MINI PROJECT INDEX

Project Report: Deep Reinforcement Learning for Online Offloading in MEC Networks

Total Pages: 60

Table of Contents

Front Matter

- **Title Page** 1
- **Abstract** 2
- **Table of Contents** 3-4
- **List of Figures and Tables** 5

Chapter 1: Introduction **6-9**

- 1.1. The Rise of Mobile-Edge Computing (MEC) 6
- 1.2. The Challenge of Online Offloading in Wireless Networks 7
- 1.3. Proposed Solution: A Deep Reinforcement Learning Approach 8
- 1.4. Project Goals and Report Structure 9

Chapter 2: System Model and Problem Formulation **10-17**

- 2.1. Network Architecture and System Model 10
- 2.2. Time-Division Multiplexing and Resource Allocation 12
- 2.3. Local Computing Model 13
- 2.4. Edge Computing (Offloading) Model 14
- 2.5. Formal Problem Formulation 15
- 2.6. Limitations of Traditional Optimization and the Need for DRL 17

Chapter 3: Exploratory Data Analysis (EDA) **18-32**

- 3.1. Dataset Overview and Preparation 18
- 3.2. High-Level Statistical Summary 19
- 3.3. Deep Dive into the Environment: Channel Gain Analysis 20
  - 3.3.1. Distribution of Channel Gains Per User 21
  - 3.3.2. Time-Series Fluctuation of Channel Gains 22
  - 3.3.3. Correlation of Gains Between Users 23
- 3.4. Understanding the "Oracle": Analysis of Optimal Decisions 24
  - 3.4.1. Overall Offloading vs. Local Processing Ratio 24
  - 3.4.2. Offloading Frequency on a Per-User Basis 25
  - 3.4.3. Distribution of Optimal Network Performance (Q*) 26
  - 3.4.4. Analysis of Resource Allocation Times (a_time, tau) 27
- 3.5. The Core Insight: Connecting Channel Conditions to Decisions 28
  - 3.5.1. The Relationship Between Channel Gain and Offloading Decisions 28
  - 3.5.2. Time-Series Overlay: Visualizing Decisions in Real-Time 30

    - 3.5.3. Impact of Collective Behavior on System Performance 31
- 3.6. EDA Conclusion: Key Findings and Implications for Agent Design 32

Chapter 4: The DROO Agent - Design and Implementation 33-40

- 4.1. The Deep Reinforcement Learning Framework 33
- 4.2. The Memory-Augmented DNN: Architecture and Rationale 34
- 4.3. The Agent-Environment Interaction Loop 35
- 4.4. Core Component 1: The MemoryDNN Agent (agent.py) 36
    - 4.4.1. Class Initialization and Network Definition 36
    - 4.4.2. The decode Method: Making Decisions 37
    - 4.4.3. The encode and experience_replay Methods: Learning from Memory 38
- 4.5. Core Component 2: The Computation Rate Optimizer (optimizer.py) 39
    - 4.5.1. The Role of the Bisection Method 39
    - 4.5.2. Calculating the Reward Signal 40

Chapter 5: Agent Training and Performance Analysis 41-48

- 5.1. Experiment Configuration and Hyperparameters 41
- 5.2. Data Handling and Preparation for Training 42
- 5.3. The Training Loop in Detail 43
- 5.4. Analyzing Agent Convergence and Learning 44
    - 5.4.1. Training Cost (MSE Loss) Over Time 45
    - 5.4.2. Normalized Computation Rate vs. The Oracle 46
- 5.5. Final Performance Evaluation on Unseen Test Data 47
- 5.6. Saving the Trained Model and Performance History 48

Chapter 6: Agent Evaluation in Dynamic Scenarios 49-56

- 6.1. Evaluation Methodology and Model Loading 49
- 6.2. Demo 1: Adaptability to Changing User Weights 50
    - 6.2.1. Scenario Description 50
    - 6.2.2. Results and Performance Analysis 51
- 6.3. Demo 2: Robustness to WDs Turning On and Off 53
    - 6.3.1. Scenario Description 53
    - 6.3.2. Results and Performance Analysis 54
- 6.4. Overall Evaluation Summary and Conclusion 56

Chapter 7: Conclusion 57

- 7.1. Summary of Findings 57
- 7.2. Future Work and Potential Enhancements 57

References 58

Appendices 59-60

- **Appendix A: Full Code Listing for agent.py** 59

- **Appendix B: Full Code Listing for optimizer.py** 60

Chapter 1: Introduction

1.1 The Rise of Mobile-Edge Computing (MEC)

(Page 6)

The last decade has witnessed a paradigm shift in the technological landscape, characterized by the exponential growth of the Internet of Things (IoT). Billions of interconnected devices, ranging from simple wireless sensors to complex augmented reality (AR) headsets, are now woven into the fabric of our daily lives. These devices are continuously generating unprecedented volumes of data, driving demand for applications that require intensive computation and real-time responsiveness.

However, the very nature of these devices—often small, battery-powered, and mobile—imposes fundamental constraints on their capabilities. They are fundamentally limited by their onboard processing power and finite battery life. For computationally intensive applications, relying solely on a device's local resources is often infeasible, leading to poor performance, high energy consumption, and an unsatisfactory user experience.

The traditional solution to this limitation has been to offload computational tasks to powerful, centralized cloud data centers. While the cloud offers virtually limitless computational power, it introduces a significant bottleneck: latency. The round-trip time required to send data from a device to a distant cloud server, process it, and return the result is often too long for applications that demand immediate feedback, such as industrial automation, real-time analytics, or immersive AR/VR experiences.

To bridge this gap between local device limitations and cloud latency, **Mobile-Edge Computing (MEC)** has emerged as a transformative network architecture. The core principle of MEC is to bring the intelligence and resources of the cloud closer to the end-user, deploying computation, storage, and networking services at the "edge" of the network. This edge infrastructure, often co-located with cellular base stations or wireless access points (APs), acts as a powerful intermediary.

By offloading tasks to a nearby MEC server instead of a distant cloud, devices can achieve a multitude of benefits:

- **Reduced Latency:** Drastically cuts down the communication delay, enabling real-time application performance.
- **Decreased Energy Consumption:** Reduces the energy-intensive task of long-range data transmission, thereby extending device battery life.
- **Enhanced Capability:** Empowers low-power devices to execute complex tasks far beyond their native processing capabilities.

This project specifically considers a **Wireless Powered MEC network**, a cutting-edge paradigm where the access point is responsible not only for receiving computation offloads but also for broadcasting Radio Frequency (RF) energy to power the wireless devices. This dual function creates a self-sustaining ecosystem, making it a highly promising solution for deploying long-lasting IoT and sensor networks without the need for manual battery replacement.

1.2 The Challenge of Online Offloading in Wireless Networks

(Page 7)

While MEC provides the necessary infrastructure, it introduces a critical and complex decision-making problem at the heart of the system: **computation offloading**. For every task generated by a wireless device (WD), a decision must be made: should the task be processed locally on the device, or should it be offloaded to the MEC server for remote execution? This is known as the binary offloading policy.

This decision is far from trivial and is fraught with challenges that make it a formidable optimization problem, particularly in a real-world, dynamic setting.

1. **The Highly Dynamic and Volatile Environment:** The primary medium for communication, the wireless channel, is inherently unreliable and unpredictable. As demonstrated in our exploratory data analysis, channel gains fluctuate rapidly and significantly over time due to factors like fading, interference, and mobility. A strong channel that is ideal for offloading at one moment can become weak and unsuitable in the next millisecond. An effective offloading policy must therefore be **online** and adaptive, capable of reacting instantly to these changes.

2. **Multi-User Resource Contention:** In a typical MEC network, multiple users are connected to the same access point and must compete for shared, finite resources. These resources include the time allocated for wireless power transfer (energy harvesting) and the time allocated for uplink transmission (offloading data). The offloading decision for one user directly impacts the resources available to all other users. For example, if one user is allocated a long time slot to offload a large task, less time is available for other users to either offload their tasks or harvest energy. This creates a deeply interconnected system where decisions must be jointly optimized across all users.
3. **The Combinatorial Explosion:** The multi-user nature of the problem leads to a combinatorial explosion in the decision space. For a network with *N* users, each with a binary choice (local or offload), there are **2^N** possible offloading configurations at any given moment. Even for a modest network of 30 users, this number exceeds one billion, making an exhaustive search for the optimal decision computationally impossible to perform in real-time.

Traditional approaches to solving this problem, such as mixed-integer programming (MIP) solvers using branch-and-bound algorithms or heuristic methods based on convex relaxation, fall short. While optimal solvers can find the best solution, they are computationally prohibitive and far too slow for an online setting. Heuristic methods, while faster, sacrifice optimality and do not guarantee high-quality solutions. Both approaches are fundamentally unsuited for the dynamism of fast-fading wireless channels, as they cannot re-solve the complex optimization problem with the millisecond-level frequency required. This creates a critical "optimality-efficiency" tradeoff and necessitates a new approach capable of delivering high-quality decisions at the speed of the network.

1.3 Proposed Solution: A Deep Reinforcement Learning Approach

(Page 8)

To overcome the limitations of traditional optimization methods, this project proposes and implements a novel solution based on **Deep Reinforcement Learning (DRL)**. DRL is a powerful class of machine learning that trains an autonomous agent to make optimal sequential decisions in a complex and dynamic environment by learning from experience. This paradigm is an ideal fit for the online offloading problem.

We have developed the **Deep Reinforcement Learning-based Online Offloading (DROO)** framework, which reframes the offloading problem from one of static optimization to one of intelligent, adaptive control. The framework consists of a DRL agent that learns a **policy**—a direct mapping from the current state of the network to an optimal offloading action.

The core components of the DROO framework are:

- **The State:** The agent's perception of the environment at each time frame. In our model, the state is defined as the vector of current wireless channel gains for all *N* users, $h = [h_1, h_2, ..., h_n]$. This is the most critical information for making an informed offloading decision.
- **The Action:** The decision the agent makes based on the current state. The action is a binary vector of length *N*, $x = [x_1, x_2, ..., x_n]$, where $x_i = 1$ signifies that User *i* offloads its task, and $x_i = 0$ signifies local computation.
- **The Reward:** A feedback signal from the environment that quantifies the quality of the agent's action. In our system, the reward is the **weighted sum computation rate**, which is the ultimate performance metric we aim to maximize.

Instead of solving a computationally expensive MIP problem at every step, the DROO agent leverages a highly optimized **Deep Neural Network (DNN)** to approximate the optimal policy. Once trained, this DNN can take the current channel gains as input and produce a high-quality offloading decision in a fraction of a second. This makes real-time control viable.

The key advantages of our DRL approach are:

1. **Computational Efficiency:** By replacing iterative optimization with a single forward pass through a DNN, we reduce the computation time by more than an order of magnitude (e.g., from several seconds to under 0.1 seconds for a 30-user network), enabling true real-time operation.
2. **Learning Without Labeled Data:** The agent learns its policy through direct interaction with the environment and a reward signal. It does not require a massive, pre-labeled dataset of "correct" answers, which would be prohibitively expensive to generate. It uses a **memory buffer** to store valuable state-action-reward experiences and replays them to continuously refine its neural network.
3. **Adaptability and Generalization:** Through training on thousands of different channel conditions, the agent learns a robust and generalizable policy. As demonstrated in our final evaluation, this allows it to adapt to unforeseen network dynamics, such as changing user priorities or users joining and leaving the network, without the need for retraining.

1.4 Project Goals and Report Structure

(Page 9)

The primary objective of this project is to design, train, and evaluate a DRL-based agent that can intelligently and efficiently manage computation offloading in a dynamic, multi-user MEC network.

The specific goals are threefold:

1. **To implement and train a DROO agent** capable of learning a near-optimal policy for making real-time, binary offloading decisions for multiple wireless devices.
2. **To quantitatively benchmark the performance** of the trained agent against a theoretical "oracle"—a traditional optimizer with the luxury of unlimited computation time—to prove that our DRL approach achieves near-optimal results.
3. **To evaluate the trained agent's robustness and adaptability** by subjecting it to challenging, dynamic scenarios that it has not encountered during training, such as shifting network priorities and a fluctuating number of active users.

This report documents the complete lifecycle of the project, from theoretical formulation to final evaluation. The document is structured as follows:

- **Chapter 2** details the underlying mathematical framework, presenting the wireless powered MEC system model and the formal problem formulation we aim to solve.
- **Chapter 3** presents a comprehensive Exploratory Data Analysis (EDA) of the dataset, uncovering the key relationships between channel conditions, optimal decisions, and network performance that guide our agent's design.
- **Chapter 4** describes the architecture and code-level implementation of the DROO agent, focusing on its core components: the MemoryDNN and the bisection optimizer used for reward calculation.
- **Chapter 5** details the agent training process, analyzing its learning convergence and benchmarking its final performance on an unseen test dataset.
- **Chapter 6** showcases the agent's capabilities in two live evaluation demos, testing its adaptability and robustness in dynamic, real-world-inspired scenarios.
- **Chapter 7** concludes the report with a summary of our key findings and a discussion of potential directions for future research.