. Once the model is ready, it is integrated with IoT devices connected through 5G technology to enable bidirectional communication, ensuring seamless data exchange and low-latency synchronization. The 5G network acts as the foundation that supports the high-speed and low-delay communication required for FL [9].

After the integration phase, the global model is distributed to selected IoT devices, where each device trains the model locally using its data. Devices send their updated model parameters back to the edge server, which aggregates them to update the global model. This iterative process continues until the global model converges. It ensures data privacy and reduces the computational load on the central server. Following this, optimization mechanisms are applied to enhance device performance and energy efficiency. A DDQN agent integrated with MEC technology is employed to dynamically select the most suitable devices for each training round based on factors such as energy level, processing capacity, and available bandwidth. This optimization process helps balance resource utilization, minimize communication delays, and improve overall energy efficiency within the network.

The integration of DRL enables continuous adaptation to changing device and network conditions, enhancing the generalizability and reliability of the proposed framework in real-world scenarios[14]. Finally, the MNIST dataset is reused to test the model's predictions on IoT devices outputs [15]. This step facilitates the evaluation of energy consumption optimization, latency, and resource allocation across the IoT network. Model performance is evaluated using established metrics, including accuracy, to guarantee robust results under various operating conditions. Comprehensive performance analysis is also conducted using evaluation metrics such as confusion matrix to identify strengths and improvement areas. This systematic evaluation highlights the model's ability to

determine effective strategies that reduce power consumption and enhance network connectivity[11].

1.6 Thesis Outline

This research is organized into five chapters:

Chapter 1 – Introduction: This chapter provides a general overview of the research topic. It presents the problem statement, aim and objectives, scope of the study, research methodology, and the overall structure of the thesis.

Chapter 2 – Literature Review: This chapter offers a comprehensive background on the core technologies relevant to the study. It begins with an overview of 5G networks and examines Machine Type Communication (MTC), including its categories, architecture, and associated challenges. It then discusses the role of the Internet of Things (IoT) within 5G environments, followed by an exploration of resource management techniques in Mobile Edge Computing (MEC). Furthermore, it introduces the fundamental concepts of Machine Learning, including Federated Learning (FL) and Deep Reinforcement Learning (DRL), and concludes with a review of related works.

Chapter 3 – Methodology: This chapter outlines the proposed system model and methodology, describing the MEC server, IoT devices, and 5G communication setup. It explains the dataset selection and preparation

process, and presents the system design, including the CNN architecture, MDP formulation, DDQN-based device selection, and the Federated Averaging (FedAvg) algorithm.

Chapter 4 – Results and Discussion: This chapter discusses the experimental setup and the performance metrics used to evaluate the system, including accuracy, convergence behavior, latency, communication overhead, and energy consumption, and the evaluation of precision, recall, and F1-score. It presents a detailed analysis of the obtained results, including model evaluation, confusion matrix interpretation, comparison with baseline methods, and a discussion of the overall findings.

Chapter 5: This chapter summarizes the key outcomes of the research, highlighting improvements in resource allocation, energy efficiency, and communication performance. It also discusses the study's limitations and proposes future directions, such as real-world implementation, advanced optimization techniques, and enhanced system scalability.

CHAPTER TWO

LITERATURE REVIEW

2.1 Background

This chapter provides a comprehensive background on the key technologies that form the foundation of this research—5G networks, MTC, IoT, MEC, and FL. Together, these technologies enable efficient, low-latency, and intelligent wireless communication environments suitable for large-scale distributed systems.

The chapter begins with an overview of 5G networks, highlighting their role as an enabling infrastructure that supports massive connectivity, ultra-low latency, and energy-efficient communication. Through the integration of Human-Type Communication (HTC) and MTC, 5G networks have paved the way for intelligent ecosystems such as smart cities, industrial automation, and autonomous systems. Next, the chapter discusses MTC, emphasizing its two primary categories, mMTC and uMTC, which are essential for connecting billions of devices and supporting latency-critical services. The section also explores the synergy between MTC and IoT, where billions of interconnected smart devices exchange data seamlessly to enable real-time, data-driven decision-making.

The discussion then shifts toward MEC, which brings computational resources closer to end users to overcome the latency, bandwidth, and energy limitations of traditional cloud computing. MEC serves as a fundamental layer in supporting intelligent applications and improving resource utilization at the network edge. Finally, the chapter introduces

FL as a privacy-preserving distributed learning framework that operates efficiently within MEC environments. By training models locally and aggregating updates at the edge server, FL reduces communication overhead and enhances data confidentiality. The chapter concludes with a review of related studies that integrate these technologies, focusing on resource management, energy optimization, and DRL, to improve performance in 5G-enabled MEC systems.

2.2 5G Networks Overview

5G technology represents a major advancement in wireless communications, offering ultra-low latency, reliable connectivity, and scalable device support. As an evolving communication standard, 5G enhances overall network performance by enabling faster and more efficient data transmission, which accelerates data processing and reduces power consumption, marking a significant improvement over previous generations [16, 17, 18]. Since the introduction of cellular communication in the 1980s, new generations of networks have emerged approximately every decade. While the first four generations focused primarily on Human-Type Communications (HTC), the 5G, also known as 5G New Radio (NR), integrates both HTC and MTC. This allows devices to communicate wirelessly without human intervention, facilitating the creation of an interconnected network of devices known as the IoT. This

evolution supports diverse applications, ranging from simple, low-cost sensors to complex industrial automation systems.

Moreover, 5G technology defines three core application scenarios: enhanced Mobile Broadband (eMBB), massive Machine-Type Communications (mMTC), and ultra-Reliable Low-Latency Communications (uRLLC). These scenarios enable innovative solutions in areas such as smart cities and industrial automation. In addition, 5G reduces resource wastage by minimizing control signaling in devices, thereby optimizing the overall communication process [20, 19].

2.2.1 Machine Type Communication (MTC) in 5G Networks

2.2.1.1 Key Categories, Architecture, and Models

MTC, defined by 3GPP as a core 5G technology, enables seamless machine-to-machine connectivity within the IoT, allowing devices to exchange information and control data automatically without human intervention. This communication occurs automatically over both wired and wireless networks, allowing intelligent machines to generate, process, and exchange data independently, with minimal human involvement [21, 22]. MTC is expected to play a significant role in the context of 5G networks, driving substantial market growth. With an anticipated ability to support the connectivity of at least 100 billion devices, MTC in 5G promises a connection speed of up to 10 Gigabits per second (Gb/s) [22].

The METIS project, a key initiative in the development of 5G technologies, adopted MTC from 3GPP and classified it into two primary categories: massive (mMTC) and ultra-reliable (uMTC) [18]. mMTC focuses on providing scalable and efficient connectivity for a massive number of devices, particularly tens of billions of low-complexity, low-power machine-type devices [22, 18, 23]. mMTC provides a wide-area network architecture optimized for high-speed and cost-effective communication, ensuring improved network performance with lower latency and higher energy efficiency, making it ideal for IoT applications [24].

One of the key challenges in mMTC is accommodating a large volume of devices transmitting very short packets of data, a scenario not optimally supported by cellular systems designed for HTC. On the other hand, uMTC is designed to meet stringent requirements for availability, low latency, and high reliability, often required by critical applications such as Vehicle-to-X (V2X) communications and industrial control systems. These two categories address the core challenges of MTC: massive access and high reliability in communication networks [18]. The envisioned holistic architecture for MTC networks is designed to ensure efficient dynamic utilization of resources while offering distributed intelligence across multiple levels, such as cloud, edge, end devices, and applications. This architecture also features software-defined design, which is easy to upgrade and capable of meeting the needs of rapidly evolving 5G environments [19].

In 5G architecture, the Session Management Function (SMF) oversees MTC device sessions, including their creation, modification, and termination, while the Access and Mobility Management Function (AMF) handles access and mobility, such as registration and authentication. Furthermore, 5G introduces the separation of the user plane and control plane, offering enhanced flexibility for MTC service deployment and management [25]. Additionally, 5G supports various MTC modes, including Enhanced MTC (eMTC), URLLC, and mMTC, each optimized for different use cases, from massive connectivity to critical industrial communications [19, 25].

2.2.1.2 Challenges of Machine Type Communication (MTC)

MTC in 5G networks faces several significant challenges that must be addressed for efficient and reliable communication among billions of connected devices. One of the primary challenges is energy efficiency, as many MTC devices rely on limited battery power. This necessitates the development of energy-efficient communication protocols to prolong the lifespan of devices and reduce energy consumption through optimized designs [19]. Another challenge is achieving low latency, which is essential for time-sensitive applications such as autonomous driving and critical healthcare. However, the high density of devices and resource constraints in massive MTC deployments make meeting these stringent requirements difficult [26]. The proliferation of MTC devices (MTCDs) presents serious challenges to current cellular networks due to the massive

number of devices, particularly in ultra-dense scenarios such as stadiums, concerts, hotspots, and flash crowds [21].

Data security and privacy are also critical concerns, as the increasing number of connected devices heightens the risk of cyberattacks. The use of machine learning (ML) and artificial intelligence (AI) tools can provide intelligence-driven security capabilities for more accurate detection of malicious attacks. FL has emerged as a potential solution by processing data locally on devices, minimizing the risk of breaches during transmission and improving overall privacy [27].

Efficient utilization of computing and storage resources at edge devices is also crucial for supporting MTC applications, as it helps reduce latency and enhance overall network performance in large-scale IoT environments [7]. Overcoming these challenges is essential for the successful deployment of MTC in 5G networks and for supporting transformative applications like smart cities, industrial IoT, and autonomous systems.

2.2.2 Internet of Things (IoT) in 5G

The IoT is an emerging technology that aims to revolutionize global connectivity by seamlessly interlinking diverse physical objects. It relies on low-power devices capable of interacting with one another through the internet, enabling machine-to-machine (M2M) communication without human intervention. This interconnected ecosystem of smart devices -

such as wearables, sensors, appliances, and transportation systems have the potential to transform daily life and business operations. By digitalizing the physical world and converting it into a source of valuable data, IoT enables advanced analytics through cloud-based functionalities, unlocking critical insights for various sectors [28,17]. The full potential of IoT lies in connecting more smart devices, thereby fostering an environment that influences multiple aspects of everyday life while driving global economic growth through both massive and critical IoT deployments [28]. MTC serves as the backbone of IoT, ensuring efficient interaction between devices and supporting diverse applications and services [21].

The evolution of 5G networks is revolutionizing IoT applications by addressing the limitations of previous cellular standards [28]. The coexistence of 5G networks with IoT enables high data transfer rates, minimal energy consumption for resource-constrained devices, ultra-low latency (less than 2 milliseconds), enhanced coverage, scalability, versatility, and seamless integration of diverse technologies and platforms, making 5G a promising solution for the large-scale deployment of IoT devices across various applications [30]. As reported by IDC, 5G is expected to significantly boost IoT growth, with 70% of companies projected to spend $1.2 billion on connectivity management solutions. The future of IoT will demand advanced performance criteria, such as massive connectivity, security, ultra-low latency, high throughput, and reliability

requirements that 5G to meet through enhanced connectivity interfaces [29].

The increasing requirements of future IoT applications and the advancements in 5G wireless technology are driving the growth of 5G-enabled IoT systems. However, conventional network infrastructure is becoming outdated and is unable to support the new demands of IoT. As a result, network management complexity increases due to manual configuration processes and the limitations of traditional hardware-based systems. Future network infrastructures must evolve to enable these emerging technologies and meet the growing demands of next-generation networks [28]. In recent years, significant research has focused on 5G and its role in shaping future IoT, leading to the development of essential enabling technologies that provide the infrastructure and capabilities necessary to support these evolving demands [29].

The integration of 5G technology with IoT introduces transformative advancements to address the diverse requirements of resource-constrained IoT devices. These devices, often limited in computational power and energy capacity, demand several key features to support the growing needs of MTC [28, 29, 17]:

1. High Bandwidth: Enables faster and more reliable data transmission, supporting high-throughput applications like HD video streaming and real-time analytics.

2. Low Latency: Essential for applications such as tactile internet, augmented reality (AR), autonomous systems, and online gaming, requiring ultra-fast response times, ideally achieving delays as low as 1 millisecond for seamless operation.

3. Reliability and Resilience: 5G networks enhance IoT performance with improved coverage, robust error correction, and efficient interference management, crucial for mission-critical applications like remote infrastructure monitoring and medical devices.

4. Energy Efficiency: Vital for billions of low-power devices in remote locations, leveraging technologies such as low-power wake-up radios, energy harvesting, and optimized communication protocols to extend battery life significantly.

5. Security: A cornerstone for IoT applications involving sensitive data (e.g., mobile payments), requiring robust measures to ensure user privacy, data integrity, and location confidentiality, alongside forward and backward security for scalability [30].

6. Connection Density: Supports seamless operation of billions of devices with minimal signal interference, ensuring timely message delivery across diverse IoT applications, from smart cities to industrial automation.

7. Interoperability and Flexibility: Facilitates effective communication between heterogeneous IoT devices, particularly in

applications that utilize DL and edge computing to handle unstructured and noisy data [31].

8. Network Slicing: Creates dedicated virtual networks tailored to specific IoT applications, optimizing resources and enhancing performance and security. This feature provides customized connectivity for use cases such as industrial automation and healthcare.

9. AI Integration: Enhances 5G IoT capabilities by analyzing device-generated data for pattern recognition, prediction, and real-time decision-making, which is critical for applications like intelligent transportation and industrial control. Swarm intelligence enables device collaboration as interconnected nodes, improving efficiency in complex systems [17].

10. Quality of Service (QoS): Achieved through innovations like massive MIMO, beam forming, and network slicing, ensuring tailored resource allocation based on application requirements, resulting in enhanced connectivity, reduced latency, and optimized performance for large-scale IoT deployments [17].

The integration of edge computing, ML, and AI within the 5G-IoT framework introduces a transformative shift by enabling intelligent, real-time decision-making and adaptive networks that respond dynamically to changing environmental demands. By processing data closer to its source, this convergence enhances data analysis, drives automation, and optimizes

caching and data processing across various industries, making advancements in distributed systems and edge computing essential for unlocking the full potential of 5G-IoT networks [30].

2.3 Resource Management in Mobile Edge Computing (MEC)

With the rapid development of data science, data-intensive smart applications such as smart transportation, smart healthcare, AR/VR/MR, and real-time gaming are becoming increasingly popular. These applications often require massive data computations/processing and have strict low/ultra-low latency requirement. Nevertheless, deploying such applications on mobile devices is still a challenging problem, primarily due to their hardware limitations. Particularly: (i) most mobile devices do not have powerful central processing units (CPU) and thus cannot host computation-intensive applications; (ii) the battery capacity of mobile devices is greatly restricted by the small physical size, i.e., intensive computation drains the battery quickly, which limits computation-intensive application hosting; and (iii) smart applications require considerable memory space and may cause device memory shortages. In particular, ML and AI-based applications can easily occupy the entire memory, making the mobile devices significantly slow. The most obvious solution to address such issues is the use of remote computation that can provide elastic and on-demand resources [31].

To overcome these challenges, edge computing has been proposed to move the computing ability from the centralized cloud servers to edge nodes near the user end. Edge computing brings two major improvements to the existing cloud computing. The first one is that edge nodes can preprocess large amounts of data before transferring them to the central servers in the cloud. The other one is, the cloud resources are optimized by enabling edge nodes with computing ability [47].

Building upon this concept, MEC extends these advantages by integrating edge servers directly within mobile networks. MEC consists of an edge server that broadcasts global model, a radio frequency source (RF) that charges devices, and n devices that are equipped with CPU, energy unit and storage, and MTC technology with 5G network to improve the performance of DL applications by reducing the response time and lower the energy consumption. Edge server estimates MEC system energy consumption and transmission delay in next global model training progress that selecting a specific set of N devices that utilizing resource information [41].

Furthermore, the MEC system has the potential to provide computational resources to edge devices. The Edge device (Ed) does not have enough computational resources, so it offloads its tasks to nearby Mobile Edge Servers (MES), while sometimes offloading tasks to MES causes MES to overload. The overload state of MES is when the data occupies all the CPU cores and memory of MES, so MES rejects requests for data offloading from other Eds [46].

MEC effectively bridges the gap between cloud and mobile devices by decentralizing computation, minimizing latency, and optimizing resource usage. This paradigm not only enhances the efficiency of intelligent applications but also plays a key role in enabling future technologies such as federated FL and AI-driven IoT systems.

2.4 Machine Learning

2.4.1 Federated Learning

Models are trained using Machine Learning which is a branch of computer science that is concerned with building algorithms which relay on a collection of examples of some phenomenon. These examples can come from nature, handwritten by humans or generated by another algorithm. There are multiple ways to train a model in ML itself, from supervised learning to unsupervised and reinforcement learning technologies. Among these, FL, a branch of reinforcement learning, has emerged as a decentralized ML technique in which multiple devices collaboratively train a shared global model while keeping their data locally. This approach enhances privacy by ensuring that personal data remains on the device, allowing the global model to improve without exposing sensitive information [10].

FL is used as a branch of DL which is also a branch of ML. DL in short is a type of machine learning that uses artificial neural networks to learn

from data. Recent advancements in DL have largely been attributed to the increasing capacity of models, computational power, and the availability of large-scale datasets [33]. Such advancements highlight the importance of representation learning, where improved base models can enhance performance across various tasks such as image classification, object detection, and semantic segmentation.

However, traditional centralized learning models face challenges related to data privacy, communication overhead, and scalability [34]. FL addresses these issues by enabling distributed training, data remain local while model updates are aggregated on a central server. Several studies have demonstrated that FL can handle non-identically distributed (non-IID) data and reduce communication costs through iterative model averaging and structured update mechanisms [35, 36, 37].

In the context of networked systems, MEC provides a distributed computing environment that places computation and storage resources closer to end users. MEC enhances FL performance by minimizing latency and energy consumption compared to Mobile Cloud Computing (MCC), due to its proximity to devices and simpler configuration. Furthermore, edge and fog computing architectures have been effective in addressing latency and bandwidth limitations of cloud-based solutions, supporting real-time and low-latency applications [6, 38].

2.4.2 Deep Reinforcement Learning (DRL)

DRL is a subfield of machine learning that combines the principles of reinforcement learning and deep learning, enabling agents to make decisions and learn optimal behaviors in complex, high-dimensional environments. The integration of reinforcement learning into decentralized and edge-based systems further amplifies their potential by allowing intelligent and adaptive operations close to the data source.

Among the foundational methods in DRL, Q-learning is a model-free reinforcement learning algorithm that helps an agent learn the optimal action-selection policy by iteratively updating Q-values, which represent the expected rewards of actions in specific states. However, traditional Q-learning suffers from a tendency to overestimate action values, which can hinder stable learning and performance. To address this issue, Double Q-learning was proposed to decouple the processes of action selection and evaluation, improving both the stability and accuracy of value estimation in complex environments [12,40]. Building upon this concept, the DDQN algorithm extends Double Q-learning by integrating deep neural networks to approximate the Q-value function more efficiently. DDQN employs two neural networks, an online network and a target network, to separate the processes of action selection and action evaluation, thereby reducing overestimation and ensuring more stable convergence during training[61].

Moreover, the combination of reinforcement learning with deep neural networks enables agents to process high-dimensional sensory inputs and achieve human-level performance across diverse tasks. These

advancements lay the groundwork for combining hierarchical sensory processing with reinforcement learning to create intelligent, adaptive agents [40]. Emerging technologies have also DRL based incentive mechanisms to optimize edge learning. Such approaches can balance latency and payment in edge-based FL by automatically adapting pricing strategies, ensuring efficient resource utilization while addressing network dynamics and privacy concerns. This highlights the intersection of distributed machine learning and real-world constraints, offering practical solutions for scalable and efficient deployment [39].

2.5 Related Works

In [30] Shams Forruque Ahmed and colleagues reviewed the challenges and opportunities associated with integrating 5G with IoT to create a secure, efficient, and scalable ecosystem. Their work highlights how 5G enhances IoT by providing ultra-low latency, high-speed connectivity, and improved bandwidth, enabling IoT applications such as smart cities, healthcare, and industrial automation. However, the integration also introduces challenges such as security vulnerabilities, privacy concerns, and resource management issues.

The authors emphasize the importance of resource-efficient technologies such as edge computing, machine learning, and blockchain to address the scalability and energy efficiency needs of 5G-enabled IoT. They also propose robust security mechanisms, such as advanced encryption and

authentication protocols, to protect data and prevent cyberattacks. Furthermore, the study discusses the role of standardization in ensuring interoperability across heterogeneous IoT devices and 5G networks. Simulation results suggest that these approaches significantly enhance the efficiency, scalability, and security of 5G-enabled IoT systems while addressing the unique challenges posed by diverse application requirements.

Also, in [31] He Li and colleagues introduced an innovative approach to integrating deep learning with edge computing for IoT applications. Their study addresses challenges such as the need for efficient data processing in IoT environments and preserving user privacy during data transfer. They formulated an elastic model that adapts to varying deep learning frameworks, enabling partial offloading of learning layers to edge nodes. This reduces the size of intermediate data sent to cloud servers, thereby optimizing network performance and safeguarding privacy.

The researchers also developed offline and online scheduling algorithms to maximize the number of deep learning tasks handled by edge nodes while ensuring quality of service (QoS). Extensive experiments with ten different convolutional neural network (CNN) models demonstrated that their method significantly outperforms traditional solutions, increasing the number of tasks deployed on edge servers and reducing network traffic. This study highlights the potential of deep learning and edge computing to improve IoT services by enhancing efficiency, scalability, and privacy,

with plans for future real-world deployment to further validate their approach.

[43] Waheed Ur Rehman and his team proposed a resource allocation scheme tailored for 5G MTC networks to address challenges such as variable quality of service (QoS) requirements, high device density, and resource constraints. The study introduces a dynamic priority-based resource allocation algorithm that considers multiple radio access technologies (RATs), prioritizing tasks based on QoS parameters like signal-to-noise ratio (SNR), queuing delay, and the number of devices awaiting transmission.

The scheme operates in two main phases: medium access and resource allocation. In the medium access phase, priority is assigned to devices using parameters like intra-frame delay, inter-frame delay, and the number of pending transmissions within a class of devices. Devices with higher priority gain faster access to the medium, reducing contention and improving fairness. During the resource allocation phase, the algorithm assigns cellular resources to devices based on calculated priorities, ensuring that devices with stringent QoS requirements or higher delays are served first. simulation results demonstrate that the proposed scheme significantly enhances success probabilities by up to 30% and reduces outage probabilities by up to 20% compared to SNR-based schemes. The study highlights the importance of combining prioritized contention mechanisms with dynamic resource allocation to meet the demands of

massive MTC in 5G networks, ensuring efficient and fair resource distribution in high-density, low-latency environments.

In [45] Silvana Trindade and her team analysed the challenges and proposed solutions for managing resources at the network edge to facilitate FL. The study emphasizes the importance of resource discovery, deployment, load balancing, migration, and energy efficiency in ensuring effective FL operations within edge computing environments. FL preserves data privacy by keeping raw data on client devices while aggregating only locally trained models into a global model.

The research highlights the distinct challenges of edge environments, including limited bandwidth, energy constraints, and uneven data distribution. It presents strategies for optimizing resource allocation, balancing workloads, and minimizing migration delays. Furthermore, the authors stress the need for energy-efficient approaches to enable edge nodes to handle FL tasks without excessive power consumption. The simulation results demonstrate that these resource management techniques significantly improve FL performance at the edge, enabling privacy-preserving, low-latency machine learning applications in real-world scenarios.

Also in [63] Silvana Trindade, Luiz F. Bittencourt, and Nelson L.S. da Fonseca explored the challenges and solutions for resource management in federated learning at the network edge. The study highlights the importance of managing limited computational and communication

resources on edge devices to enable efficient and privacy-preserving training of machine learning models. Key issues such as resource discovery, deployment, load balancing, migration, and energy efficiency are discussed. The authors emphasize that addressing these challenges is essential to support scalable and low-latency federated learning applications in real-world edge computing environments.

In [44] Shiqiang Wang and his team investigated resource management in FL within resource-constrained MEC environments. Their work focuses on optimizing the balance between local updates and global aggregation to address computational, communication, and energy limitations. By analysing gradient-descent-based FL algorithms, they proposed a control algorithm that dynamically adjusts the frequency of global aggregations in real-time to minimize learning loss under fixed resource budgets.

The study highlights the interplay between resource usage and training accuracy, considering non-IID data distributions and system dynamics. The proposed algorithm achieves near-optimal performance by efficiently utilizing computation and communication resources, adapting to various machine learning models and data distributions. Experimental results in both hardware prototypes and simulations demonstrated significant improvements in FL performance, making it suitable for MEC systems in real-world applications.

Additionally, in [20] Jie Li and colleagues proposed a federated reinforcement learning-based mechanism for adaptive task offloading in

MEC to address challenges such as time-varying transmission channels, privacy concerns, and computational cost. The study introduces a DDQN based task offloading algorithm combined with FL to improve Quality of Service (QoS) by optimizing task scheduling decisions based on real-time channel conditions.

The proposed framework enables distributed training of decision models on devices, protecting user privacy while enhancing the global model's generality. To mitigate delays caused by FL's communication overhead, a univariate outlier detection algorithm was designed to optimize transmission delay by screening out users with low contribution rates during global iterations. This ensures better utilization of FL's advantages while improving task processing efficiency.

Simulation results demonstrate that the method significantly reduces energy consumption and task delay while maintaining high QoS under dynamic channel conditions. The integration of FL allows devices with limited private data to leverage collaborative training, making the solution particularly effective for users with sparse local data or privacy concerns. By combining MEC, FL, and reinforcement learning, this approach enhances task offloading performance and user experience in 5G environments.

Similarly, in [42] Takayuki Nishio and Ryo Yonetani proposed FedCS, a novel protocol designed to improve the efficiency of FL in MEC frameworks with heterogeneous clients. FL enables privacy-preserving

model training by keeping user data on local devices while iteratively aggregating updates from selected clients. However, challenges such as varying computational resources and wireless conditions among clients can delay training processes. FedCS addresses this by actively managing client resources and implementing a client selection mechanism that maximizes the number of updates aggregated within a set time frame, optimizing training efficiency.

The protocol sets deadlines for clients to complete downloading, updating, and uploading tasks, ensuring timely server aggregation and efficient model training. Experiments on large-scale image classification tasks in a simulated MEC environment demonstrated that FedCS significantly reduces training time compared to standard FL protocols. The method consistently performs well regardless of data distribution (IID or non-IID) and varying resource constraints. By incorporating more clients into the training process and addressing heterogeneous resource conditions, FedCS enhances the practicality and scalability of FL in real-world MEC applications.

And in [36] Hao Wang and colleagues introduced FAVOR, a novel framework designed to enhance the efficiency of FL on non-independent and identically distributed (non-IID) data. The framework utilizes reinforcement learning, specifically a Deep Q-Network (DQN) enhanced with Double DQN, to intelligently select a subset of devices for participation in each training round. This approach counters the bias

introduced by non-IID data, accelerates convergence, and reduces communication overhead.

Through mathematical analysis and empirical experiments, the researchers identified a relationship between the local data distribution and the model weights trained on that data, enabling device selection without direct access to raw data. FAVOR actively selects devices that contribute most effectively to the global model's improvement while minimizing the number of communication rounds. By addressing the challenges of non-IID data and communication constraints, FAVOR significantly improves the scalability and performance of FL systems.

Finally, in [41] H. Zhang and colleagues conducted a study on adaptive client selection in resource-constrained FL systems using DRL. The proposed approach integrates FL with MEC to address challenges such as limited bandwidth, energy constraints, and dynamic environments. Their work introduced a novel mechanism for adaptive client selection in MEC, modeled as a Markov Decision Process (MDP) and solved using a DDQN algorithm. The study demonstrated that the DDQN-based approach reduces energy consumption by up to 50% and decreases training delays by 20.7% compared to traditional static methods. It also showed scalability across datasets like MNIST and Fashion-MNIST, achieving consistent improvements in resource utilization and faster global model convergence with fewer communication rounds. By optimizing client selection and addressing non-IID data distributions, this method enhances

efficiency, scalability, and privacy-preserving capabilities in MEC systems.

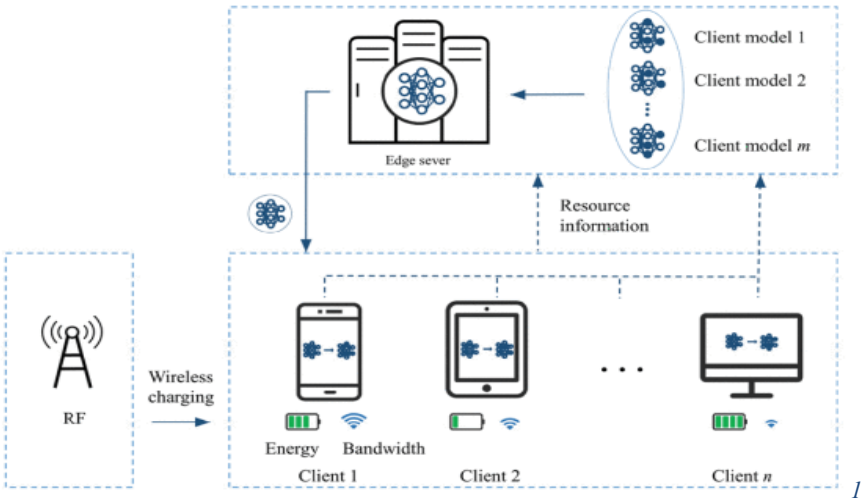

Figure 2.1: An adaptive client selection algorithm in resource constrained FL systems [41]

2.6 Summary

This chapter has presented a review of the advancements in 5G networks and their crucial role in enabling seamless wireless communication for modern smart systems. It explored the integration of MTC and HTC, with a particular focus on (mMTC) and (uMTC). The discussion highlighted their importance in achieving high connectivity, reliability, and scalability. Key challenges associated with MTC, such as energy efficiency, low latency, and security, were analyzed, along with emerging solutions like FL and intelligent resource management. Additionally, the role of MEC in optimizing computational efficiency and reducing network congestion was examined, particularly in IoT

applications. Furthermore, the chapter explored the integration of FL within MEC environments, emphasizing its potential to enhance privacy and computational performance.

Recent research has demonstrated the effectiveness of advanced learning algorithms such as the DDQN, which improves decision-making stability and accelerates resource optimization in federated and edge learning scenarios. Finally, the reviewed studies demonstrated how the synergy between 5G, MEC, FL, and DRL contributes to enhancing the overall performance, resource management efficiency, scalability, and intelligence of next-generation IoT systems.

CHAPTER THREE

METHODOLOGY

3.1 Introduction

This chapter presents the project methodology adopted to investigate the implementation of MEC within the framework of FL [48] for efficient resource management of IoT devices in 5G environments [4]. The proposed system model integrates several key components, including the MEC server, intelligent IoT edge devices, the 5G communication infrastructure, and advanced deep learning techniques such as CNNs and reinforcement learning algorithms like DDQN [12]. The primary objective of this research is to optimize computational and energy resource utilization in MEC environments while minimizing latency and enhancing the global model performance in FL.

This is achieved through intelligent coordination of the training process among edge devices, energy consumption management, and device selection optimization using DDQN-based strategies. To support communication efficiency, the 5G network plays a crucial role in ensuring high-speed data transmission and low-latency interaction between edge devices and the MEC server [49, 11]. For simulation purposes, the study employs the MNIST dataset [13] distributed across IoT devices to mimic real-world federated training scenarios. Furthermore, Markov Decision Processes (MDPs) are used to model decision-making in the system, while the Federated Averaging (FedAvg) algorithm is applied to aggregate local models and enhance the global model's accuracy [34].

3.2 System Model

3.2.1 System Model Overview

In traditional systems, cloud computing (CC) was used to process data for devices with limited computational capabilities, such as IoT devices that suffer from low processing power or limited energy. These devices would send their data to cloud servers, Where the processing is done and then the results are returned to the devices. However, this model introduced significant latency due to the distance between cloud data centers and edge devices, negatively impacting applications requiring real-time responsiveness, such as (MTC).

To reduce this latency and improve data processing efficiency, MEC emerged. In Figure 3.1, the MEC server is positioned closer to edge devices, allowing it to process data locally and significantly reduce delays [51]. When integrated with 5G networks, MEC further enhances system performance by providing ultra-low latency, high-speed data transfer, and reliable connectivity between IoT devices and edge servers [50]. The MEC is utilized to efficiently manage the FL process by coordinating training among edge devices, receiving their resource information, and selecting training participants based on their available resources. This approach helps reduce network congestion, enhance model performance, and lower energy consumption in participating devices.

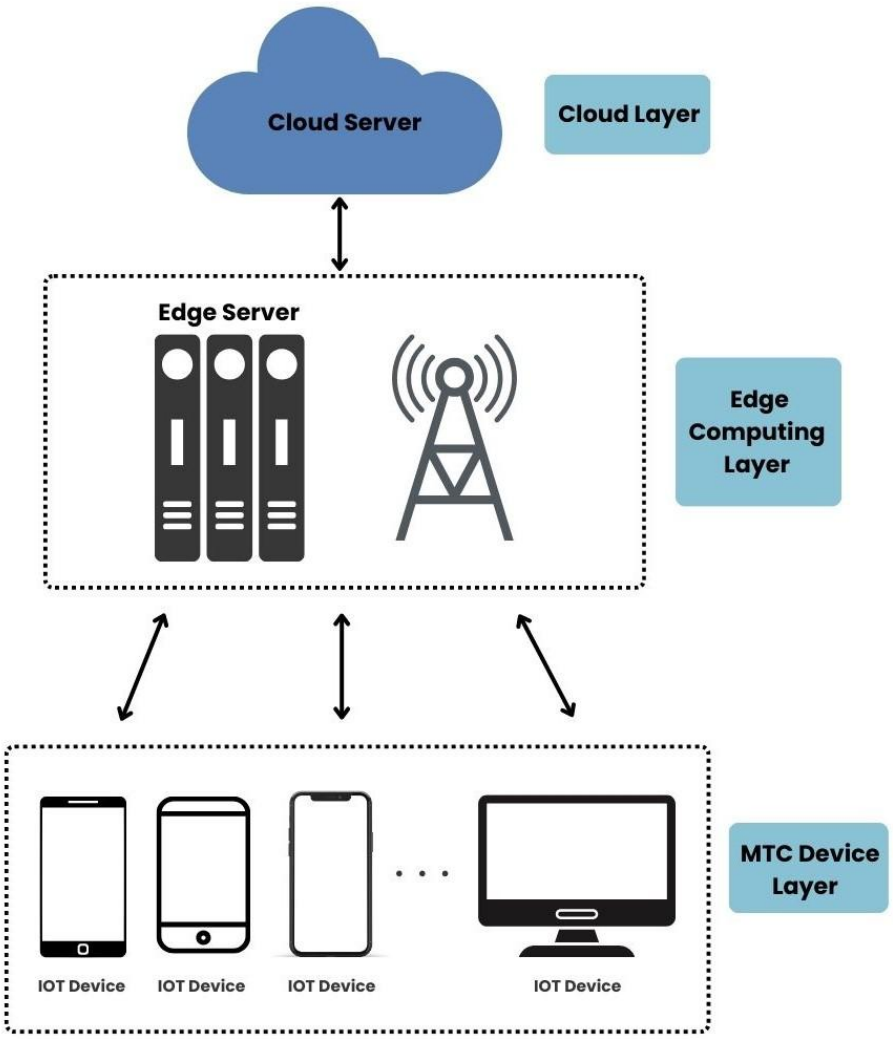

Figure 3.1: Mobile Edge Computing Architecture

3.2.2 Mobile Edge Computing (MEC) Server

The MEC server is the central component of the system, responsible for managing the FL process and coordinating training among edge devices to ensure efficient performance as illustrated in Figure 3.2. It begins by receiving resource information from n device, including CPU-cycle frequency f_k , energy units e_k, wireless bandwidth r_k, and wireless charging capability. Using this information, the MEC server estimates system energy consumption and transmission delay in the next global model training round. It then selects a specific subset $N' = \{1,2,\ldots,m\}$ of device based on their available resources [41].

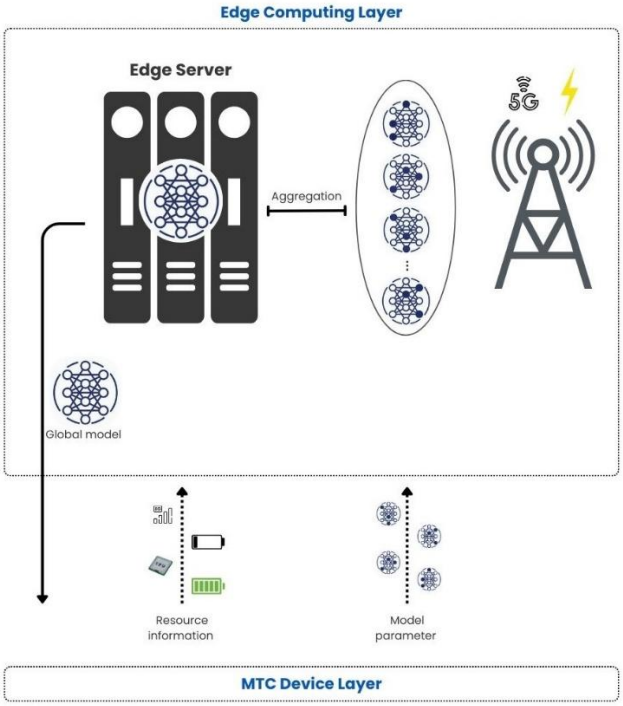

Figure 3.2: Edge Computing Layer

One of the key functions of MEC is federated model management. The MEC server hosts the global model and transmits it to selected edge devices for local training using their private data. The model in this system is based on CNNs due to their efficiency in processing visual data, such as the MNIST dataset used in this project [41]. Each round server decides the ML model training task, i.e., the number of local training iterations ε and the corresponding training data requirements μ. Each selected device performs local training, requiring computational resources proportional to its dataset size. The local computing time for device k is given by:

$$T_k^{local} = \frac{\mu G}{f_k}, \qquad where\ k \in N' \tag{3.1}$$

where G is the number of CPU cycles needed per bit of data [41].

Once training is completed, devices upload their local model parameters to the MEC server via wireless communication channel. The transmission time for uploading local model parameters is given by:

$$T_k^{trans} = \frac{D}{r_k}, \qquad where\ k \in N' \tag{3.2}$$

where D is the global model size [41].

To ensure efficient learning, the MEC server estimates the total system delay and ensures that the total time for broadcasting global model, updating local model, and local model uploading does not exceed the maximum time limit L_{max} [41], given by:

$$L_{max} = max\{T_k^{local} + T_k^{trans}\}, \qquad where\ k \in N' \qquad (3.3)$$

The MEC server aggregates the received updated local models' parameters using the FedAvg algorithm; to improve the global model progressively over multiple training rounds and evaluates the new global model performance with certain validation data set. To optimize device selection, the MEC server employs a DDQN algorithm, which considers the resources of each device to balance training efficiency while minimizing energy depletion and performance degradation. The device selection mechanism can be modeled as an MDP, enabling dynamic adaptation to resource variations without requiring prior environmental knowledge [41].

Throughout the training process, the MEC server continually updates and redistributes the improved global model among eligible devices. This iterative approach enhances model accuracy while ensuring optimal energy consumption for each device. Ultimately, it achieves a balance between learning efficiency, resource utilization, and system responsiveness within the 5G-enabled edge computing environment.

3.2.3 Smart Edge Devices (IoT Devices)

Edge devices shown in Figure 3.3 play a fundamental role in executing federated training, utilizing their local resources for learning without transferring raw data to the MEC server. These devices begin by reporting their available resources including processor frequency, remaining energy, bandwidth, and wireless charging levels to the MEC server. This process allows devices to indicate their capability to participate in training without compromising their primary functions or depleting their energy reserves [41].

The system is based on a MTC environment, which includes smart IoT devices connected to the 5G network, ensuring high-speed connectivity for fast and efficient responses while exchanging unified models. As outlined in Figure 3.3 once the MEC server selects eligible devices, each device receives the global model over 5G and starts local training using the MNIST dataset, which consists of handwritten digit images ranging from 0 to 9 [13]. Devices leverage their internal processors to perform computations, consuming processing cycles that affect both energy consumption and execution time. Each training iteration enables devices to refine the model using their private data, allowing adaptation to their specific non-IID data distributions.

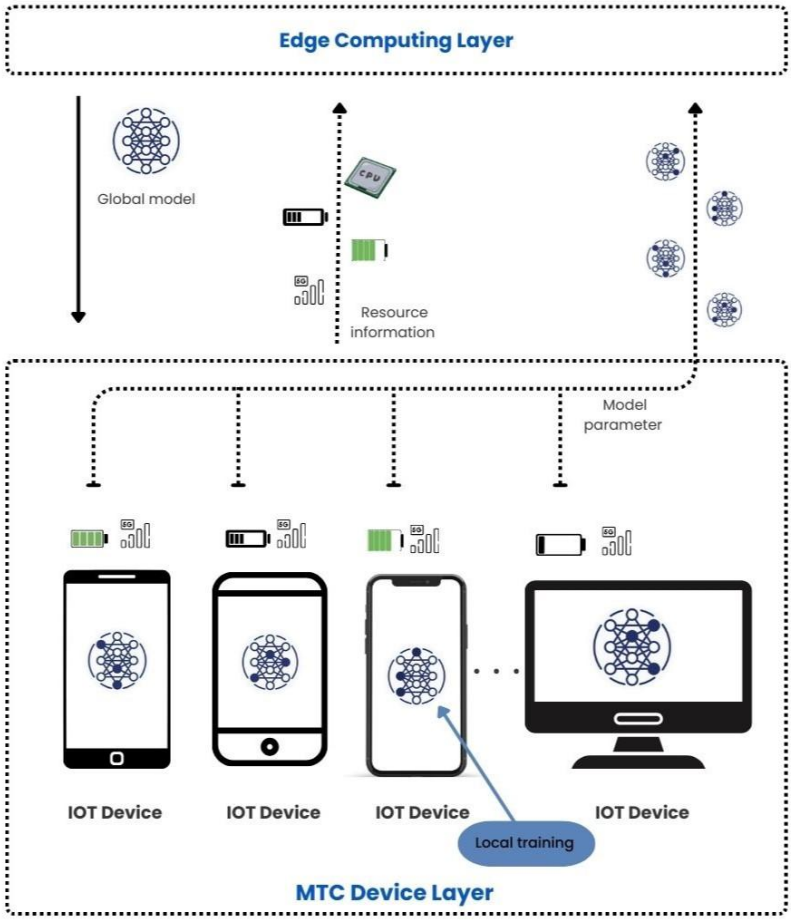

Figure 3.3: MTC Device Layer

Each participating device consumes energy units during training and transmission. However, since transmission power is relatively low compared to training, energy consumption primarily depends on the local training process [41]. The energy unit consumption per training iteration for device k is given by:

$$B_k = f_k^2 \tau \mu G \tag{3.4}$$

Where f follows a uniform distribution and τ is the effective switched capacitance that depends on the device's chip architecture [41].

At each iteration, device k has an available energy level e_k and can be wirelessly charged by receiving C_k energy units from the environment via radio frequency source (RF). Charging energy units C_k is assumed to follow the Poisson distribution, i.e.,

$$\mathrm{P}(C_k = \mathrm{t}) = \mathrm{e}^{\omega} \frac{\omega^t}{t!} \tag{3.5}$$

where ω is the average energy arrival rate [41].

The transition of the energy state is given by:

$$e_k{}' = max\{e_k - B_k + C_k, 0\} \tag{3.6}$$

This equation ensures that a device never has negative energy and accounts for both consumption during training and replenishment from wireless charging [41].

Once training is complete, devices compute their updated model weights and transmit them over the network, contributing to global model improvement without exposing their raw data. Since transmission

consumes some device resources, balancing energy consumption and communication efficiency is critical to ensure sustained participation in future training rounds. The effectiveness of IoT devices in this process depends on optimized device selection and load distribution, ensuring that FL progresses efficiently without degrading device performance.

3.2.4 5G Network

5G technology plays a crucial role in FL within edge computing environments by offering ultra-fast connectivity, massive network capacity, and ultra-low latency. These features facilitate real-time data exchange between IoT devices and MEC servers, allowing for efficient global model updates, local weight transmission, and device resource monitoring. As a result, 5G significantly reduces transmission delays, ensuring frequent and accurate model aggregation, which enhances convergence speed and overall learning performance [52, 17].

In addition to speed, 5G networks provide wide coverage and high bandwidth, enabling a large number of IoT devices to participate in FL without overloading the network or compromising communication quality [17]. In this system, IoT devices first report their available resources, including processing power, energy levels, and wireless bandwidth, to the MEC server through 5G communication links, ensuring reliable transmission of status information. The MEC server then selects the most suitable devices for participation. The scalability of 5G allows the

deployment of FL across diverse environments, ranging from dense urban areas with thousands of connected devices to remote and rural regions where stable and high-speed connectivity was previously a challenge [53,54].

Furthermore, the system integrates 5G-enabled wireless power transfer (WPT) through radio frequency (RF) energy harvesting, addressing the energy constraints of IoT devices involved in FL. Since training deep learning models in an FL framework requires substantial computational power, ensuring a stable energy supply for participating devices is crucial. By utilizing RF waves from 5G infrastructure, IoT devices can wirelessly harvest energy, reducing dependency on manual battery recharging [55, 56]. This approach enhances system sustainability, allowing uninterrupted training cycles while extending device longevity especially in large-scale FL deployments where frequent human intervention is impractical. Accordingly, by integrating 5G, FL, and IoT technologies, the proposed system establishes a highly adaptive and resource-efficient edge computing framework. The ultra-low latency and high reliability of 5G facilitate real-time coordination between MEC and IoT devices, optimizing communication efficiency and energy consumption.

3.2.5 Energy Consumption Management

Energy consumption is a major challenge in implementing FL on edge devices, as training and transmission processes require significant

energy resources, potentially affecting device efficiency and sustainability in the learning process. To address this challenge, the system employs the DDQN algorithm to select suitable devices based on their available resources. The MEC analyzes resource data sent from edge devices to assess their training efficiency. Devices that maintain a balance between performance and energy efficiency are selected to ensure that the training process does not negatively impact on the device or deplete its resources entirely. Additionally, the selection policy is dynamically updated based on past experiences through DRL, allowing the system to adapt to continuous changes in available resources [41].

Furthermore, with each training round, each device gradually consumes energy, which may affect its ability to continue participating. To address this issue, the system minimizes energy consumption by selecting the most efficient devices through DDQN, ensuring that training tasks are assigned to devices with sufficient resources to maintain continuous participation without performance degradation or complete energy depletion.

In addition, wireless charging mechanisms are utilized to compensate for energy consumption during training, thereby enhancing device sustainability while executing complex computational tasks. Once training is completed, devices compute updated model weights and transmit them to the MEC. Since transmission consumes device resources, balancing energy consumption and communication efficiency is crucial to ensure continued participation in future training rounds. This is achieved by optimizing device selection to distribute the workload evenly among

devices, ensuring maximum efficiency in FL without negatively impacting edge device performance [41].

3.3 Dataset

3.3.1 Dataset Selection

In this project, managing resources in MEC environment requires a well-structured approach to data selection and preprocessing, as these stages are essential for optimizing performance and ensuring accurate modeling of resource management strategies. To achieve this, the MNIST dataset is utilized, obtained through Torchvision library in PyTorch which provides a pre-structured and normal implementation widely recognized for benchmarking machine learning algorithms. Selecting an appropriate dataset is crucial for effectively modeling energy consumption, bandwidth utilization, and 5G network performance in IoT-based MEC environments [13,64].

The MNIST dataset is particularly suitable for this purpose due to its computational efficiency and compatibility with distributed training across resource-constrained devices. It consists of grayscale images of handwritten digits (0–9), making it lightweight yet sufficiently diverse to evaluate learning performance under limited computational and energy resources. The structured format of MNIST enables efficient preprocessing and straightforward integration into FL experiments. Its

simplicity facilitates faster convergence and reduced computational overhead, both of which are essential for evaluating the proposed DDQN-based FL resource management framework. Furthermore, MNIST's balanced distribution supports stable model aggregation across federated rounds, effectively simulating real-world IoT environments where the MEC server dynamically allocates resources among distributed devices to minimize energy consumption and delay [57].

3.3.2 MNIST Dataset Overview

The MNIST dataset is a well-known benchmark in the field of machine learning and pattern recognition, primarily used for image classification tasks. It comprises 70,000 grayscale images of handwritten digits ranging from 0 to 9, with each image having a resolution of 28×28 pixels. The dataset is divided into 60,000 training samples and 10,000 testing samples, allowing for consistent evaluation of model performance across different experiments [65]. Each image in the dataset represents a single handwritten digit, centered and size-normalized to maintain uniformity as shown in Figure 3.4.

Figure 3.4: MNIST Dataset

3.3.3 Data Preparation

Before initiating the training process, the MNIST dataset undergoes a series of preprocessing steps to ensure consistency, efficiency, and compatibility with the FL framework implemented in this project. These preprocessing operations are crucial for optimizing model performance and reducing computational overhead during distributed training on IoT devices. Each MNIST image is converted into a tensor representation using the Torchvision.transforms module in PyTorch, which allows the model to process image data numerically. The pixel intensity values, originally ranging from 0 to 255, are normalized to a scale between 0 and 1 to improve gradient stability and accelerate convergence during the

learning phase. This normalization also ensures uniform input distribution across devices participating in the FL process.

Additionally, the dataset is partitioned into local subsets to simulate a federated environment, where each IoT device maintains its own private data segment without sharing raw images with the MEC server. The preprocessed data are then loaded using PyTorch DataLoader functions, which efficiently manage batch loading and shuffling to enhance training performance and balance computational workloads among devices. Through these preprocessing steps, the system achieves optimized data representation.

3.3.4 Data Preprocessing for DRL and FL

To effectively utilize the processed data within DRL and FL frameworks, the data must be appropriately structured. Key steps include transforming raw features into suitable state representations for reinforcement learning, such as encoding the current resource states of devices as numerical vectors. Following this, the data is partitioned and distributed across IoT devices in a manner that preserves data privacy, thereby simulating real-world federated environments where devices perform local model training and only share updates with the global model. This structured preprocessing ensures that the simulation data closely reflects practical deployment scenarios, enabling our DDQN-based model to efficiently learn optimal resource management strategies. The MNIST dataset supports these steps by allowing normalization of

image pixel values to enhance training stability and convergence. Finally, model performance is evaluated and visualized using outputs such as the confusion matrix, which provides insights into classification accuracy across different digit classes. [66].

3.4 System Design

3.4.1 Convolutional Neural Network (CNN)

The key components of the system design include a trainable CNN model. A CNN is employed to process and analyse large-scale IoT data efficiently. The CNN model being trained consists of multiple layers. The edge server initializes a global model that consists of two 32 × 32 convolution layers followed by 2 × 2 max pooling, and two 64 × 64 convolution layers followed by 2 × 2 max pooling [41]. The first convolutional layer extracts low-level features such as edges, textures, and simple patterns. It applies a set of filters (kernels) to the input image, generating feature maps that highlight crucial regions of interest. The second convolutional layer refines the extracted features by detecting more complex patterns. This layer is followed by a max pooling operation, which reduces the dimensionality of feature maps while preserving essential information. Pooling enhances computational efficiency and mitigates overfitting by focusing on dominant features [58].

3.4.2 Markov Decision Processes (MDPs)

MDP offer a robust mathematical framework for decision-making in stochastic environments, such as MEC-based FL systems. In this context, an MDP consists of several key components. The state space S defines all possible states the system can be in, with each state representing various device resource conditions, such as CPU frequency f_k, energy consumption e_k, and wireless bandwidth r_k. The overall state space of the system is expressed as:

$$S = \prod_{k=1}^{n} S_k \tag{3.7}$$

where $\prod$ is the Cartesian product, and S_k is the state of device k. The state space of device k is expressed as:

$$S_k = \{f_k, e_k, r_k; f_k \leq F, e_k \leq E, r_k \leq R\} \tag{3.8}$$

where F, E, R are the limitation of CPU-cycle frequency, energy units and wireless bandwidth, respectively [41].

The action space A defines the possible actions taken by the edge server, which include selecting devices to participate in FL. Each device either participates or does not in the current round, represented as:

$$A = \prod_{k=1}^{n} A_k \tag{3.9}$$

where $A_k = \{0\} \cup \{1\}$ that the action state of device k. A_k=0 means that device k does not participate in updating global model this round, where as A_k=1 means that device k take part in this round to train a local model [41].

The transition function governs how the system moves from one state to another based on the selected actions, considering factors such as network latency, energy consumption, and device availability. The reward function $R(s, a)$ measures the system's performance based on these actions. It is defined as:

$$R(s,a) = \alpha_n \frac{m}{n} - \alpha_e \frac{E}{E_{max}} - \alpha_l \frac{L}{L_{max}} \quad (3.10)$$

where α_n is Scale factor for device number , α_e is Scale factor for energy units , α_l Scale factor for delay [41]. E is the total energy unit's consumption by MEC system in each iteration

$$E = \sum_{k=1}^{m} B_k, \qquad where\ k \in N' \quad (3.11)$$

E_{max} are the total energy units of system that is formulated as:

$$E_{max} = \sum_{k=1}^{n} E, \qquad where\ k\ \in\ N' \quad (3.12)$$

L is the maximum delay for devices that taking part in this iteration:

$$L = max\left\{\left(\frac{\mu G}{f_k} + \frac{D}{r_k}\right)\right\} \ where\ k \in N' \tag{3.13}$$

To solve the MDP, two popular algorithms are applied: Policy Iteration and Value Iteration. Policy Iteration alternates between policy evaluation and policy improvement steps until convergence to an optimal policy [59]. Value Iteration uses the Bellman Equation to iteratively update value functions, ensuring that each state's value accounts for future rewards. According to Datacamp, "Value iteration" is a method that uses the Bellman Equation to update value functions iteratively. The goal is to find the optimal value for each state so that the agent can determine the best actions. In value iteration, the agent updates each state's value based on possible future states and their rewards" (Datacamp, 2024) [60] .In this context, The Value Iteration algorithm is employed to iteratively update state values, which enhances the performance of the DDQN algorithm in selecting devices more effectively.

3.4.3 Double Deep Q-Network (DDQN)

To enhance FL efficiency, the DDQN algorithm, depicted in Figure 3.5, is utilized to achieve adaptive device selection, which represents an advanced practical application of the Bellman equation. DDQN ensures

stable learning and prevents overestimation. It is derived from the Bellman equation [61], which is formulated as follows:

$$\mathrm{Q}(s,a;\theta) = R(s,a) + \gamma Q(s',\ argmax_{a'}\mathrm{Q}(s',a';\theta);\theta') \quad (3.14)$$

DDQN consists of an action-value online network Y and an action-value target network Y'. The online neural network (NN) updates its weights θ, so it selects the best action based on experience replay (state s, action a, reward $R(s,a)$, and next state s'), and the target neural network (NN) evaluates the action selected by the online network, and resets its weights $\theta' = \theta$ periodically using the gradient descent algorithm. DDQN selects an action for edge server based on online NN and evaluates the action with target NN that prevents the policy overoptimistic problem [41].

The DDQN training process begins by initializing the replay memory, action-value function, and target function with random weights. For each training episode, the system observes the current state and selects an action using an epsilon-greedy policy. It then executes the action and stores the transition (state, action, reward, next state) in memory. A mini batch of past experiences is sampled, and the target value is computed. The online network is updated using gradient descent, which is formulated as follows:

$$\theta' \leftarrow \theta\ -\ \beta\ \nabla_{\theta} L\theta \quad (3.15)$$

where β is learning rate, controlling the step size of weight updates [62] and $\nabla_\theta L\theta$ is the gradient of the loos function with respect to θ which defined as:

$$L\theta = \mathbb{E}[y - Q(s, a; \theta)]^2 \tag{3.16}$$

where target value y is defined as:

$$y = R(s, a) + \gamma Q'(s', argmax_{a' \in A} Q(s', a'; \theta); \theta') \tag{3.17}$$

Periodically, the target network is updated to align with the online network [41].

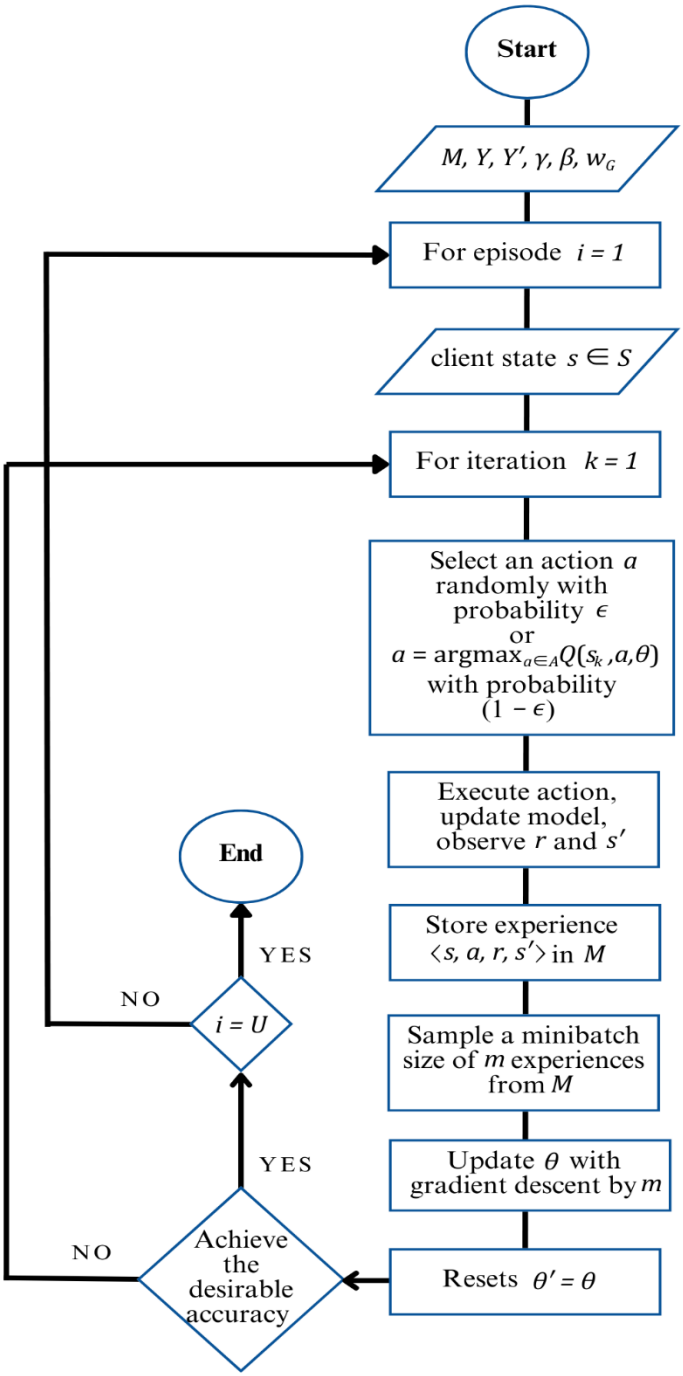

Figure 3.5: DDQN Based - Device Selection Algorithm.

3.4.4 Federated Averaging (FedAvg)

FedAvg is an essential algorithm for aggregating local models into a global model at the edge server in FL systems which is formulated as follows:

$$w_G^{i+1} = \frac{1}{\sum_{k=1}^{m} |\mu_k|} \sum_{k=1}^{m} |\mu_k| w_k^i \tag{3.18}$$

Where w_G^{i+1} denotes the global model parameters, and w_k^i is local model parameters of device k in communication round i. Additionally, $|\mu_k|$ denotes the data size of k, while m denotes the selected devices. According to devices willing and server aggregation efficiency, the corresponding training data requirement is defined as μ [41]. Therefore, the aggregation function in server if the data size is equal for all devices is

$$w_{\mathrm{G}}^{\mathrm{i+1}} = \frac{1}{\mathrm{m}} \sum_{\mathrm{k=1}}^{\mathrm{m}} w_k^i \tag{3.19}$$

FL training task is iterated till the global loss function converges or achieved desirable accuracy Ω [41]. The algorithm begins with the initialization of a global model w_0^0 on the edge server, which is then distributed to selected IoT devices, then w_G^0 denotes the global model parameters in first iteration. Each device performs local training using its own dataset and computes weight updates. These updated weights are then

aggregated based on the number of data points at each device, resulting in an improved global model [34].

For ease of reference, our commonly used symbols are summarized in Table 3.1.

Table 3.1: Summary of Commonly Used Symbols

Symbol	Quantity
n	Number of devices participating in updating model
f_k	CPU-cycle frequencies for local training
e_k	Energy level of device k
r_k	Wireless bandwidth of device k
μ	Training data requirements
D	Global model size
G	Number of CPU cycles required to process one-bit local data
ω	Average energy arrival rate
τ	Effective switched capacitance
ε	Number of local training iterations
B_k	Energy consumed by device k
C_k	Amount of harvested energy of device k
Ω	Desirable accuracy
L_{max}	Maximum time limit
γ	Discount factor
w_G^i	Global model parameters in iteration i

w_k^i	Local model parameters of device k in iteration i
α_n	Scale factor for device number
α_e	Scale factor for energy units
α_l	Scale factor for delay
β	Learning rate for model optimization
l_r	Learning rate for the local model

3.5 Summary

This chapter outlines the methodology focusing on optimizing computational and energy resources, reducing latency, and enhancing the performance of the global FL model. The system model incorporated key components, including the MEC server, IoT edge devices, 5G networks, and advanced machine learning techniques such as CNNs and DDQN. The MNIST dataset was used to simulate real-world FL. The MEC server managed energy consumption coordinated training among IoT devices, and optimized device selection. IoT devices performed local training using their private data and reported resource availability to the MEC server, ensuring energy-efficient participation.

The 5G network enabled ultra-low latency and high-speed communication, facilitating efficient data exchange between devices and the MEC server. Additionally, WPT through 5G infrastructure addressed energy constraints, enhancing device sustainability. The MNIST dataset

was preprocessed through normalization, to improve model performance. The system design incorporated CNNs for data processing, MDPs for decision-making, DDQN for adaptive device selection, and FedAvg for aggregating local models into a global model. Together, these components optimized resource management, reduced latency, and improved the overall efficiency of FL in 5G-enabled computing environments.

CHAPTER FOUR

RESULTS AND DECISIONS

4.1 Introduction

The following sections present an analysis of the experimental results obtained from simulating the proposed adaptive FL framework within a 5G-enabled MEC environment. The system was designed to jointly optimize accuracy, latency, and energy consumption while addressing the dense connectivity requirements characteristic of MTC. All experiments were conducted using the MNIST dataset, selected for its suitability in examining the behavior of the adaptive FL under constrained edge device resources. The simulation models the interaction between multiple IoT devices and a central MEC server operating under dynamic network conditions, including variations in bandwidth availability, latency levels, and device energy states. DDQN agent is integrated into the system to intelligently guide device selection and optimize resource allocation during training.

This chapter highlights how the model performs in terms of accuracy, convergence, energy efficiency, and latency, while also discussing the significance of the findings in comparison with baseline approaches and discussion of Findings.

4.2 General Setup

Our MEC system consists of one edge server and four IoT devices, providing an efficient framework for FL in a 5G environment.

Specifically, the system uses two primary models: a DDQN model with a CNN for adaptive device selection, and a global CNN model for aggregating local model updates.

The DDQN network at the edge server is based on a CNN with three convolutional layers (64 filters, 3×3 kernel, padding=1), each followed by a 2×2 max-pooling layer, and two fully connected layers (from 64 × 4 × 4 to 64 units, then to the output). This architecture is optimized for MNIST images, which have a single grayscale channel.

The global CNN model differs slightly to support FL. It has two convolutional layers with 32 filters (3×3 kernel, padding=1) followed by 2×2 max pooling, then two convolutional layers with 64 filters (3×3 kernel, padding=1) followed by 2×2 max pooling, and finally two fully connected layers with dropout. These design choices allow efficient processing and stable training performance for this dataset.

In our experiments, we ran 1000 episodes, selecting only one device for training in each episode. Continuous iterations led the model to achieve an accuracy of 88% by iteration 160. Each device provides resource information, such as wireless channel and energy states, while retaining private data of size μ = 1 MB.

The DDQN model optimizes device selection based on resource states, with f_k, e_k, and r_k sampled from uniform distributions $U[0,1]$, $U[2,5]$, and $U[0,2]$ respectively, ensuring resource-efficient participation in training rounds [41]. All IoT devices connect to a 5G network initialized

with a base bandwidth of 1000 Mbps, a base latency of 0.005 seconds, and a sub-6 GHz spectrum, with an interference factor of 0.01, a fading factor of 0.97, and a packet loss rate of 0.0003. Upon connection, the DDQN model selects suitable devices for training, and the 5G network dynamically reallocates 80% of the bandwidth to selected devices, enabling faster and more efficient training. All parameters are presented in Table 4.1.

Table 4.1: Simulation Parameters

Parameters	Value
n	4
μ	1 MB
D	20 Mbit
G	7000
ω	1
τ	10^{-28}
E	5
F	1GHz
R	2Mbps
L_{max}	500 s
γ	.99
ϵ - greedy	$0.1 \rightarrow 0$

α_n	3
α_e	2
α_l	2
β	.001
l_r	.0001

4.3 Performance Metrics

4.3.1 Accuracy and Convergence

Accuracy was our main concern in this project alongside both latency and energy consumption. For this reason, multiple processes were established to ensure the desired accuracy is met which was estimated to be %88. For the iterations we calculated the accuracy for each device with ease. The accuracy section of the process also returned to a single scalar value representing accuracy, which is straightforward to interpret and compare across experiments.

The resulting model achieved the desired accuracy by consistently selecting resource allocation strategies that minimize energy consumption and bandwidth usage while maintaining high task success rates. During our training for the dataset, as you can see from the following Figure 4.1, we got an accuracy of 0.88 derived, updates toward a globally stable and accurate policy.

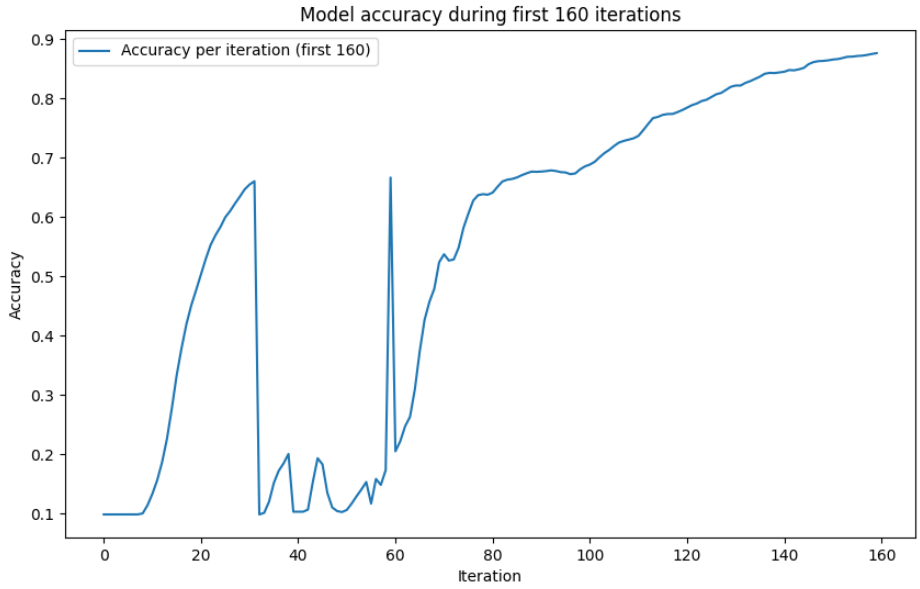

Figure 4.1: Accuracy and Convergence of the FL Model in Resource Allocation

4.3.2 Energy Consumption

Energy efficiency is essential for the operation of edge devices in MEC systems, particularly in MTC environments where devices often operate under strict energy constraints. To evaluate the performance of our proposed FL framework within the MEC-enabled 5G setting, we recorded energy consumption per episode throughout the training process using the MNIST dataset. Figure 4.2 illustrates the energy consumption pattern across 1000 training episodes. In the early training phase, we observe noticeable fluctuations in energy usage starting from 4 energy unit. These variations stem from the random initialization of device energy levels and the exploratory behavior of our DDQN agent, which initially tests diverse

device-selection actions. Such fluctuations reflect the dynamic scheduling process and the heterogeneous energy states of the participating MEC server.

As training advances, our DDQN agent progressively identifies more energy-efficient scheduling strategies. This results in a clear stabilization trend, where the average energy consumption converges toward a consistent range. In our experiments, the energy values eventually stabilize near 0.5 energy unit, as observed at the end of the plot. This convergence indicates that the agent has learned an effective policy that balances computation and communication overhead while ensuring sustained device participation.

To model realistic energy behavior, we initialize each edge device with a limited energy budget and allow it to consume energy during both local computation and data transmission. The computation cost depends on the CPU frequency and model size, whereas the transmission cost is driven by available 5G bandwidth and latency. Additionally, we incorporate a stochastic energy replenishment mechanism based on Poisson charging events to periodically restore part of the devices' energy reserves This mechanism mitigates energy depletion, reduces dropout risk, and extends the operational lifespan of the system.

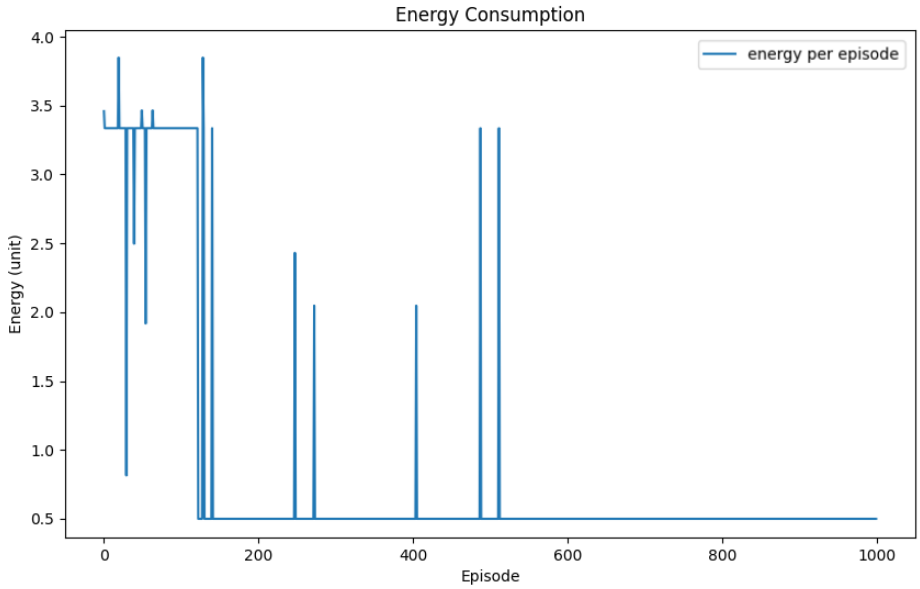

Figure 4.2: Energy Consumption and Stabilization of Edge Devices during FL Training

4.3.3 Latency and Communication Overhead

Latency was evaluated as the sum of the local computation delay and the transmission delay for all selected devices in each training round using the MNIST dataset. This metric is particularly important for MTC applications that require ultra-reliable low-latency communication in 5G enabled environments. Figure 4.3 illustrates the latency trend across 1000 training episodes. At the beginning of training, we observed significant fluctuations in latency, ranging from relatively low values to extremely large spikes. These variations occur due to the exploratory behavior of our DDQN agent, which initially performs diverse device-selection actions

while system parameters such as device load, channel quality, and processing frequency are randomly initialized.

As training progresses, our DDQN policy becomes more stable, leading to fewer abrupt peaks in latency. A clear convergence pattern emerges, where the delay settles into a narrower and more consistent range compared to earlier episodes. In the final stages, the latency stabilizes around 0.4 seconds, as shown by the plot. It is important to note that we explicitly imposed an upper bound of 0.005 seconds on 5G transmission latency. As a result, the 5G network contributed almost no meaningful delay, since all values were capped at this threshold.

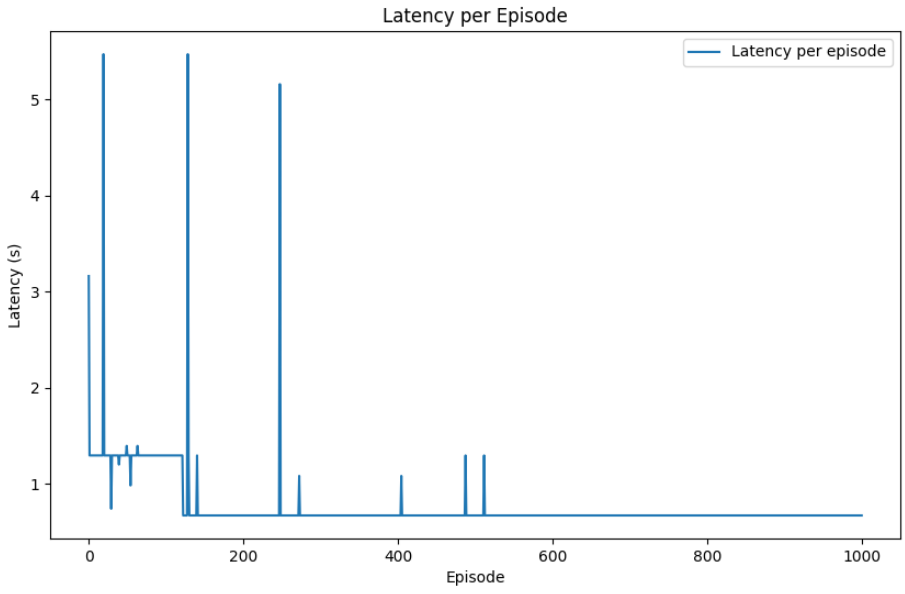

Figure 4.3: End-to-End Latency and Communication Overhead in the MEC-Enabled FL System

4.4 Analysis of Results

4.4.1 Model Performance Evaluation and Testing

The evaluation function measures the classification accuracy of the global model on the given test dataset, as it was used to evaluate the model. We switched the model to evaluation mode to deactivate training-specific layers like dropout and batch normalization. Afterwards, we performed all computations on the CPU to ensure compatibility. The process involves iterating through the test data without gradient calculations for efficiency purposes, generating predictions, and comparing them with the true labels. Then we took the number of correctly classified samples accumulated, divided by the total number of samples to obtain the accuracy score. We provided an unbiased estimate of model performance on the chosen dataset; by using a special built-in function to avoid unnecessary computational overhead. This evaluation method was effective for quantifying the predictive capability of the global model as we used the resulting accuracy score to track improvements.

As shown in Figure 4.4, the evolution of precision, recall, and F1-score over the first 160 training iterations, illustrating the model's learning behavior over time. Precision reflects how reliable the model's positive predictions are, meaning it measures the proportion of correctly predicted positive cases out of all cases predicted as positive. Recall represents the model's ability to identify all relevant instances, measuring how many actual positive cases are successfully detected. The F1-score combines

both precision and recall into a single metric by taking their harmonic mean, providing a balanced view of performance, especially when there is a trade-off between the two.

Across the first 160 iterations, all three metrics generally show an upward trend, which indicates progressive improvement as the model learns from the data. Early iterations are characterized by low and unstable values, suggesting that the model initially struggles to make accurate and complete predictions. As training continues, both precision and recall increase and become more stable, implying that the model is not only making fewer incorrect positive predictions but is also capturing a larger portion of the true positive cases. The F1-score follows a similar pattern, rising as the balance between precision and recall improves. Since the analysis is limited to the first 160 iterations, the plot specifically highlights the early and mid-stage learning dynamics rather than final convergence, showing how performance steadily matures within this initial training window.

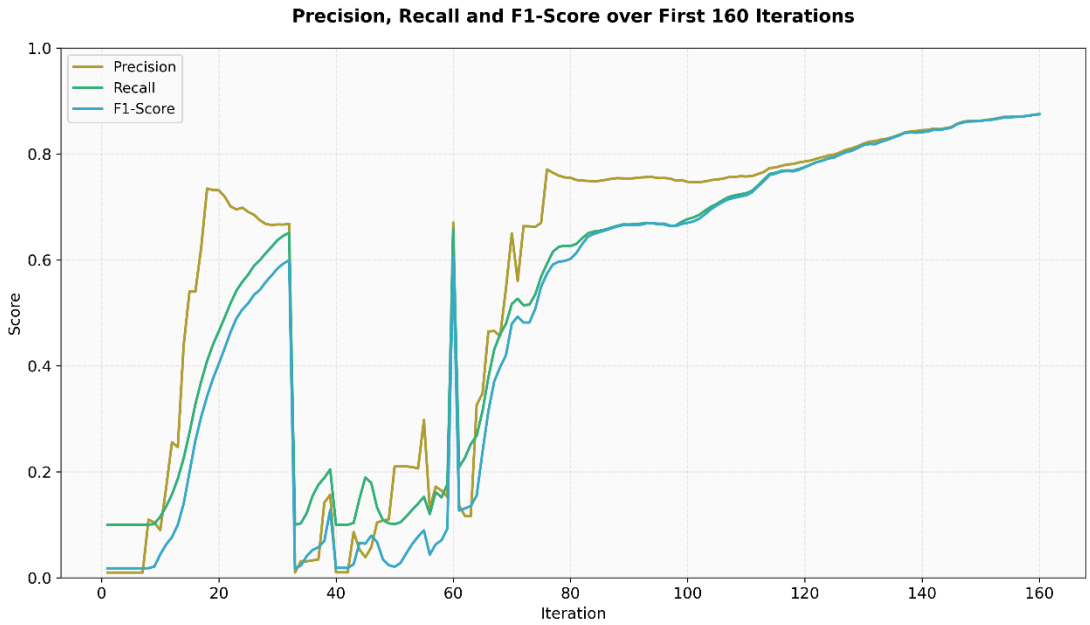

Figure 4.4: Evolution of Precision, Recall, and F1-score over the first 160 training iterations

Figure 4.5 presents the Receiver Operating Characteristic (ROC) curves for each class, illustrating the model's classification performance across different decision thresholds. The ROC curve plots the true positive rate against the false positive rate, showing how well the model distinguishes a given class from the others as the threshold varies. A curve that rises sharply toward the top-left corner indicates strong discriminative ability, meaning the model achieves high detection of true positives while keeping false positives low.

The diagonal dashed line represents random guessing, where the true positive rate increases proportionally with the false positive rate. All class-specific ROC curves lie well above this baseline, which indicates that the model performs significantly better than chance for every class. The Area

Under the Curve (AUC) values, reported for each class, summarize this performance into a single number. AUC values close to 1 reflect excellent separability, showing that the model can reliably rank true instances of a class higher than non-instances. Overall, the consistently high AUC scores across classes demonstrate that the model generalizes well in a multi-class setting, maintaining strong discrimination capability for all categories rather than favoring only a subset. This confirms that the classifier is robust and effective at separating classes across a wide range of thresholds.

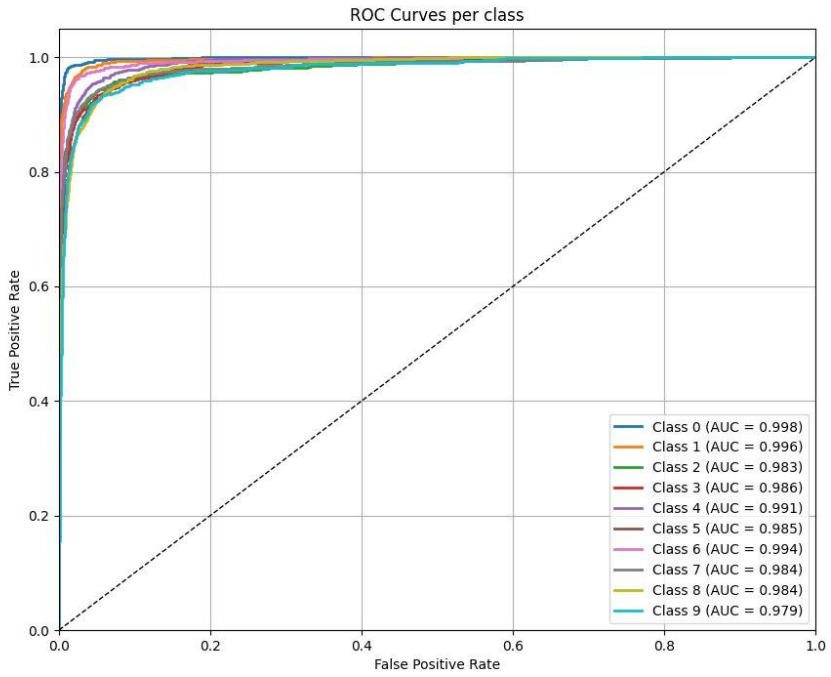

Figure 4.5: Receiver Operating Characteristic (ROC) curves and corresponding AUC values for each class

These results provide a natural link to the Confusion Matrix Analysis, which offers a more detailed, class-by-class view of the model's predictions. While accuracy, precision, recall, F1-score, and ROC/AUC curves summarize overall and threshold-dependent performance, the confusion matrix explicitly shows how many instances of each class are correctly or incorrectly predicted, revealing specific patterns of misclassification and guiding further improvements.

4.4.2 Confusion Matrix Analysis

The confusion matrix in Figure 4.6 shows the true labels versus the predicted labels for each class that evaluates the performance of a multi class classification model across 10 classes (labeled 0–9). In the main diagonal, top-left to bottom-right, the values represent the number of correct predictions for each class. For example, in the first cell in the main diagonal, class 0 was correctly predicted 948 times. The second cell shows that class 1 has the highest correct predictions with 1060 times. While class 5 has 724 correct predictions which is the lowest compared to others.

The remaining cells indicate misclassifications—instances where the model predicted the wrong class. For example, class 4 is often confused with class 9 (70 times) which is the highest misclassification. Class 8 as well is frequently misclassified as class 5 (60 times). Class 7 is misclassified as class 9 (50 times). Class 9 is misclassified as 7 (47 times) and as class 4 (46 times). We can notice that the model performs very well in classes 0, 1, 2, 6, and 7, as they have the highest diagonal counts with

relatively fewer errors. In contrast, classes 4, 5 and 8 show more confusion with other classes, suggesting these categories are harder for the model to distinguish. The darkest the square indicates how much the strength of the performance is. For example, on the diagonal for class 1 (with 1060) shows the strongest performance compared to other performances. Misclassifications are spread out but more concentrated between certain pairs (e.g., class 4 with class 9 (70 times), class 7 with class 9 (50 times), and class 8 with class 5 (60 times)).

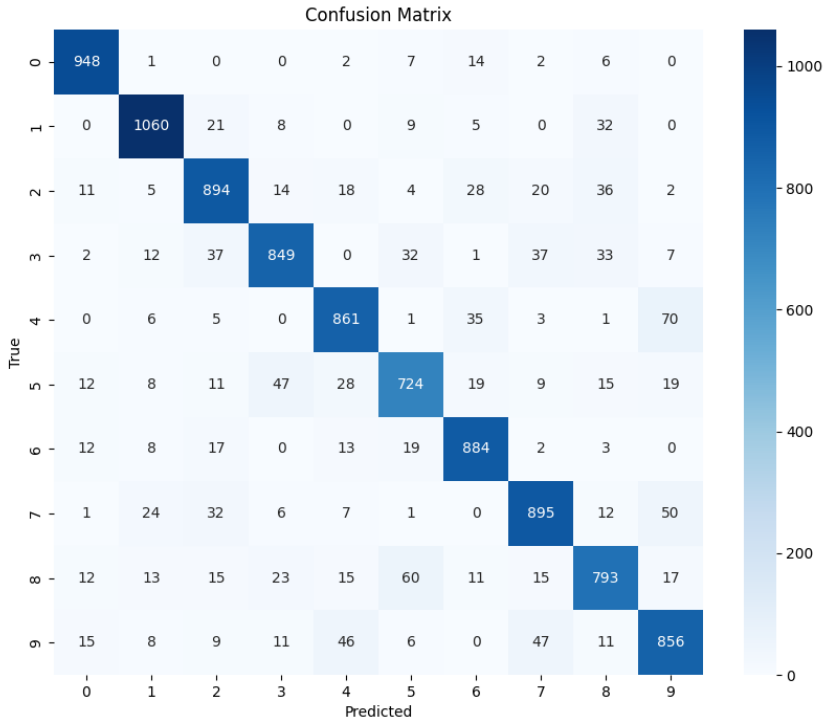

Figure 4.6: Confusion Matrix for Model Evaluation

4.4.3 Comparison with Baseline Approaches

Our proposed framework introduces several enhancements that extend beyond the methodology presented in the baseline study [Adaptive Client Selection in Resource-Constrained FL Systems] [41]. While the baseline focuses primarily on the DDQN mechanism under simplified assumptions, our approach integrates 5G connectivity, MEC computation, and MTC communication principles into a unified architecture. This integration enables more efficient scheduling, faster device coordination, and enhanced support for large-scale deployments. The adoption of 5G improves model exchange reliability through high bandwidth and ultralow latency, while MEC reduces communication distance and local processing overhead, enabling MTC driven scalability that supports the effective participation of devices. These additions make our framework more reflective of realistic wireless environments and better suited for deployment in emerging intelligent network infrastructures.

When comparing accuracy performance, the baseline DDQN model achieves 99% accuracy on MNIST within more than 200 communication rounds, benefiting from idealized conditions. In contrast, our system converges to approximately 88% accuracy obtained over 160 iterations closely approaching the baseline performance and demonstrating strong robustness within a real-world setup. Although lower than the baseline, this result remains competitive considering the complexity and variability of the operational environment we simulate. Moreover, while the baseline smooth consumption values primarily due to its simplified and controlled

assumptions, our model demonstrates greater energy stability in practical settings, exhibiting more consistent consumption patterns despite operating under more realistic and dynamic conditions.

However, the latency observed in our system is lower and can be attributed to the integration of 5G technology, despite occasional peaks caused by realistic bandwidth variations and dynamic device availability. On the other hand, the baseline achieves consistently minimal delay but it does not account for real world MTC behavior or MEC related scheduling overhead.

The relatively lower performance observed in some cases can be attributed to several practical factors. First, the datasets used in our experiments were not perfectly clean, containing noisy and slightly distorted samples, unlike the carefully preprocessed data typically used in controlled academic studies. Second, the computational devices used in the simulation for training had limited processing power and memory, which may have caused increased energy consumption. Despite this limitation, the FL process still achieves reasonable accuracy, highlighting the tradeoff between the number of selected devices, energy consumption, and communication delay.

Overall, the performance of our framework remains stable, efficient, and suitable for practical deployment. The system consistently maintains acceptable energy consumption and manageable low latency while sustaining a learning trajectory that approaches baseline performance

under far more restrictive and realistic conditions. This demonstrates that incorporating 5G, MEC, and MTC into the DDQN based device selection process produces a robust and scalable solution capable of supporting FL in real world intelligent wireless networks.

4.4.4 Discussion of Findings

The experimental results demonstrate that our proposed MEC-enabled FL system effectively balances accuracy, latency, and energy consumption. Specifically, integrating DDQN-based device selection with the FL model improved performance over the baseline. For the MNIST dataset, the framework reached approximately 88% accuracy after 160 training rounds, achieving stable and gradual convergence. This indicates the system's ability to adapt to varying data characteristics while implementing resource allocation strategies that reduce energy usage and bandwidth consumption.

The confusion matrix confirms this accuracy, showing strong classification performance in classes 0, 1, 2, 6, and 7. Misclassifications were most frequent in classes 4, 5, and 8 due to visual similarity, which slightly increased energy consumption and latency, illustrating the inherent trade-offs between these metrics.

Initially, latency exhibited noticeable fluctuations due to the DDQN agent's exploratory behavior and the random initialization of device parameters. As training progressed, the latency gradually stabilized and ultimately converged to approximately 0.4 seconds, indicating improved

decision-making and more consistent communication performance. with 5G transmission latency capped at 0.005 seconds.

Similarly, energy consumption showed significant variability at the beginning of the simulation, where the initial results indicated that it was approximately 4 units due to random device energy states and exploratory scheduling actions. Over time, it steadily decreased and eventually converged to around 0.5 energy units, demonstrating the effectiveness of the proposed scheduling strategy in optimizing resource utilization while maintaining stable device participation.

Overall, these findings demonstrate that the proposed framework successfully manages the trade-offs between accuracy, latency, and energy efficiency. The DDQN-based device selection mechanism plays a central role in this balance, ensuring resource-efficient yet high-performance training. These results confirm the framework's robustness and reliability, highlighting its potential for real-world deployment in 5G MEC environments supporting FL applications [41].

4.5 Summary

This chapter presented a comprehensive analysis of the results the adaptive FL framework deployed within a 5G-enabled MEC environment. The system was designed to jointly optimize accuracy, latency, and energy consumption while addressing the dense connectivity requirements of

MTC. Experimental results demonstrate that the model successfully converges to an accuracy of approximately 88% after 160 training rounds, showing stable and gradual improvement despite dynamic bandwidth, latency variations, and heterogeneous device energy states. The confusion matrix indicates strong classification performance for most classes, with misclassifications concentrated among visually similar digits, highlighting inherent dataset challenges rather than model deficiencies. Energy consumption initially fluctuated due to exploration behavior and random device conditions but gradually stabilized around 0.5 energy units as the DDQN agent learned efficient scheduling strategies. Similarly, latency showed early spikes but ultimately converged to approximately 0.4 seconds, supported by the ultra-low 5G transmission delay capped at 0.005 seconds.

The results collectively show that the proposed framework effectively balances the trade-offs among accuracy, latency, and energy efficiency. Compared with the baseline study, the system demonstrates strong robustness despite operating under more realistic and restrictive conditions. The integration of 5G, MEC, and DDQN-based scheduling provides clear advantages for scalable FL deployments in intelligent wireless networks.

CHAPTER FIVE

CONCOLUTION

5.1 Conclusion

This research introduced an adaptive federated learning framework designed to enhance resource management within a 5G-enabled Mobile Edge Computing environment, supporting the demanding requirements of Machine Type Communication applications. By combining federated learning with a DDQN-based reinforcement learning mechanism, the system successfully addressed challenges related to device heterogeneity, limited energy resources, and variable wireless conditions. The framework coordinated local training across IoT devices using a CNN based global model, while the MEC server managed device selection and resource allocation dynamically.

Using the MNIST dataset and realistic simulation conditions, the system demonstrated stable convergence behavior, achieving an accuracy of approximately 88% after 160 iterations. Both latency and energy consumption showed gradual stabilization as the DDQN agent learned more efficient scheduling strategies. These results indicate that the integration of MEC, 5G connectivity, and intelligent reinforcement learning can significantly enhance distributed learning performance while maintaining energy efficiency and low communication delay. The overall findings confirm that the proposed framework is well suited for scalable, adaptive, and privacy-preserving learning in next-generation IoT networks.

5.2 Limitations

Although the proposed system achieved promising outcomes, several constraints were observed. The evaluation was fully simulation based, which limits the ability to capture real world behaviors such as unpredictable wireless interference, hardware limitations, and device failures. The experimental setup used only four IoT devices, which is far smaller than typical MTC scenarios involving large-scale device deployments. The framework relied on the MNIST dataset, a simple dataset that does not fully represent the complexity of real IoT data, which is often non-IID complicated or high-dimensional.

In addition, the wireless and energy models were simplified using fixed statistical distributions, whereas real devices experience irregular energy patterns and diverse channel conditions.

5.3 Future Work

Future improvements could enhance realism, scalability, and adaptability. Deploying the framework on a fully integrated 5G-IoT platform would provide deeper insight into real-time constraints and system responsiveness. Expanding the number of participating devices would make the system more representative of dense MTC environments and enable exploration of more advanced reinforcement learning

strategies capable of handling larger action spaces. Adopting more complex datasets or real sensor-generated data would also offer a better understanding of model behavior under realistic conditions.

In addition, supporting multi-device selection in each training iteration would accelerate convergence and better reflect practical FL deployments. The framework could further be extended to incorporate multi-tier edge hierarchies, enabling cooperation between nano-edge devices, MEC servers, and cloud layers to optimize both energy consumption and computational load.

Future systems may also benefit from adaptive compression mechanisms that dynamically adjust model updates based on bandwidth availability and device capabilities, thereby reducing communication overhead and efficiently managing resources. Moreover, evaluating and adapting the proposed framework within emerging network generations such as 5G Advanced and 6G could provide valuable insights into leveraging ultrahigh capacity, sub-millisecond latency, and AI-native network features. These advancements have the potential to further enhance the performance, scalability, and real-time responsiveness of adaptive federated learning in large scale IoT and MTC environments.

REFRENCES AND APPENDIX

References

[1] M. Shafi, A. F. Molisch, P. J. Smith, T. Haustein, P. Zhu, P. De Silva, F. Tufvesson, A. Benjebbour, and G. Wunder, “5G: A tutorial overview of standards, trials, challenges, deployment, and practice,” IEEE Journal on Selected Areas in Communications, vol. 35, no. 6, 2017, doi: 10.1109/JSAC.2017.2692307.

[2] Gupta and R. K. Jha, “A survey of 5G network: Architecture and emerging technologies,” IEEE Access, vol. 3, 2015.

[3] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, “A survey on mobile edge computing: The communication perspective,” IEEE Communications Surveys & Tutorials, vol. 19, no. 4, pp. 2322–2358, 2017.

[4] M. Chiang and T. Zhang, “Fog and IoT: An overview of research opportunities,” IEEE Internet of Things Journal, vol. 3, no. 6, 2016.

[5] X. Zhang and S. Debroy, “Resource management in mobile edge computing: A comprehensive survey,” ACM Computing Surveys, vol. 55, no. 13s, Art. no. 291, pp. 1–37, 2023, doi: 10.1145/3589639.

[6] R. Yu and P. Li, “Toward resource-efficient federated learning in mobile edge computing,” IEEE Network, 2021, doi: 10.1109/MNET.011.2000295.

[7] A. E. Ahmed et al., "Adaptive Federated Learning Based Resource Management in Mobile Edge Computing for Machine Type Communication in 5G Network," 2026 IEEE 5th International Maghreb Meeting of the Conference on Sciences and Techniques of Automatic Control and Computer Engineering (MI-STA), Sebha, Libya, 2026, pp. 625-631, doi: 10.1109/MI-STA68962.2026.11511256.

[8] M. A. Kamal, H. W. Raza, M. M. Alam, M. M. Su’ud, and A. A. B. Sajak, “Resource allocation schemes for 5G network: A systematic review,” Sensors, vol. 21, no. 19, Art. no. 6588, 2021.

[9] W. Y. B. Lim, N. C. Luong, D. T. Hoang, Y. Jiao, Y.-C. Liang, Q. Yang, D. Niyato, and C. Miao, “Federated learning in mobile edge networks: A comprehensive survey,” arXiv preprint arXiv:1909.11875, 2019.

[10] M. M. Saeed, et al, "Utilizing Deep Reinforcement Learning for Task Offloading and Resource Management in MEC-Enabled 6G Networks," 2025 10th International Conference on Computer and Communication Engineering (ICCCE), Kuala Lumpur, Malaysia, 2025, pp. 363-368, doi: 10.1109/ICCCE66530.2025.11474095.

[11] M. M. Amiri and D. Gündüz, “Machine learning at the wireless edge: Distributed stochastic gradient descent over-the-air,” IEEE Transactions on Signal Processing, 2019.

[12] E. Dahlman, S. Parkvall, and J. Skold, 5G NR: The Next Generation Wireless Access Technology. Amsterdam, The Netherlands: Elsevier Academic Press, 2020.

[13] M. Pons, E. Valenzuela, B. Rodríguez, J. A. Nolazco-Flores, and C. Del-Valle-Soto, "Utilization of 5G technologies in IoT applications: Current limitations by interference and network optimization difficulties—A review," Sensors, 2023.

[14] M. Barakat, R. A. Saeed and S. Edam, "Enhancing Vehicular Service Allocation with MEC Dynamic Task Offloading in a Dammam District Using CloudSim-SUMO," 2025 22nd International Learning and Technology Conference (L&T), Jeddah, Saudi Arabia, 2025, pp. 192-197, doi: 10.1109/LT64002.2025.10940140.

[15] N. H. Mahmood et al., "Machine type communications: Key drivers and enablers towards the 6G era," EURASIP Journal on Wireless Communications and Networking, 2021.

[16] A. E. Ahmed et al., "Adaptive Federated Learning Based Resource Management in Mobile Edge Computing for Machine Type Communication in 5G Network," 2026 IEEE 5th International Maghreb Meeting of the Conference on Sciences and Techniques of Automatic Control and Computer Engineering (MI-STA), Sebha, Libya, 2026, pp. 625-631, doi: 10.1109/MI-STA68962.2026.11511256.

[17] M. Barakat et al., "Energy-Aware Task Completion Delay Optimization of Space-Aerial Enabled MEC System," 2024 IEEE 4th International Maghreb Meeting of the Conference on Sciences and Techniques of Automatic Control and Computer Engineering (MI-STA), Tripoli, Libya, 2024, pp. 440-445, doi: 10.1109/MI-STA61267.2024.10599674.

[18] N. A. Mohammed, A. M. Mansoor, and R. B. Ahmad, "Mission-critical machine-type communication: An overview and perspectives towards 5G," IEEE Access, 2019.

[19] E. Dutkiewicz, X. Costa-Perez, I. Z. Kovacs, and M. Mueck, "Massive machine-type communications," IEEE Network, 2017.

[20] J. Zheng, K. Li, N. Mhaisen, W. Ni, E. Tovar, and M. Guizani, "Federated learning for online resource allocation in mobile edge computing: A deep reinforcement learning approach," in Proc. IEEE Wireless Communications and Networking Conference (WCNC), Glasgow, U.K., 2023, pp. 1–6.

[21] F. M. B. Abdo, et al, "Tri-Band Circular Microstrip Patch Antenna with Slits and Multilayered Substrate for 5G mmWave Communications," 2025 5th International Conference on Emerging Smart Technologies and Applications (eSmarTA), Ibb, Yemen, 2025, pp. 1-7, doi: 10.1109/eSmarTA66764.2025.11132136.

[22] S. Li, L. D. Xu, and S. Zhao, “5G Internet of Things: A survey,” Journal of Industrial Information Integration, vol. 10, pp. 1–9, 2018.
[23] S. F. Ahmed et al., “Toward a secure 5G-enabled Internet of Things: A survey on requirements, privacy, security, challenges, and opportunities,” IEEE Access, 2024.
[24] H. Li, K. Ota, and M. Dong, “Learning IoT in edge: Deep learning for the Internet of Things with edge computing,” IEEE Network, vol. 33, no. 3, pp. 50–57, 2019.
[25] S. Rahman et al., “Resource management across edge server in mobile edge computing,” IEEE Access, 2024.
[26] C. Sun, A. Shrivastava, S. Singh, and A. Gupta, “Revisiting unreasonable effectiveness of data in deep learning era,” in Proc. IEEE International Conference on Computer Vision (ICCV), 2017, pp. 843–852.
[27] F. M. B. Abdo et al., "Design of a Wideband Circular Microstrip Patch Antenna for Ka-Band 5G Wireless Communications," 2025 5th International Conference on Emerging Smart Technologies and Applications (eSmarTA), Ibb, Yemen, 2025, pp. 1-5, doi: 10.1109/eSmarTA66764.2025.11132274.
[28] K. Bonawitz, H. Eichner, W. Grieskamp, and D. Huba, “Towards federated learning at scale: System design,” arXiv preprint arXiv:1902.01046, 2019.
[29] H. Wang, Z. Kaplan, D. Niu, and B. Li, “Optimizing federated learning on non-IID data with reinforcement learning,” in Proc. IEEE Conference on Computer Communications (INFOCOM), 2020, pp. 1698–1707.
[30] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, “Federated learning: Strategies for improving communication efficiency,” arXiv preprint arXiv:1610.05492, 2016.
[31] L. Huang, S. Bi, and Y. J. A. Zhang, “Deep reinforcement learning for online computation offloading in wireless powered mobile-edge computing networks,” IEEE Transactions on Mobile Computing, vol. 19, no. 11, pp. 2581–2593, 2020.
[32] M. H. Babikir, et al., "Optimization Efficiency of 5G MIMO Cooperative Spectrum Sharing NOMA Networks," 2024 9th International Conference on Mechatronics Engineering (ICOM), Kuala Lumpur, Malaysia, 2024, pp. 183-187, doi: 10.1109/ICOM61675.2024.10652572.
[33] V. Mnih et al., “Human-level control through deep reinforcement learning,” Nature, 2015.
[34] H. Zhang, Z. Xie, R. Zarei, T. Wu, and K. Chen, “Adaptive client selection in resource constrained federated learning systems: A deep reinforcement learning approach,” IEEE Access, vol. 9, pp. 98423–98432, 2021, doi: 10.1109/ACCESS.2021.3095915.

[35] T. Nishio and R. Yonetani, “Client selection for federated learning with heterogeneous resources in mobile edge,” in Proc. IEEE International Conference on Communications (ICC), Shanghai, China, 2019, pp. 1–7, doi: 10.1109/ICC.2019.8761315.

[36] W. U. Rehman, T. Salam, A. Almogren, K. Haseeb, I. Ud Din, and S. H. Bouk, “Improved resource allocation in 5G MTC networks,” IEEE Access, vol. 8, pp. 49187–49197, 2020, doi: 10.1109/ACCESS.2020.2974632.

[37] S. Wang et al., “Adaptive federated learning in resource-constrained edge computing systems,” IEEE Journal on Selected Areas in Communications, vol. 37, no. 6, pp. 1205–1221, 2019, doi: 10.1109/JSAC.2019.2904348.

[38] M. A. M. Ali, et al., "Design and Performance Analysis of a 38 GHz Microstrip Patch Antenna with Slits Loading for 5G Millimeter-Wave Communications," 2023 3rd International Conference on Emerging Smart Technologies and Applications (eSmarTA), Taiz, Yemen, 2023, pp. 1-6, doi: 10.1109/eSmarTA59349.2023.10293289.

[39] T. X. Tran, A. Hajisami, P. Pandey, and D. Pompili, “Collaborative mobile edge computing in 5G networks: New paradigms, scenarios, and challenges,” IEEE Communications Magazine, vol. 55, no. 4, pp. 54–61, Apr. 2017.

[40] C. Liu et al., “A new deep learning-based food recognition system for dietary assessment on an edge computing service infrastructure,” IEEE Transactions on Services Computing, 2017.

[41] M. Hassan et al., "BER Improvement of Cooperative Spectrum Sharing of NOMA in 5G Network," 2023 IEEE 3rd International Maghreb Meeting of the Conference on Sciences and Techniques of Automatic Control and Computer Engineering (MI-STA), Benghazi, Libya, 2023, pp. 647-652, doi: 10.1109/MI-STA57575.2023.10169494.

[42] X. Wang, Y. Han, V. C. M. Leung, D. Niyato, X. Yan, and X. Chen, “Convergence of edge computing and deep learning: A comprehensive survey,” IEEE Communications Surveys & Tutorials, vol. 22, no. 2, pp. 869–904, 2020.

[43] Z. Zhou, H. Wu, S. Chen, Y. Zhang, and X. Zhang, “Federated learning meets edge computing in 5G ecosystems,” TechRxiv, 2024.

[44] A.-C. Eriksson, M. Forsman, H. Ronkainen, P. Willars, and C. Östberg, “5G NR RAN & transport options,” Ericsson Technology Review, 2019.

[45] M. S. Elbasheir, et al, "5G Base Station Deployment Review for RF Radiation," 2021 International Symposium on Networks, Computers and Communications (ISNCC), 2021, pp. 1-5, doi: 10.1109/ISNCC52172.2021.9615689.

[46] N. Zhao, X. Liu, J. Sun, X. Wang, W. Zhong, and X. Wang, “5G as a wireless power grid,” Scientific Reports, vol. 11, Art. no. 636, 2021.

[47] B. Ababio, J. Bieniek, M. Rahouti, T. Hayajneh, M. Aledhari, D. C. Verma, and A. Chehri, “A blockchain-assisted federated learning framework for secure and self-optimizing digital twins in industrial IoT,” Future Internet, vol. 17, no. 1, Art. no. 13, 2025, doi: 10.3390/fi17010013.

[48] R. Khanam, M. Hussain, R. Hill, and P. Allen, “A comprehensive review of convolutional neural networks for defect detection in industrial applications,” IEEE Access, vol. 12, pp. 94250–94295, 2024, doi: 10.1109/ACCESS.2024.3425166.

[49] M. L. Littman, T. L. Dean, and L. P. Kaelbling, “On the complexity of solving Markov decision problems,” arXiv preprint arXiv:1302.4971, 2013.

[50] S. Trindade, L. F. Bittencourt, and N. L. S. da Fonseca, “Resource management at the network edge for federated learning,” Digital Communications and Networks, 2024.

[51] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” Proceedings of the IEEE, vol. 86, no. 11, pp. 2278–2324, 1998.

[52] Imteaj, U. Thakker, S. Wang, J. Li, and M. H. Amini, “A survey on federated learning for resource-constrained IoT devices,” IEEE Internet of Things Journal, 2022.

APPENDIX A — FULL SOURCE CODE

```
import numpy as np

import torch

import torch.nn as nn

import torch.optim as optim

import random

from collections import deque

import matplotlib.pyplot as plt

import torchvision

import torchvision.transforms as transforms

import logging

from sklearn.metrics import confusion_matrix, precision_score,
recall_score, f1_score, roc_auc_score

import seaborn as sns

import csv

from collections import Counter

import matplotlib

from sklearn.metrics import confusion_matrix, precision_score,
recall_score, f1_score, roc_auc_score, roc_curve, auc

from sklearn.preprocessing import label_binarize

import sys

matplotlib.use('Agg')

# Setup logging
```

```
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(message)s',
    handlers=[
        logging.FileHandler('15dec-output.txt', mode='a',
encoding='utf-8'),
        logging.StreamHandler(sys.stdout)
    ]
)
# Set random seeds for reproducibility
SEED = 42
np.random.seed(SEED)
random.seed(SEED)
torch.manual_seed(SEED)
torch.cuda.manual_seed_all(SEED)
torch.backends.cudnn.deterministic = True
torch.backends.cudnn.benchmark = False
# Parameters
GAMMA = 0.99
LR = 0.001
MIN_BATCH_SIZE = 32
MEMORY_SIZE = 10000
EPSILON = 0.1
EPSILON_MAX = 1.0
```

```
EPSILON_GROWTH = 1.001

EPSILON_MIN = 0.0

EPSILON_DECAY = 0.995

TARGET_UPDATE = 10

NUM_EPISODES = 1000

NUM_DEVICES = 4

FEATURES_PER_DEVICE = 3

INPUT_DIM = (NUM_DEVICES, FEATURES_PER_DEVICE)

DESIRED_ACCURACY = 0.86

MAX_ITERATIONS = 1

ALPHA_N = 3.0

ALPHA_E = 2.0

ALPHA_L = 2.0

L_MAX = 500

G = 7000

TAU = 1e-28

MU_MB = 1

MU_BITS = MU_MB * 8 * 1e6

D = 20 * 1e6

LAMBDA = 1

ENERGY_SCALE = 1e12

SAMPLES_PER_DEVICE = 1276

SAMPLES_PER_CLASS = SAMPLES_PER_DEVICE // 10
```

```
# CNN for DDQN

class CNN(nn.Module):

    def __init__(self, input_channels, output_dim):

        super(CNN, self).__init__()

        self.conv1 = nn.Conv2d(input_channels, 64, kernel_size=3,
padding=1)

        self.pool1 = nn.MaxPool2d(2, 2)

        self.conv2 = nn.Conv2d(64, 64, kernel_size=3, padding=1)

        self.pool2 = nn.MaxPool2d(2, 2)

        self.conv3 = nn.Conv2d(64, 64, kernel_size=3, padding=1)

        self.fc1 = nn.Linear(64 * 4 * 4, 64)

        self.fc2 = nn.Linear(64, output_dim)


    def forward(self, x):

        x = torch.relu(self.conv1(x))

        x = self.pool1(x)

        x = torch.relu(self.conv2(x))

        x = self.pool2(x)

        x = torch.relu(self.conv3(x))

        x = x.view(x.size(0), -1)

        x = torch.relu(self.fc1(x))

        x = self.fc2(x)

        return x
```

```python
# Global Model for federated learning
class GlobalModel(nn.Module):
    def __init__(self):
        super(GlobalModel, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, kernel_size=3, padding=1)
        self.conv2 = nn.Conv2d(32, 32, kernel_size=3, padding=1)
        self.pool1 = nn.MaxPool2d(2, 2)
        self.conv3 = nn.Conv2d(32, 64, kernel_size=3, padding=1)
        self.conv4 = nn.Conv2d(64, 64, kernel_size=3, padding=1)
        self.pool2 = nn.MaxPool2d(2, 2)
        self.fc1 = nn.Linear(64 * 7 * 7, 128)
        self.dropout = nn.Dropout(0.2)
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
        x = torch.relu(self.conv1(x))
        x = torch.relu(self.conv2(x))
        x = self.pool1(x)
        x = torch.relu(self.conv3(x))
        x = torch.relu(self.conv4(x))
        x = self.pool2(x)
        x = x.view(x.size(0), -1)
```

```
        x = torch.relu(self.fc1(x))
        x = self.dropout(x)
        x = self.fc2(x)
        return x


# FiveGNetwork
class FiveGNetwork:
    def __init__(self, base_bandwidth=1000, base_latency=0.005,
capacity=1000, spectrum="sub6"):
        self.base_bandwidth = base_bandwidth
        self.base_latency = base_latency
        self.capacity = capacity
        self.connected_devices = {}
        self.spectrum = spectrum
        self.network_slice = {
            "bandwidth": base_bandwidth,
            "latency": base_latency,
        }
        self.interference_factor = 0.01
        self.fading_factor = 0.97 if spectrum == "sub6" else 0.9
        self.packet_loss_rate = 0.0003
        self.throughput_history = []
        self.latency_history = []
        self.packet_loss_history = []
```

```
    def connect_device(self, device_id):
        if len(self.connected_devices) >= self.capacity:
            logging.info(f"5G Network: capacity exceeded for device
{device_id}, using fallback.")
            return 10, 0.01, self.packet_loss_rate

        self.connected_devices[device_id] = True
        slice_info = self.network_slice
        num_devices = len(self.connected_devices)
        interference = self.interference_factor * (num_devices - 1)
        variation = np.random.uniform(0.95, 1.05) * self.fading_factor
        effective_bandwidth = slice_info["bandwidth"] * (1 -
interference) * variation
        effective_bandwidth = max(5, effective_bandwidth)
        latency = np.random.uniform(0.003, 0.01) * (1 + 0.005 *
num_devices)
        latency = min(latency, 0.005)
        packet_loss = self.packet_loss_rate * (1 +
np.random.uniform(0.01, 0.05) * num_devices)
        if self.spectrum == "mmWave":
            packet_loss *= 1.3
            packet_loss = min(packet_loss, 0.05)

        self.throughput_history.append(effective_bandwidth)
```

```
        self.latency_history.append(latency)

        self.packet_loss_history.append(packet_loss)


        logging.info(f"Device {device_id} connected:
bandwidth={effective_bandwidth:.2f} Mbps, "

                     f"latency={latency*1000:.2f} ms,
packet_loss={packet_loss:.4f}, service=mMTC")


        return effective_bandwidth, latency, packet_loss


    def disconnect_device(self, device_id):

        if device_id in self.connected_devices:

            del self.connected_devices[device_id]

            logging.info(f"Device {device_id} disconnected.")


    def reallocate_resources(self, selected_devices, devices):

        num_devices = len(self.connected_devices)

        num_active = len(selected_devices)

        if num_devices == 0:

            return

        active_bandwidth = self.base_bandwidth * 0.8 / max(1,
num_active) if num_active > 0 else self.base_bandwidth

        inactive_bandwidth = max(5, self.base_bandwidth * 0.2 / max(1,
num_devices - num_active))

        for device_id in self.connected_devices:
```

```
            original_bandwidth = self.network_slice["bandwidth"]

            self.network_slice["bandwidth"] = active_bandwidth if
device_id in selected_devices else inactive_bandwidth

            effective_bandwidth, latency, packet_loss =
self.connect_device(device_id)

            devices[device_id].update_network_params(effective_bandwidt
h, latency, packet_loss)

            self.network_slice["bandwidth"] = original_bandwidth

        logging.info(f"Resource reallocation:
active_bandwidth={active_bandwidth:.2f} Mbps,
inactive_bandwidth={inactive_bandwidth:.2f} Mbps")



    def simulate_channel_conditions(self):

        self.fading_factor = np.random.uniform(0.9, 0.98) if
self.spectrum == "sub6" else np.random.uniform(0.85, 0.95)

        self.interference_factor = np.random.uniform(0.005, 0.01)

        logging.info(f"Channel conditions updated:
fading={self.fading_factor:.2f},
interference={self.interference_factor:.2f}")



    def get_network_status(self):

        return {

            "connected_devices": len(self.connected_devices),

            "average_throughput": np.mean(self.throughput_history) if
self.throughput_history else 0,

            "average_latency": np.mean(self.latency_history) if
self.latency_history else self.base_latency,
```

```
            "average_packet_loss": np.mean(self.packet_loss_history) if
self.packet_loss_history else self.packet_loss_rate,
            "total_bandwidth": self.network_slice["bandwidth"]
        }


# EdgeDevice
class EdgeDevice:
    def __init__(self, id, cpu_freq, energy, bandwidth, fiveg_network):
        self.id = id
        self.cpu_freq = cpu_freq
        self.energy = energy
        self.base_bandwidth = bandwidth
        self.fiveg_network = fiveg_network
        self.model = GlobalModel()
        self.optimizer = optim.Adam(self.model.parameters(), lr=0.0001)
        self.criterion = nn.CrossEntropyLoss()
        self.local_data = None
        self.effective_bandwidth, self.latency, self.packet_loss =
self.fiveg_network.connect_device(self.id)


    def update_network_params(self, effective_bandwidth, latency,
packet_loss):
        self.effective_bandwidth = effective_bandwidth
        self.latency = latency
```

```
        self.packet_loss = packet_loss

        logging.info(f"Device {self.id} updated:
bandwidth={self.effective_bandwidth:.2f} Mbps, "

                     f"latency={self.latency*1000:.2f} ms,
packet_loss={self.packet_loss:.4f}")


    def set_local_data(self, data):

        self.local_data = data

        logging.info(f"Device {self.id}: assigned {len(data) if data
else 0} samples")


    def charge_energy(self, w=1):

        return np.random.poisson(w* 0.8) * ENERGY_SCALE


    def train_local_model(self, epochs, energy_rate=1):

        if not hasattr(self, 'call_counter'):

            self.call_counter = 0

        self.call_counter += 1


        if self.local_data is None:

            logging.info(f"Device {self.id}: no assigned data, skipping
training.")

            return self.model.state_dict(), 0, 0, 0


        device = torch.device("cpu")
```

```
        self.model.to(device)

        self.model.train()


        batch_size = 1276



        loader = torch.utils.data.DataLoader(self.local_data,
batch_size=batch_size, shuffle=True, drop_last=False)


        for data, target in loader:

            data, target = data.to(device), target.to(device)

            self.optimizer.zero_grad()

            output = self.model(data)

            loss = self.criterion(output, target)

            loss.backward()

            self.optimizer.step()


        T_local = MU_BITS * G / self.cpu_freq if self.cpu_freq > 0 else
float('inf')

        B_k = (self.cpu_freq ** 2) * TAU * MU_BITS * G * ENERGY_SCALE

        C_k = self.charge_energy(energy_rate)

        logging.info(f"Device {self.id}: charged with {C_k} energy
units.")

        if self.energy < B_k:
```

```
            logging.info(f"Device {self.id}: insufficient energy
({self.energy:.2f} < {B_k:.2f}), skipping training.")

            return self.model.state_dict(), 0, 0, 0

        self.energy = max(self.energy - B_k + C_k, 0)

        T_trans = D / self.effective_bandwidth if
self.effective_bandwidth > 0 else float('inf')

        return self.model.state_dict(), T_local, T_trans, B_k



    def report_resources(self):

        return {'cpu_freq': self.cpu_freq, 'energy': self.energy,
'bandwidth': self.effective_bandwidth}



    def __del__(self):

        self.fiveg_network.disconnect_device(self.id)



# MECServer

class MECServer:

    def __init__(self, num_devices=NUM_DEVICES):

        self.fiveg_network = FiveGNetwork()

        self.global_model = GlobalModel()

        self.devices = []

        for i in range(num_devices):

            cpu_freq = np.random.uniform(0, 1)

            energy = np.random.uniform(2, 5)
```

```
            bandwidth = np.random.uniform(0, 2)

            device = EdgeDevice(i, cpu_freq, energy, bandwidth,
self.fiveg_network)

            self.devices.append(device)

        self.energy_history = []

        self.bandwidth_history = []

        self.class_indices = {i: [] for i in range(10)}

        self.class_pointers = {i: 0 for i in range(10)}

        self.distributed_indices = []

        logging.info(f"Created {len(self.devices)} unique devices")


    def initialize_class_indices(self, trainset):

        for idx in range(len(trainset)):

            _, label = trainset[idx]

            self.class_indices[label].append(idx)

        for label in self.class_indices:

            logging.info(f"Class {label}:
{len(self.class_indices[label])} samples")


    def distribute_data(self, trainset, selected_devices):

        if not self.class_indices[0]:

            self.initialize_class_indices(trainset)


        for device_id in selected_devices:
```

```
            device = self.devices[device_id]
            device_indices = []

            for label in range(10):
                class_list = self.class_indices[label]
                class_len = len(class_list)
                if class_len == 0:
                    logging.warning(f"Class {label}: no samples
available")
                    continue

                pointer = self.class_pointers[label]
                num_samples = SAMPLES_PER_CLASS

                selected_indices = []
                if pointer + num_samples > class_len:
                    selected_indices.extend(class_list[pointer:])
                    remaining = num_samples - (class_len - pointer)
                    selected_indices.extend(class_list[:remaining])
                    new_pointer = remaining
                else:
                    selected_indices.extend(class_list[pointer :
pointer + num_samples])
                    new_pointer = pointer + num_samples
```

```
                self.class_pointers[label] = new_pointer % class_len

                device_indices.extend(selected_indices)

            if device_indices:

                subset = torch.utils.data.Subset(trainset,
device_indices)

                device.set_local_data(subset)

                labels_for_device = [trainset[idx][1] for idx in
device_indices]

                label_counts = dict(Counter(labels_for_device))



                logging.info(f" Device {device_id} received
{len(labels_for_device)} samples")

                for cls, count in sorted(label_counts.items()):

                    logging.info(f"   - Class {cls}: {count} samples")

                logging.debug(f"   Indices: {device_indices}")

            else:

                logging.warning(f" No data distributed to device
{device_id}")



    def fed_avg(self, local_weights):

        avg_weights = {key: torch.zeros_like(local_weights[0][key]) for
key in local_weights[0]}

        num_clients = len(local_weights)

        for weights in local_weights:
```

```
            for key in weights:
                avg_weights[key] += weights[key] / num_clients
        return avg_weights


    def simulate_training_round(self, selected_devices, epochs=10,
episode=0):
        np.random.seed(SEED + episode)
        random.seed(SEED + episode)
        local_weights = []
        total_energy = 0
        total_bandwidth = 0
        delays = []
        logging.info(f"Selected devices: {selected_devices}")
        self.fiveg_network.simulate_channel_conditions()
        self.fiveg_network.reallocate_resources(selected_devices,
self.devices)


        self.distribute_data(trainset, selected_devices)


        for device_id in selected_devices:
            device = self.devices[device_id]
            total_bandwidth += device.effective_bandwidth
            weights, T_local, T_trans, energy_used =
device.train_local_model(epochs)
```

```
            logging.info(f"Device {device_id}: energy used:
{energy_used} J, bandwidth: {device.effective_bandwidth:.2f} Mbps")

            delay = T_local + T_trans

            delays.append(delay)

            if energy_used > 0:

                local_weights.append(weights)

                total_energy += energy_used


        if local_weights:

            new_weights = self.fed_avg(local_weights)

            self.global_model.load_state_dict(new_weights)

        self.energy_history.append(total_energy)

        self.bandwidth_history.append(total_bandwidth)

        max_latency = max(delays) if delays else 0

        return max_latency, total_energy, total_bandwidth,
self.energy_history


    def evaluate_global_model(self, testloader):

        self.global_model.eval()

        correct = 0

        total = 0

        device = torch.device("cpu")

        self.global_model.to(device)

        all_preds = []
```

```
        all_targets = []
        all_probs = []
        with torch.no_grad():
            for data, target in testloader:
                data, target = data.to(device), target.to(device)
                outputs = self.global_model(data)
                probs = torch.softmax(outputs, dim=1)
                _, predicted = torch.max(outputs.data, 1)
                total += target.size(0)
                correct += (predicted == target).sum().item()
                all_preds.extend(predicted.cpu().numpy())
                all_targets.extend(target.cpu().numpy())
                all_probs.extend(probs.cpu().numpy())
        accuracy = correct / total
        all_preds_np = np.array(all_preds)
        all_targets_np = np.array(all_targets)
        all_probs_np = np.array(all_probs)
        precision = precision_score(all_targets_np, all_preds_np,
average='macro', zero_division=0)
        recall = recall_score(all_targets_np, all_preds_np,
average='macro', zero_division=0)
        f1 = f1_score(all_targets_np, all_preds_np, average='macro',
zero_division=0)
        auc = roc_auc_score(all_targets_np, all_probs_np,
multi_class='ovr', average='macro')
```

```
        return accuracy, precision, recall, f1, auc, all_preds,
all_targets, all_probs


# DeviceSelectionEnv

class DeviceSelectionEnv:

    def __init__(self, mec_server):

        self.mec_server = mec_server

        self.num_devices = len(mec_server.devices)

        self.e_max_per_device = np.array([device.energy for device in
mec_server.devices])

        self.e_max_total = np.sum(self.e_max_per_device)

        self.action_space = [i for i in range(self.num_devices)]



    def generate_state(self):

        state = np.array([list(device.report_resources().values()) for
device in mec_server.devices])

        return state



    def step(self, action, episode=0):

        selected_indices = [i for i, val in enumerate(action) if val ==
1]

        num_selected = len(selected_indices)



        max_latency, total_energy_consumed, total_bandwidth, e =
self.mec_server.simulate_training_round(selected_indices,
episode=episode)
```

```
        self.e_max_total = np.sum(e)


        reward = (ALPHA_N * (num_selected / NUM_DEVICES)
                  - ALPHA_E * (total_energy_consumed / self.e_max_total
if self.e_max_total > 0 else 1.0)
                  - ALPHA_L * (max_latency / L_MAX))


        state = self.generate_state()
        logging.info(f"Round: reward={reward:.2f},
energy={total_energy_consumed}, bandwidth={total_bandwidth:.2f}")
        return state, reward, max_latency


    def reset(self):
        return self.generate_state()
# ReplayMemory
class ReplayMemory:
    def __init__(self, capacity):
        self.memory = deque(maxlen=capacity)


    def push(self, transition):
        self.memory.append(transition)


    def sample(self, min_batch_size):
        return random.sample(self.memory, min_batch_size)
```

```
    def __len__(self):
        return len(self.memory)
# DDQNAgent
class DDQNAgent:
    def __init__(self, input_channels, output_dim):
        self.policy_net = CNN(input_channels, output_dim)
        self.target_net = CNN(input_channels, output_dim)
        self.optimizer = optim.Adam(self.policy_net.parameters(),
lr=LR)
        self.memory = ReplayMemory(MEMORY_SIZE)
        self.epsilon = EPSILON

    def preprocess_state(self, state):
        state_array = np.array(state).flatten()
        state_array = (state_array - np.mean(state_array)) /
(np.std(state_array) + 1e-8)
        target_size = 256
        padded_state = np.pad(state_array, (0, target_size -
len(state_array)), mode='constant', constant_values=0)
        return padded_state.reshape(1, 16, 16)

    def select_action(self, state):
        n = len(state)
```

```
        action = [0] * n
        if random.random() < self.epsilon:
            num_selected = 1
            selected_indices = random.sample(range(n), num_selected)
            for idx in selected_indices:
                action[idx] = 1
        else:
            with torch.no_grad():
                state_tensor =
torch.tensor(self.preprocess_state(state),
dtype=torch.float32).unsqueeze(0)
                scores = self.policy_net(state_tensor).squeeze()
                with open("scores_output_fixed_5g_lat15dec.txt", "a")
as f:
                    f.write(f"scores: {scores.tolist()}\n")
                num_selected = 1
                selected_indices = torch.argsort(scores,
descending=True)[:num_selected].tolist()
                for idx in selected_indices:
                    action[idx] = 1
        return action


    def optimize_model(self):
        if len(self.memory) < MIN_BATCH_SIZE:
            return
```

```
        batch = self.memory.sample(MIN_BATCH_SIZE)

        states, actions, rewards, next_states = zip(*batch)

        states = torch.tensor(np.stack([self.preprocess_state(s) for s
in states]), dtype=torch.float32)

        next_states = torch.tensor(np.stack([self.preprocess_state(s)
for s in next_states]), dtype=torch.float32)

        rewards = torch.tensor(rewards, dtype=torch.float32)



        q_values = self.policy_net(states).squeeze()

        next_q_values_policy =
self.policy_net(next_states).detach().squeeze()

        next_q_values_target =
self.target_net(next_states).detach().squeeze()



        expected_q_values = []

        predicted_q_values = []

        for i in range(MIN_BATCH_SIZE):

            action_indices = [idx for idx, val in enumerate(actions[i])
if val == 1]

            if not action_indices:

                predicted_q_values.append(torch.tensor(0.0,
dtype=torch.float32))

                expected_q_values.append(rewards[i])

                continue

            q_pred = q_values[i, action_indices]

            predicted_q_values.append(q_pred.mean())
```

```
            q_next = next_q_values_target[i,
torch.argmax(next_q_values_policy[i])]
            expected_q_values.append(rewards[i] + GAMMA * q_next)

        predicted_q_values = torch.stack(predicted_q_values)
        expected_q_values = torch.stack(expected_q_values)
        loss = nn.MSELoss()(predicted_q_values, expected_q_values)
        self.optimizer.zero_grad()
        loss.backward()
        self.optimizer.step()

    def update_epsilon(self):
        self.epsilon = max(EPSILON_MIN, self.epsilon * EPSILON_DECAY)

    def update_target_network(self):
        self.target_net.load_state_dict(self.policy_net.state_dict())

def plot_confusion_matrix(true_labels, pred_labels, classes):
    cm = confusion_matrix(true_labels, pred_labels)
    plt.figure(figsize=(10, 8))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
xticklabels=classes, yticklabels=classes)
    plt.xlabel('Predicted')
    plt.ylabel('True')
```

```
    plt.title('Confusion Matrix')


def plot_roc_curves(all_targets, all_probs, classes):

    plt.figure(figsize=(10, 8))

    n_classes = len(classes)

    y_true_bin = label_binarize(all_targets,
classes=list(range(n_classes)))

    for i in range(n_classes):

        fpr, tpr, _ = roc_curve(y_true_bin[:, i], all_probs[:, i])

        roc_auc = auc(fpr, tpr)

        plt.plot(fpr, tpr, lw=2, label=f'Class {i} (AUC =
{roc_auc:.3f})')


    plt.plot([0, 1], [0, 1], 'k--', lw=1)

    plt.xlim([0.0, 1.0])

    plt.ylim([0.0, 1.05])

    plt.xlabel('False Positive Rate')

    plt.ylabel('True Positive Rate')

    plt.title('ROC Curves per class')

    plt.legend(loc="lower right")

    plt.grid(True)


# Main Simulation

if __name__ == "__main__":
```

```
    print("Starting program...")

    transform = transforms.Compose([

        transforms.ToTensor(),

        transforms.Normalize((0.5,), (0.5,))

    ])


    trainset = torchvision.datasets.MNIST(root='./data', train=True,
download=True, transform=transform)

    trainloader = torch.utils.data.DataLoader(trainset, batch_size=85,
shuffle=True, num_workers=2)

    testset = torchvision.datasets.MNIST(root='./data', train=False,
download=True, transform=transform)

    testloader = torch.utils.data.DataLoader(testset, batch_size=85,
shuffle=False, num_workers=2)


    mec_server = MECServer()

    env = DeviceSelectionEnv(mec_server)

    agent = DDQNAgent(1, NUM_DEVICES)

    rewards_per_episode = []

    episode_energy_history = []

    episode_bandwidth_history = []

    episode_accuracies = []

    iteration_accuracies = []

    iteration_precisions = []

    iteration_recalls = []
```

```
    iteration_f1s = []
    iteration_aucs = []
    episode_latency_history = []
    all_preds_list = []
    all_targets_list = []
    all_probs_list = []
    episode_precisions = []
    episode_recalls = []
    episode_f1s = []
    episode_aucs = []


    with open("episode_metrics.csv", "w", newline="", encoding='utf-8')
as f:
        writer = csv.writer(f)
        writer.writerow(["Episode", "Precision", "Recall", "F1-score",
"AUC"])


    for episode in range(NUM_EPISODES):
        logging.info(f"\nStarting episode {episode+1}")
        state = env.reset()
        total_reward = 0
        iteration = 0
        avg_reward = float('inf')
```

```
        accuracy = 0.0
        max_accuracy = 0.0
        episode_energy = 0
        episode_bandwidth = 0
        episode_iterations = 0
        episode_latency_list = []



        while accuracy < DESIRED_ACCURACY:
            action = agent.select_action(state)
            next_state, reward, max_latency = env.step(action, episode)
            agent.memory.push((state, action, reward, next_state))
            agent.optimize_model()
            state = next_state
            total_reward += reward
            iteration += 1
            episode_iterations += 1
            episode_energy += mec_server.energy_history[-1] if
mec_server.energy_history else 0
            episode_bandwidth += mec_server.bandwidth_history[-1] if
mec_server.bandwidth_history else 0
            avg_reward = total_reward / iteration if iteration > 0 else
reward
            accuracy, precision, recall, f1, auc1, all_preds,
all_targets, all_probs = mec_server.evaluate_global_model(testloader)
```

```
            precision = precision_score(all_targets, all_preds,
average='macro', zero_division=0)

            recall = recall_score(all_targets, all_preds,
average='macro', zero_division=0)

            f1 = f1_score(all_targets, all_preds, average='macro',
zero_division=0)

            all_probs = np.array(all_probs)



            n_classes = all_probs.shape[1] if all_probs.ndim > 1 else 1

            y_true_bin = label_binarize(all_targets,
classes=list(range(n_classes)))



            try:

                auc_macro = roc_auc_score(y_true_bin, all_probs,
average='macro', multi_class='ovr')

            except Exception as e:

                logging.warning(f"ROC AUC failed: {e}")

                auc_macro = None

            episode_precisions.append(precision)

            episode_recalls.append(recall)

            episode_f1s.append(f1)

            episode_aucs.append(auc_macro if auc_macro is not None else
0)



            logging.info(f"Accuracy: {accuracy:.4f}  Precision(macro):
{precision:.4f}  Recall(macro): {recall:.4f}  F1(macro): {f1:.4f}")
```

```
            print(f"Acc={accuracy:.4f} Precision={precision:.4f}
Recall={recall:.4f} F1={f1:.4f}")

            if auc_macro is not None:

                logging.info(f"AUC (macro, OVR): {auc_macro:.4f}")

                print(f"AUC (macro): {auc_macro:.4f}")

            max_accuracy = max(max_accuracy, accuracy)

            iteration_accuracies.append(accuracy)

            iteration_precisions.append(precision)

            iteration_recalls.append(recall)

            iteration_f1s.append(f1)

            iteration_aucs.append(auc1)

            all_preds_list.append(all_preds)

            all_targets_list.append(all_targets)

            all_probs_list.append(all_probs)

            episode_latency_list.append(max_latency)

            logging.info(f"Iteration {iteration}, total_reward:
{total_reward:.2f}, avg_reward: {avg_reward:.2f}, "

                         f"accuracy: {accuracy:.4f}, precision:
{precision:.4f}, recall: {recall:.4f}, f1: {f1:.4f}, auc: {auc1:.4f}, "

                         f"max_accuracy: {max_accuracy:.2f}, energy:
{episode_energy:.2f}, "

                         f"bandwidth: {episode_bandwidth:.2f}")


        agent.update_epsilon()

        if episode % TARGET_UPDATE == 0:
```

```
            agent.update_target_network()

        rewards_per_episode.append(total_reward)

        episode_accuracies.append(max_accuracy)

        episode_energy_history.append(episode_energy /
episode_iterations if episode_iterations > 0 else 0)

        episode_bandwidth_history.append(episode_bandwidth /
episode_iterations if episode_iterations > 0 else 0)

        episode_latency_history.append(np.mean(episode_latency_list) if
episode_latency_list else 0)

        with open("episode_metrics.csv", "a", newline="",
encoding='utf-8') as f:

            writer = csv.writer(f)

            writer.writerow([

                episode + 1,

                episode_precisions[-1],

                episode_recalls[-1],

                episode_f1s[-1],

                episode_aucs[-1]

            ])


        logging.info(f"Finished episode {episode+1}, total_reward:
{total_reward:.2f}, avg_reward: {avg_reward:.2f}, "

                     f"iterations: {iteration}, max_accuracy:
{max_accuracy:.2f}, energy: {episode_energy_history[-1]:.2f}, "

                     f"bandwidth: {episode_bandwidth_history[-1]:.2f}")
```

```
    logging.info(f"Episode accuracies so far: {episode_accuracies}")

    total_accuracy = np.mean(episode_accuracies) if episode_accuracies
else 0

    logging.info(f"Overall accuracy across episodes:
{total_accuracy:.4f}")

    num_iters_for_avg = min(160, len(iteration_accuracies))

    avg_accuracy = np.mean(iteration_accuracies[:num_iters_for_avg]) if
iteration_accuracies else 0

    avg_precision = np.mean(iteration_precisions[:num_iters_for_avg])
if iteration_precisions else 0

    avg_recall = np.mean(iteration_recalls[:num_iters_for_avg]) if
iteration_recalls else 0

    avg_f1 = np.mean(iteration_f1s[:num_iters_for_avg]) if
iteration_f1s else 0

    avg_auc = np.mean(iteration_aucs[:num_iters_for_avg]) if
iteration_aucs else 0

    logging.info(f"Average Metrics up to Iteration 160:
Accuracy={avg_accuracy:.4f}, Precision={avg_precision:.4f}, "

                 f"Recall={avg_recall:.4f}, F1={avg_f1:.4f},
AUC={avg_auc:.4f}")

    if len(all_preds_list) >= 160 and len(all_targets_list) >= 160:

        classes = [str(i) for i in range(10)]

        plot_confusion_matrix(all_targets_list[159],
all_preds_list[159], classes)

        plt.savefig('confusion_matrix_fixed_5g_lat_15dec.png'

        plt.close()

        plot_roc_curves(all_targets_list[159], all_probs_list[159],
classes)
```

```
            plt.savefig('roc_curves_fixed_5g_lat_15dec.png')
            plt.close()
        else:
            logging.warning("Cannot plot confusion matrix: fewer than 160
iterations completed")
        plt.figure(figsize=(10, 6))
        pIt.pIot(episode_energy_history, label="energy per episode")
        plt.xlabel("Episode")
        plt.ylabel("Energy (unit)")
        plt.title("Energy Consumption")
        plt.legend()
        plt.savefig('energy per episode_fixed_5g_lat_15dec.png')
        plt.close()
        plt.figure(figsize=(10, 6))
        plt.plot(episode_latency_history, label="Latency per episode")
        plt.xlabel("Episode")
        plt.ylabel("Latency (s)")
        plt.title("Latency per Episode")
        plt.legend()
        plt.savefig('Latency per episode_fixed_5g_lat_15dec.png')
        plt.close()
        plt.figure(figsize=(10, 6))
        plt.plot(iteration_accuracies, label="Accuracy per iteration")
        plt.xlabel("Iteration")
```

```
    plt.ylabel("Accuracy")

    plt.title("Model accuracy across episodes")

    plt.legend()

    plt.savefig('Accuracy per iteration1_fixed_5g_lat_15dec.png')

    plt.close()

    plt.figure(figsize=(10, 6))

    plt.plot(iteration_accuracies[:160], label="Accuracy per iteration
(first 160)")

    plt.xlabel("Iteration")

    plt.ylabel("Accuracy")

    plt.title("Model accuracy during first 160 iterations")

    plt.legend()

    plt.savefig('Accuracy per for 160
iteration1_fixed_5g_lat_v15dec.png')

    plt.close()

    plt.rcParams.update({

        'font.size': 12,

        'axes.grid': True,

        'grid.linestyle': '--',

        'grid.alpha': 0.7,

        'grid.linewidth': 0.8,

        'grid.color': '#dddddd',

        'axes.edgecolor': '#333333',

        'figure.facecolor': 'white',
```

```
    'axes.facecolor': '#fafafa'
})
if len(iteration_accuracies) >= 1:
    num_iters = min(160, len(iteration_precisions))
    prec_data = iteration_precisions[:num_iters]
    recall_data = iteration_recalls[:num_iters]
    f1_data = iteration_f1s[:num_iters]
    x = list(range(1, num_iters + 1))
    plt.figure(figsize=(12, 7))
    plt.plot(x, prec_data, label='Precision',
             linewidth=2, color=sns.color_palette("husl", 5)[1])
    plt.plot(x, recall_data, label='Recall',
             linewidth=2, color=sns.color_palette("husl", 5)[2])
    plt.plot(x, f1_data, label='F1-Score',
             linewidth=2, color=sns.color_palette("husl", 5)[3])
    plt.xlabel('Iteration', fontsize=12)
    plt.ylabel('Score', fontsize=12)
    plt.title('Precision, Recall and F1-Score',
             fontsize=14, fontweight='bold', pad=20)
    plt.ylim(0, 1)
    plt.grid(True, linestyle='--', alpha=0.7)
    plt.legend(fontsize=11)
    plt.tight_layout()
```

```
        plt.savefig('precision_recall_f1_first_160_iterations15dec.png'
                    dpi=300, bbox_inches='tight')
        plt.close()


        logging.info(
            f"Combined plot saved for Precision, Recall, and F1-Score "
            f"up to iteration {num_iters}.")
    else:
        logging.warning("Not enough iterations to plot
Precision/Recall/F1")
        logging.info("Metrics averages up to 160 plotted
successfully!")
```