National College of Ireland

Ensemble Q-Learning Algorithm: An effective and novel approach for task offloading in Edge Computing

MSc Research Project
MSc in Cloud Computing

Rakesh Malik
Student ID:
21172692

School of Computing
National College of Ireland

Supervisor: Mr. Rashid Mijumbi

National College of Ireland
Project Submission Sheet
School of Computing

National College of Ireland

Student Name:	Rakesh Malik
Student ID:	21172692
Programme:	MSc Cloud Computing
Year:	2023
Module:	MSc Research Project
Supervisor:	Rashid Mijumbi
Submission Due Date:	14/12/2023
Project Title:	Ensemble Q-Learning Algorithm: An effective and novel approach for task offloading in Edge Computing
Word Count:	XXX
Page Count:	19

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	31st January 2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Ensemble Q-Learning Algorithm: An effective and Novel approach for task offloading in Edge Computing

Rakesh Malik
21172692

Abstract

Q-Learning algorithm has emerged as a prominent approach for optimizing task offloading strategies in edge computing environments. Task offloading which is an aspect of edge computing, requires algorithms to strike a balance between resource utilization and power consumption. This research introduces the formulation of an Ensemble Q-Learning algorithm as a solution to improve the efficiency of task offloading mechanisms.

Driven by the need to enhance resource utilization and address growing concerns about power consumption the proposed algorithm incorporates a Q-Learning strategy. Taking inspiration from Random Migration, Q Learning and Graph Convolutional Network (GCN) based Q-Learning, the Ensemble Q-Learning algorithm utilizes replay and buffer mechanisms to enhance adaptability and performance.

Simulation results demonstrate a 44% reduction in power consumption compared to existing algorithms confirming the effectiveness of the proposed Ensemble Q-Learning algorithm. The findings align with the research objective of advancing sustainability and resource efficiency in edge computing systems. However, it is important to acknowledge that simulation environments have limitations. Therefore future work aims to refine and validate the proposed algorithm in real world scenarios by considering real time network conditions and evolving edge computing architectures.

1 Introduction

Picture the world of computers like a bustling city where tasks are like people moving around. Now, imagine if we could make these tasks move more efficiently to get things done faster. That is exactly what we are exploring in this report – the smart way of distributing tasks among computers at the edge which is like the outskirts of our computing city.

In the dynamic landscape of modern computing, the exponentially growing prevalence of edge computing has become a focal point, reshaping conventional paradigms and promising revolutionary advancements in service delivery. At the heart of this transformation lies the imperative need to address the challenges posed by resource-intensive applications and the escalating demands for real-time processing. One crucial aspect of this research is the study of task offloading, a critical mechanism aimed at optimizing the distribution of computational tasks within edge computing environments. The contemporary trend in edge computing revolves around leveraging distributed computation to meet the diverse needs of applications. The challenge lies bringing the computation power closer

to where the data is generated which was earlier the other way around. Now there is paradigm shift in how the mobile devices task processing is handled. The emergence of Multi-Access Edge Computing (MEC) Pham et al. (2020) addresses these challenges by bringing computational capabilities closer to the network's edge, presenting a transformative shift from traditional cloud computingCao et al. (2020).
As mentioned in the article published by Jiang et al. (2019), in various scenarios like smart homes, connected vehicles and industrial IoT, cloud computing despite its advantages faces challenges. One major issue is the substantial data transfer needed from end devices and sensors, consuming significant network bandwidth. Additionally, analyzing massive data in cloud data centers becomes impractical due to computing and storage limitations. This results in delayed responses and high latency for big data analytics from numerous IoT devices. Moreover, concerns about data privacy and security make relying on cloud data centers less trustworthy for data analytics, emphasizing the need for local processing near the data source.

Enter edge computing, an emerging paradigm offering the capability to process or store critical data locally before transmitting it to a central data center or cloud repository. For instance, in IoT applications, edge devices collect and process data on-site, only resorting to cloud processing when local capabilities are insufficient. This approach lightens the load on central cloud data centers, shifting tasks from the cloud to network edge devices and potentially eliminating the processing burden at the central location. Additionally, fog computing Habibi et al. (2020) complements traditional cloud models, providing a solution to IoT challenges and introducing a highly heterogeneous computing and network environment through fog-cloud integration. This leads me to ask the question that this paper diligently tries to answer - **How can the computational offloading be optimized to enhance the data processing speed of the edge devices?** The aim of this research is to find the answer to the above question by designing a novel method that not only speeds up the task offloading but also provide the solution for balanced usage of the CPU and Memory because then only the proposed solution can be practically applied to the low powered and resource constraint mobile devices.
Talking about the structure of the report, it is organised into following sections:-

1. **Section 1 : Introduction** : It provides the background of the research domain and sets the scene for the report.

2. **Section 2 : Related Work**: As the name suggests, related work describes the critical analysis of the work done by other notable researchers in the domain of edge computing.

3. **Section 3: Methodology** : This section talks about the procedures and methods followed for conducting the research.

4. **Section 4: Design Specification**: This section focuses on the environment in which the experiments have been performed giving the setup and architecture of the system used.

5. **Section 5: Implementation**: It gives the detailed view of the approach followed and the steps used for the proposed algorithm.

6. **Section 6: Evaluation** : This section presents the results of the experiments done and comment on the overall performance.

7. **Section 7 : Conclusion and Future Work** : This section wraps the report by giving final thoughts on the entire research conducted.

2 Related Work

2.1 Challenges in Task Offloading domain

I went through an interesting article by Islam et al. (2021) and his team in which he has conducted a detailed survey of the work ongoing in the field of task offloading in edge computing and he has identified following challenges that the domain face:

1. **Latency Requirements**: The survey underscores the inability of cloud computing technology to meet the latency demands of delay-sensitive applications due to the propagation delay between User Equipment (UE) devices and the cloud. This latency issue has led to the emergence of Multi-Access Edge Computing (MEC) as a new network paradigm to address the computation needs of resource-intensive applications running on UEs.

2. **Resource Constraints**: Resource-intensive mobile applications pose critical design challenges in achieving the desired performance for resource-constrained Mobile Devices (MDs). The survey highlights the need to address these constraints to ensure efficient task offloading in MEC environments.

3. **Classification and In-depth Analysis**: The survey addresses the lack of in-depth analysis and classification of MEC task offloading in existing literature. By providing a comprehensive survey, the authors aim to fill this gap and offer a detailed understanding of the task offloading scheme for MEC proposed by various researchers.

4. **Machine Learning Model Formulation**: The survey highlights the challenge of formulating a machine learning model that considers attributes such as computation power, task-related information, CPU cycle requirements and latency deadlines.

All of the above challenges demand an intelligent and detailed solution that can represent the environment of edge computing in realistic manner and address the above challenges especially considering the CPU utilization and power consumption requirements. on researching further into the domain I found a solution proposed by Guo et al. (2020) who conducted research on intelligent task offloading at the edge and presented a compelling integration of machine learning into mobile-edge computation offloading (MECO) offering several strengths. The proposed scheme demonstrates good adaptability providing offloading decision profiles for newly added MDs' tasks without rerunning the offloading decision-making scheme. Also, the decision tree-based offloading scheme (DTOS) exhibits higher computational efficiency compared to existing task offloading schemes achieving high prediction accuracy and scalability. However, the research also reveals weaknesses in the form of limited security, poor adaptability of previous schemes and inconclusive convergence performance of the chosen offline learning method. These limitations highlight the need for further investigation and improvement particularly in addressing security concerns and enhancing the adaptability and scalability of the proposed intelligent task offloading scheme.

2.2 Current state of the art technique for Task Offloading

While looking for the answers of the above questions raised by Guo et al. (2020), I came across one of the promising solutions in the form of Q-learning algorithm. Q-learning is employed for task offloading in edge computing by making real-time decisions on whether to process a task locally on an edge device or offload it to the cloud. The algorithm maintains a table of Q-values, adjusting them based on observed rewards to optimize factors like latency and energy consumption. Balancing exploration and exploitation, Q-learning adapts to dynamic conditions and facilitates quick, intelligent decision-making in response to changing environments. Its simplicity and effectiveness make it a practical choice, especially in scenarios where the decision-making process require real-time adjustments to unforeseen circumstances.

The algorithm's ability to learn optimal strategies through trial and error, without requiring a prior knowledge of the system further enhances its applicability in diverse settings. This makes Q-learning well-suited for environments where uncertainties and variations are inherent allowing it to continually refine its decision-making policies over time. The solution proposed by author Chen et al. (2019) presents a Q-Learning approach for optimizing computation offloading and resource allocation in mobile edge computing (MEC) systems. The paper highlights the challenges faced by MEC, such as the spatial distance between user equipment (UEs) and cloud servers which can result in extra transmission costs and potential QoS issues for latency-sensitive applications To address these challenges, the paper formulates the problem as a mixed integer non-linear programming problem (MINLP) Montoya et al. (2020) and introduces a value iteration based Reinforcement Learning (RL) approach called Q-Learning. The author and his team defines the state space, action space, reward function and introduces a Markov decision process for the proposed solution. The paper provides a detailed system model of the multi-user MEC network, including the network architecture, communication model, computing model and energy consumption of UEs. It also presents the strategy of offloading and resource allocation based on Q-Learning. However, the paper has limitations such as limited comparison with other existing approaches. It would benefit from further validation and a more comprehensive comparison with existing methods.

another work conducted by Dab et al. (2019) I reviewed introduces a novel approach, QL-Joint Task Assignment and Resource Allocation (QL-JTAR), utilizing a Q-Learning algorithm to optimize joint computation offloading and resource allocation in a multi user WiFi-based Mobile Edge Computing (MEC) architecture. I appreciate the authors' focus on the critical challenge of minimizing energy consumption on the mobile terminal side while adhering to application latency constraints. The proposed approach is substantiated through extensive simulations in the NS3 network simulator, using real input traces for performance evaluation.

One of the notable strengths of the work is its comprehensive treatment of the joint task assignment and resource allocation problem. The incorporation of a Q-Learning algorithm adds adaptability to the approach and the extensive simulations effectively showcase its superiority in terms of energy consumption and latency ensuring near-optimal solutions.

However, I find a potential weakness in the limited discussion of the approach's limitations. A more detailed exploration of scenarios where the QL-JTAR approach may not be as effective would provide a more balanced view of its applicability. Additionally, a more thorough comparison with existing literature or alternative methods would offer a

clearer perspective on the contributions and uniqueness of the paper.
it is important to note that the reliance on the NS3 network simulator may introduce skepticism about the credibility of the findings, especially in the context of mobile edge computing. I see a weakness in the choice of the NS3 simulator which might not fully capture the intricacies of real-world scenarios in this specific field. Exploring alternative simulators or considering real-world implementations could address this concern and bolster the validity of the proposed approach.

In conclusion, the challenges identified in the task offloading domain, as highlighted by Islam et al. (2021), underscore the limitations of current solutions particularly in addressing latency requirements, resource constraints, the lack of in-depth analysis and the formulation of machine learning models. While Guo et al. (2020) intelligent task offloading scheme shows promise by integrating machine learning into mobile-edge computation offloading, it suffers from limitations in security, adaptability and scalability, necessitating further investigation and improvement.

The exploration of Q-learning, as presented byChen et al. (2019) and Dab et al. (2019) offers a promising solution for task offloading in edge computing. Q-learning's adaptability to dynamic conditions and real-time decision-making aligns well with the challenges posed in the domain. However, limitations in existing studies, such as limited comparisons, lack of validation and reliance on NS3 network simulator highlight the need for a more comprehensive and validated solution.

In light of these findings, the inadequacy of previous solutions necessitates a more advanced and tailored approach. The proposed Ensemble Q-learning algorithm with replay and buffer, as discussed in the below sections aims to address the limitations of existing solutions. By leveraging reinforcement learning with replay mechanisms, the proposed algorithm seeks to enhance adaptability, scalability, CPU and memory utilization and power consumption, hence providing a robust solution to the challenges identified in the task offloading domain. This research is essential to advancing the field and offering a more effective and efficient approach to task offloading in edge computing.

3 Methodology

The research is primarily based upon the EdgeSimPy Souza et al. (2023) framework which is a Python-based modeling and simulation tool for edge computing resource management policies. To ensure the validity and reliability of our findings, I have employed a systematic evaluation methodology. The decisions regarding the methodology are intricately connected to the insights gained from related work. Previous research by Guo et al. (2020) and Chen et al. (2019) influenced the adoption of machine learning-based approaches, particularly the Q-learning algorithm as a key component of our evaluation strategy.
In my research, I utilize network switches to facilitate connections, mainly conceptualizing task migrations as network flows. The Max-Min fairness algorithm Hosaagrahara and Sethu (2008) is applied in these switches to schedule bandwidth, ensuring fair distribution for effective task migration within the edge computing setup.

I employed an Intel HexaCore i7-9750H processor, utilizing the Skylake architecture Doweck et al. (2017), paired with an NVIDIA 1050 Ti GPU featuring 16 GB of RAM. The simulations were conducted on the Linux Operating System within the Google Colab environment. To implement the deep reinforcement learning algorithms, I leveraged

the PyTorch library. The hyperparameters and their respective values utilized in these experiments are detailed in Table 1 below. I utilized the following hyperparameters and their respective values in the experiments:

Hyperparameter	**Hyperparameter Value**
α (Rate at which algorithm learns)	0.07
ϵ (Exploration Factor)	1
γ (Future Reward Weight Factor)	0.9
ϵ Decay (rate at which ϵ decreases)	0.995
Dropout Probability in Neural-Network	0.5

Table 1: Hyperparameters and Values

I have considered a scenario where I have 6 edge servers with varying CPU, memory, and disk capacities. The specifications are outlined in Table 2.

Server	**CPU**	**Memory**	**Disk**
Edge-Server 1	8	16384	131072
Edge-Server 2	8	16384	131072
Edge-Server 3	8	8192	131072
Edge-Server 4	8	8192	131072
Edge-Server 5	12	16384	131072
Edge-Server 6	12	16384	131072

Table 2: Server Specifications

3.1 Research Procedure

- Simulation Setup
  The simulation setup involves defining the characteristics of edge servers and network switches, which are crucial components in our research. These servers are virtually placed with varying capacities and locations forming the infrastructure for our task offloading experiments.

- Q-learning Algorithm Implementation
  The Q-learning algorithm has been integrated into our simulation as a dynamic decision-making mechanism for task offloading. This algorithm adapts to changing conditions, considering factors such as CPU, memory and disk availability of edge servers. The EnsembleDQNAgent, a reinforcement learning approach is employed to optimize service provisioning decisions.

- Service and Scenario Configuration
  Services have been defined that need to be provisioned on the edge servers, simulating real-world scenarios where mobile applications or tasks are offloaded to the edge. The simulation captures the dynamic nature of service demands, incorporating factors such as CPU demand, memory demand and power consumption.

- Data Collection
  During the simulation run, detailed metrics related to edge servers, including power

consumption, CPU usage and memory usage are collected using a custom collect method. This method records key performance indicators for each edge server providing granular data for analysis.

- Equipment Used
  The simulation is conducted on the Google Colab platform Bisong and Bisong (2019), leveraging its computational resources for efficient execution of the EdgeSimPy framework and the Q-learning algorithm. The choice of this platform ensures a standardized and scalable environment for the conducted experiments.

- Statistical Techniques
  The raw data collected from the simulation runs are subjected to statistical analysis. Pandas McKinney et al. (2011), a powerful data manipulation and analysis library in Python is used to organize and preprocess the data.

- Results and Discussion
  The final results are presented in a structured format showcasing the impact of the Q-learning algorithm on task offloading efficiency, power consumption and overall system performance. These results are critically analyzed, drawing comparisons with other state of the art algorithms to benchmark the proposed solution

4 Design Specification

In this section I would describe the architecture of the EdgeSimPy framework which has been used for the simulation of the edge-cloud computing scenario and then I would discuss about the details of the proposed algorithm- Ensemble Q-learning algorithm with Replay and buffer mechanism.

4.1 EdgeSimPy Architecture

EdgeSimPy Souza et al. (2023) is designed to support the modeling and evaluation of computational offloading strategies in Edge Computing environments. It consists of following key components as shown in the figure 1
Base Stations: These provide network connectivity for mobile devices in their coverage areas, ensuring equitable connectivity within each cell.
Network Switches: They establish wired connections between base stations and edge servers. Task migrations are represented as network flows, and bandwidth scheduling is done using the Max-Min fairness algorithm Hosaagrahara and Sethu (2008).
Resource Modeling: Edge servers host services and their power consumption is modeled based on CPU, RAM, and hard disk utilization. Different power consumption models (Linear, Quadratic, Cubic) are applied depending on utilization parameters.
Users: Users consume services hosted on edge servers. Mobility models define user movement and changes in base station access for service consumption.
Task Modeling: Tasks represent applications or services with specific resource requirements, such as CPU and memory demands. Allocating tasks to edge servers influences power consumption and resource utilization.
AI Model Modeling: The AI module integrates Reinforcement Learning (RL) based task migration algorithms. The documentation discusses effective strategies for task allocation and migration.

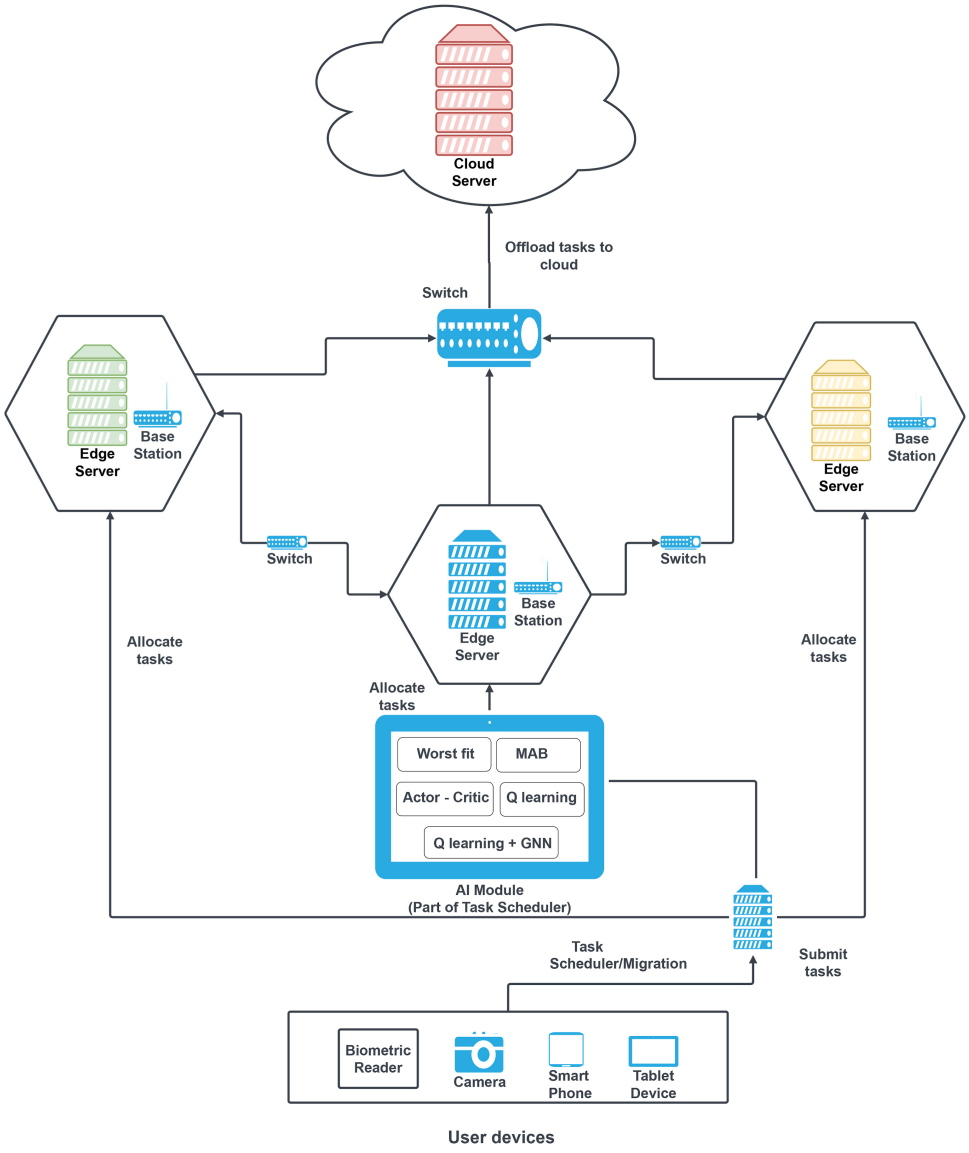

Figure 1: EdgeSimPy framework architecture

4.2 Ensemble Replay Buffer Q-Learning Algorithm

The main contribution of this research lies in the optimization of the Q-Learning algorithm by incorporating replay and buffer mechanism. The replay buffer acts as a reservoir of past experiences, allowing the agent to revisit and learn from a diverse set of historical situations. This addition brings about a paradigm shift in the way the agent processes information, contributing to more optimized decision-making in dynamic and resource-constrained environments, characteristic of scenarios encountered in edge computing.
As edge devices contend with fluctuating workloads, varying resource capacities and dynamic user interactions, the replay buffer provides a mechanism to break the sequential correlation of experiences. This, in turn mitigates the impact of similar encounters on the learning process,fostering stability and preventing the algorithm from becoming biased towards recent experiences.

The base equation for the Q-learning algorithm is the Q-value update rule. This rule governs how the Q-values for state-action pairs are updated based on the observed rewards and the estimated future rewards.The Q-value update equation is as follows:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \cdot (r + \gamma \cdot \max Q(s',\cdot) - Q(s,a)) \tag{1}$$

where:

- Q(s,a) is the Q-value for the state-action pair.
- s is the current state.
- a is the chosen action.
- r is the received reward.
- s' is the next state after taking action a
- is the learning rate (a hyper-parameter that controls the step size of updates).
- is the discount factor (a hyper-parameter that controls the importance of future rewards).

Now, let's break down how this equation affects the entire Q-learning process:

1. **Step 1: Initialization** Initialize Q-values for all state-action pairs, as well as the replay buffer.
2. **Step 2: Action Selection** The agent selects an action for the current state based on the current Q-values, following the exploration-exploitation strategy.
3. **Step 3: Observation and Reward** The agent takes the chosen action, observes the new state and receives a reward from the environment.
4. **Step 4: Updating Q-values (Original Equation)** Without the replay buffer, the agent updates the Q-value using the original equation (1) as stated above.
5. **Step 5: Replay Buffer Addition** With the replay buffer, the agent stores the experience tuple (s,a,r,s) in the replay buffer.

6. **Step 6: Replaying Experiences** Periodically, the agent samples experiences from the replay buffer. For each sampled experience, the agent updates the Q-value using the modified equation:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \cdot (r + \gamma \cdot \max Q(s',\cdot) - Q(s,a)) \tag{2}$$

7. **Step 7: Repeat** The agent repeats the process for multiple episodes, updating Q-values based on both new experiences and replayed experiences.

The following figure describes the architecture of Ensemble Q-Learning Algorithm:

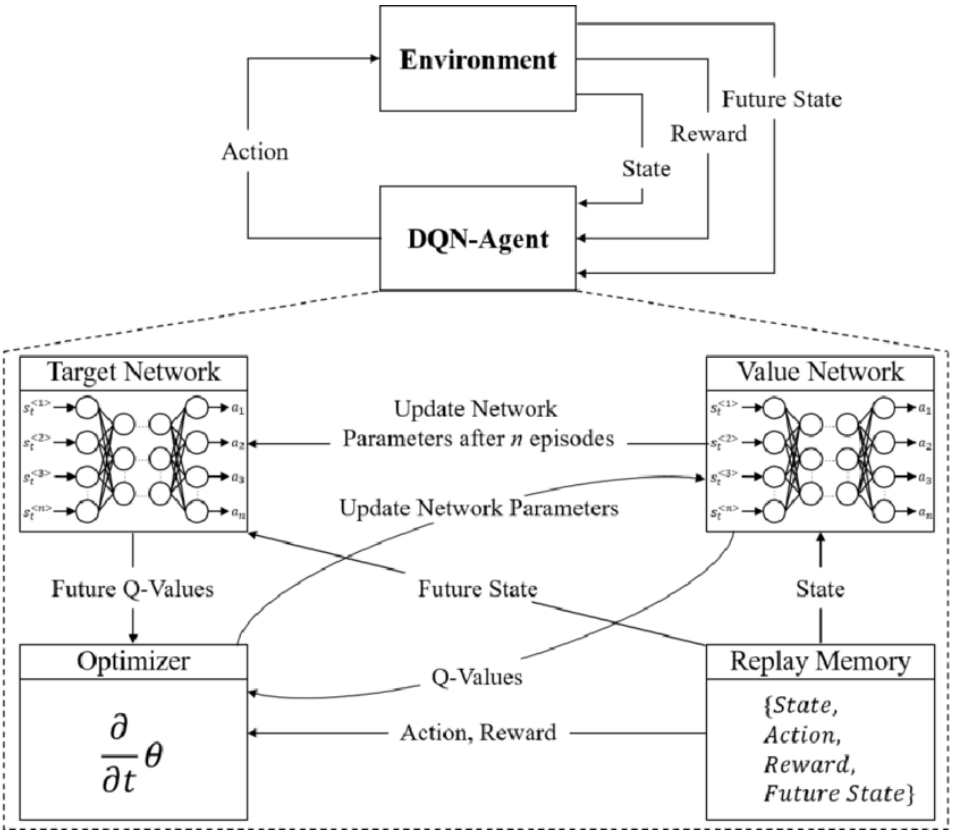

Figure 2: Ensemble Q-Learning Algorithm Architecture

In my research, I introduced a novel approach to enhance the decision-making capabilities of a group of Q-learning agents through the creation of an Ensemble Deep Q-Network (DQN). This ensemble comprises multiple Q-learning agents, each initialized with distinct settings or exploration policies. Additionally, a pivotal component of my methodology involves the incorporation of a replay buffer, a memory system that stores past experiences for efficient learning. Let me break down the key elements of my approach in simpler terms:

1. Ensemble Creation:

    - Traditional Twist: Instead of a single Q-learning agent, I introduced a team of them, forming what I call an Ensemble. Each agent has its unique starting point or way of exploring the environment.

2. Replay Buffer:

    - Memory Boost: Imagine a smart memory system that stores snapshots of past experiences—what the agents saw, the actions they took and the rewards they received. This is the replay buffer and it's a crucial tool for more effective learning.

3. Agent Interaction with Environment:

    - Seeing and Deciding: Each agent keeps an eye on what is happening in the environment and makes decisions based on what it observes. They're like individual decision-makers in a team.

4. Replay Buffer Management:

    - Learning from History: After agents interact with the environment, they do not forget what happened. They store these experiences (what they saw, what they did, what happened next) in the replay buffer.
    - Random Playbacks: Instead of learning only from the latest happenings, agents occasionally replay past experiences. It is like going back in time to learn from different situations.

5. Ensemble Learning:

    - Getting Smarter: Each agent learns and gets smarter by updating its decision-making rules using the experiences from the replay buffer. The goal is to make their predictions match the real rewards they receive.
    - Teamwork in Decisions: When it is time to make decisions, the agents in the ensemble might team up. They could vote on actions or average their thoughts to arrive at a group decision.

6. Replay Buffer Benefits:

    - Rewind for Wisdom: The replay buffer allows agents to rewind and learn from a diverse set of experiences, not just consecutive ones. It is like learning from a variety of real-life situations.
    - Learning Efficiency: By reusing experiences, agents become more efficient learners. This makes the learning process more stable and robust, ensuring they make better decisions even in tricky situations.

5 Implementation

This section provides the step by step implementation of the simulation showing how the computational offloading is handled by the edge servers mentioned above in the methodology section using the proposed **Ensemble Q-Learning algorithm** as shown below:

1. **Import Libraries and Classes**: In this initial step, the Python programming language is used along with various libraries such as NumPy, Pandas, Matplotlib and the DQN module. The DQN module contains the DQNAgent class. This class represents the Q-network, which is a neural network used to approximate the Q-values for different state-action pairs in a reinforcement learning environment.

2. **Define a Custom Collection Method**: A custom collection method is defined, which is crucial for monitoring and logging server metrics during the simulation. The metrics include "Instance ID", "Power Consumption", "CPU Demand" and "Memory Demand." This method is designed to be utilized with instances of the EdgeServer class. It provides crucial insights into the simulation dynamics and the impact of task offloading on server metrics.

3. **Define a Replay Buffer Class**: The ReplayBuffer class is introduced to manage a buffer of experiences. This buffer stores information about the state, action, reward and next state during the simulation. The class includes methods to add experiences and sample batches for training.It breaks the temporal correlation in the data to allow agents to learn from a diverse set of experiences rather than consecutive transitions.

4. **Define an EnsembleDQNAgent Class**: After defining the buffer class, the EnsembleDQNAgent class is created to handle multiple instances of DQNAgent. This ensemble approach provides diversity in decision-making during the simulation. Methods include action selection, updating agents, exploration and decay of exploration parameters.

5. **Initialize Global Variables**: Global variables are declared for the agent and reward tracking. These variables serve as a global context for tracking and storing important information throughout the execution of the simulation.

6. **Initialize a Replay Buffer and EnsembleDQNAgent**: Instances of the ReplayBuffer and EnsembleDQNAgent are created. The buffer is used to store experiences and the ensemble of DQN agents is prepared for the simulation.

7. **Define Q-learning Algorithm for Ensemble**: A Q-learning algorithm specific to the ensemble of agents is defined. This algorithm likely involves iterating through services, selecting actions, updating states and managing the replay buffer. It forms the core logic for decision-making during the simulation.

8. **Define a Stopping Criterion for the Simulator**: In this step a stopping criterion is established for the simulator. This criterion, based on the number of simulation steps, determines when the simulation should conclude. This is essential for controlling the duration of the simulation.

9. **Initialize and Run the Simulator**: The Simulator class is utilized for running the simulation. Specific configurations such as tick duration, tick unit, stopping criterion and Q-Learning algorithm are set. The simulator is then initialized and the simulation is executed.

10. **Collect Simulation Logs and Visualize Results**: Simulation logs are collected in a Pandas DataFrame to perform further analysis. The logs show the information about each EdgeServer instance such as time steps, power consumption,CPU

usage and memory usage. Matplotlib library has been used to visualize the power consumption trends over time for each EdgeServer.

The above steps are repeated for 3 different algorithms namely Random Migration algorithm, Q-Learning algorithm, GCN-Q Learning algorithm (Graph Convolutional Network) to compare them with the proposed Ensemble Q-Learning algorithm on the basis of the mainly 3 parameters:

1. Power consumption
2. CPU Usage
3. Memory usage

The results of the simulations for the above metrics are discussed in the next section to comment on the suitability of the proposed algorithm.

6 Evaluation

This section depicts and discusses the critical analysis of the results of the above simulations for the following algorithms used for Task Offloading to 6 EdgeServers:

1. Random Migration
2. Q-Learning
3. GCN based Q-Learning algorithm
4. Ensemble Q-Learning with Replay and Buffer

6.1 Power Consumption of different algorithms

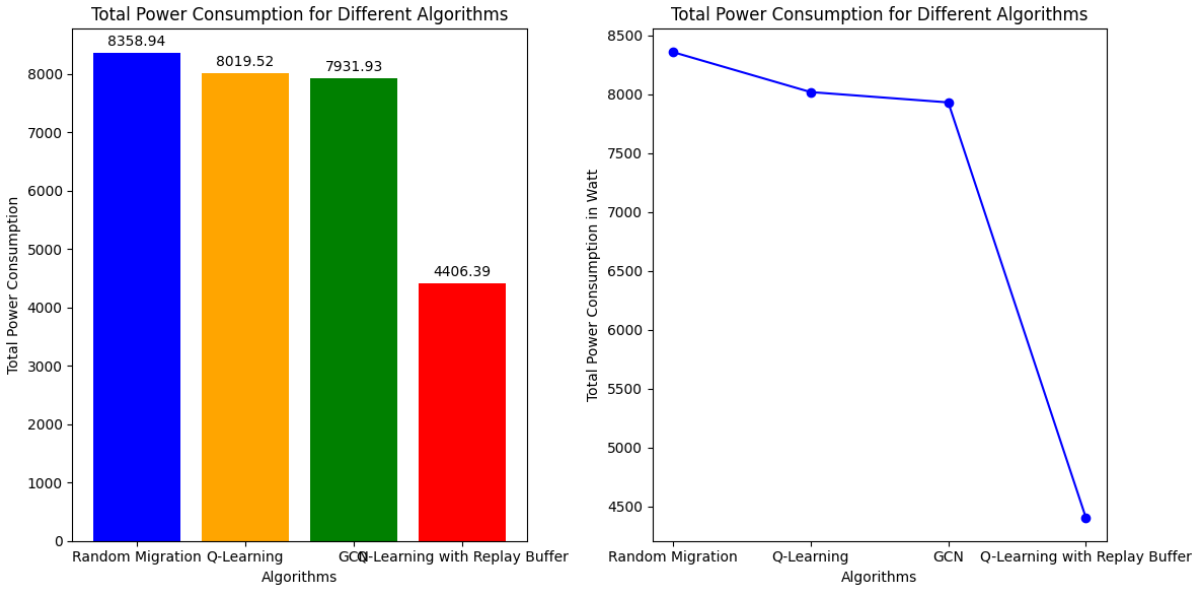

Figure 3: Total Power Consumption of the algorithms

1. **Random Migration (RM)**: The total power consumption for this algorithm is 8358.94 Watts. Random Migration is a non-optimized strategy where tasks are assigned to Edge Servers randomly. Higher power consumption indicates inefficiency in resource utilization as tasks are be placed on servers without considering their current load or capacity.

2. **Q-Learning**: The total power consumption for Q-Learning is 8019.52 Watts. Q-Learning is a reinforcement learning algorithm that aims to optimize task assignment based on learned policies. The power consumption is still high.

3. **GCN (Graph Convolutional Network) based Q-Learning**: The total power consumption for GCN is 7931.93 Watts. GCN utilizes a graph-based approach to optimize task offloading decisions. The power consumption is comparable to Q-Learning.

4. **Q-Learning with Replay Buffer (QL with RB)**: The total power consumption for my proposed algorithm is 4406.39 Watts. The addition of a replay buffer led to more efficient decision-making which has resulted in lower power consumption.

Leveraging widely recognized algorithms as benchmarks in the field of edge computing provides a robust method for evaluating the proposed solution. The algorithms chosen for this comparative analysis bear distinctive significance within the field. Random Migration (RM) acts as a basic benchmark, helping us understand how the proposed algorithm compares to random task allocation strategies. Q-Learning (QL), a commonly used reinforcement learning algorithm, sets a standard for evaluating my proposed Q-Learning with Replay Buffer against the traditional Q-Learning approach. Graph Convolutional Network (GCN)on he other hand follow graph-based learning which represents a cutting-edge machine learning method allowing us to assess the proposed solution against a sophisticated graph-based algorithm widely used in edge computing. The significance of this benchmarking lies in providing an unbiased measure of our proposed algorithm's performance, enabling comparisons across different approaches and highlighting potential advancements.**The proposed algorithm demonstrates a 44% reduction in power consumption, positioning it as an effective and efficient solution in comparison to the established benchmarks. This showcases its potential to address real-world challenges in edge computing task offloading with a notable reduction in power usage.**

6.2 CPU Utilization Rate

Each algorithm's average CPU utilization is a critical metric in evaluating its efficiency in managing computational resources. In the context of edge computing and task offloading, these values provide insights into how well each algorithm optimizes the utilization of Edge Servers.

The Random Migration (RM) algorithm, with an average CPU utilization of approximately 1.95, showcases a moderately loaded server environment. Since RM involves random task assignments, the CPU utilization reflects a non-optimized strategy where some servers may operate closer to their capacity.

Q-Learning (QL), with an average CPU utilization of around 1.67, demonstrates more informed decision-making compared to RM. The lower average CPU utilization suggests

that Q-Learning achieves a better balance in distributing tasks among Edge Servers, indicating a more optimized approach to task offloading.

Graph Convolutional Network (GCN) Chiang et al. (2019), leveraging a graph-based methodology, exhibits an average CPU utilization of approximately 1.56. This relatively low value indicates that GCN effectively distributes tasks, preventing servers from experiencing heavy loads. The graph-based approach seems to contribute to the efficient use of CPU resources.

Q-Learning with Replay Buffer (QL with RB) introduces a slightly higher average CPU utilization of approximately 1.72. The addition of a replay buffer enhances decision-making, leading to a more balanced utilization of CPU resources. This suggests that incorporating replay mechanisms in reinforcement learning algorithms can contribute to improved resource management.

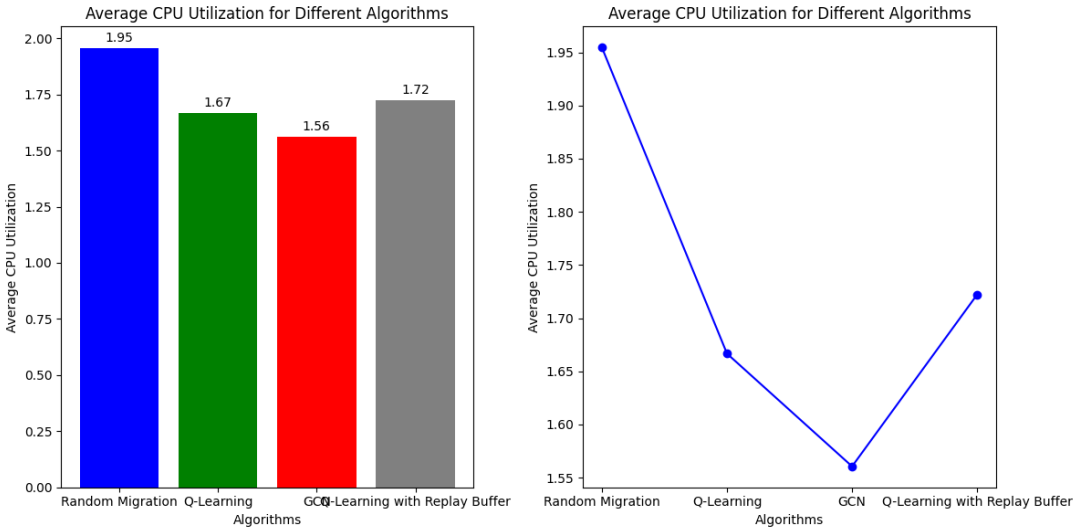

Figure 4: Average CPU Utilization of the algorithms

6.3 Average Memory Consumption of each Algorithm

In evaluating the memory consumption of various algorithms:

- Random Migration (RM): In my observation, RM tends to use a considerable amount of memory. This suggests that the algorithm might not be making the most efficient use of memory when assigning tasks.
- Q-Learning (QL): I noticed that QL is quite efficient in handling memory. This indicates that the reinforcement learning approach it employs helps in reducing the overall memory usage.
- Graph Convolutional Network (GCN): In my analysis, GCN stands out for its highly efficient use of memory. It seems to distribute tasks in a way that requires less memory making it a promising choice for memory-conscious scenarios.
- Q-Learning with Replay Buffer (QL with RB): Upon observation, QL with RB shows a slightly higher memory consumption compared to QL. The addition of a

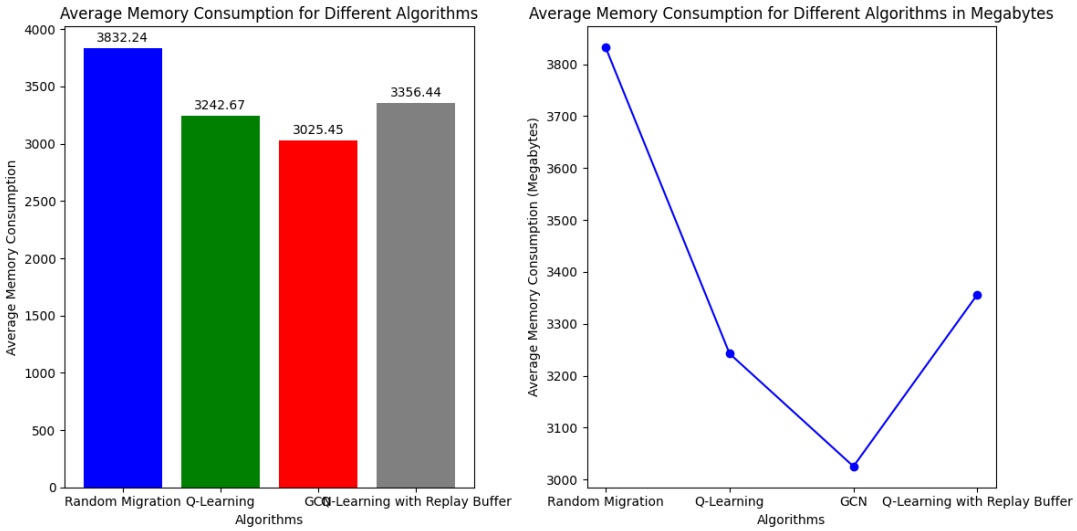

Figure 5: Average Memory Consumption of the algorithms

replay buffer may have contributed to a more complex decision-making process, resulting in a small increase in memory usage.

This detailed examination provides insights into how each algorithm handles memory resources. It helps in understanding their efficiency in minimizing memory usage, a crucial factor for resource optimization in edge computing.

6.4 Discussion

In the comprehensive analysis of the experimental outcomes, examination of algorithmic performance in terms of task offloading to six Edge Servers is imperative. The scrutiny extends to pivotal metrics such as power consumption, CPU utilization rate and average memory consumption, providing nuanced insights into the intricate operational dynamics of each algorithm.

Throughout the experiments, the algorithms under evaluation—Random Migration (RM), Q-Learning (QL), GCN based Q-Learning and Ensemble Q-Learning with Replay and Buffer (QL with RB) revealed distinctive characteristics influencing their efficacy in edge computing scenarios.

In terms of power consumption, RM emerged as a non-optimized strategy, demonstrating higher power consumption due to its random task assignment. QL, despite its reinforcement learning foundation, exhibited considerable power usage, suggesting areas for potential optimization. The GCN-based approach showcased competitive power consumption, emphasizing its efficiency in resource utilization. Notably, the proposed Ensemble Q-Learning with Replay and Buffer displayed a remarkable 44% reduction in power consumption, signifying its potential as an efficient solution in real-world edge computing environments.

The examination of CPU utilization rates provided insights into how each algorithm optimized the utilization of Edge Servers. RM, with its random task assignments, reflected a moderately loaded server environment. QL demonstrated more informed decision-making, achieving a balanced distribution of tasks among Edge Servers. The graph-based methodology of GCN contributed to efficient CPU resource utilization, preventing server

overload. The proposed algorithm:Ensemble Q-Learning with Replay and Buffer introduced a slightly higher CPU utilization, indicating the enhanced decision-making brought about by the replay buffer.

Memory consumption analysis further delineated the efficiency of each algorithm. RM exhibited notable memory consumption, implying potential inefficiencies in task assignment. QL showcased efficient memory usage, highlighting the effectiveness of its reinforcement learning approach. GCN stood out for highly efficient memory usage, positioning itself as an excellent choice for scenarios where memory optimization is crucial. Ensemble Q-Learning with Replay and Buffer displayed slightly higher memory consumption, suggesting added complexity introduced by the replay buffer.
In light of the experimental outcomes and the critical analysis conducted on algorithmic performance, several modifications and improvements to the design could enhance the robustness and efficacy of the results such as :-

1. For the Q-Learning algorithm, a detailed exploration of hyperparameters such as learning rate, discount factor, and exploration-exploitation trade-off could be conducted. Fine-tuning these parameters might lead to improved convergence and better decision-making, ultimately reducing power consumption.

2. Replay Buffer Size in Ensemble Learning: Given the significant reduction in power consumption achieved by Ensemble Q-Learning with Replay and Buffer, experimenting with different replay buffer sizes could be beneficial. Optimizing the size of the replay buffer might strike a balance between learning efficiency and memory consumption.

3. For reinforcement learning algorithms, including an adaptive mechanism for exploration-exploitation could be valuable. Dynamic adjustment of the exploration parameter based on the system's performance or workload could lead to more efficient decision-making.

7 Conclusion and Future Work

In tackling the research question about optimizing task offloading strategies in edge computing environments my goals revolved around implementing and assessing algorithms to understand their impact on resource utilization and power consumption. The work involved developing and simulating algorithms like Random Migration, Q Learning, Graph Convolutional Network (GCN) based Q Learning and Ensemble Q Learning with Replay and Buffer.

The results demonstrate a decrease in power consumption by 44%, when utilizing the proposed Ensemble Q Learning algorithm that incorporates a replay and buffer mechanism. This achievement aligns with the objective of improving the efficiency of task offloading mechanisms in edge computing. The key findings highlight the effectiveness of this algorithm in striking a balance between resource utilization and power consumption surpassing established benchmarks.

The implications of this research are substantial as they suggest that the proposed algorithm could make a contribution to sustainable and resource efficient edge computing systems. By reducing power consumption this algorithm addresses the increasing demand for energy solutions, as edge computing infrastructures continue to expand.

However it is important to acknowledge limitations in this research. While the simulation environment is representative it may not capture all intricacies found in real world edge computing scenarios. The outcomes depend on settings and conditions highlighting a level of sensitivity.

Looking forward I prioritize refining and validating the proposed algorithm, in real world scenarios for work. It would be beneficial to integrate workloads considering dynamic network conditions and explore how the algorithm can adapt to evolving edge computing architectures to make the research more applicable.

For research proposals I also look forward to exploring algorithms or multi agent systems that combine different approaches to create even stronger task offloading strategies. Additionally studying the algorithms performance under security and privacy constraints could lead to effective investigations.

When it comes to commercialization carefully considering the application of the algorithm in industry edge computing deployments such as IoT networks or smart cities is important. Collaborating with industry partners for real world implementations can validate the usefulness of the algorithm.

In conclusion while the research has successfully shown a reduction in power consumption through the proposed algorithm it is crucial to acknowledge that edge computing environments come with limitations and complexities. Future efforts should focus on improving adaptability, realism and applicability of the algorithm paving the way, for efficient edge computing solutions.

References

Bisong, E. and Bisong, E. (2019). Google colaboratory, *Building machine learning and deep learning models on google cloud platform: a comprehensive guide for beginners* pp. 59–64.

Cao, K., Liu, Y., Meng, G. and Sun, Q. (2020). An overview on edge computing research, *IEEE access* **8**: 85714–85728.

Chen, Y., Zhang, N., Zhang, Y., Chen, X., Wu, W. and Shen, X. (2019). Energy efficient dynamic offloading in mobile edge computing for internet of things, *IEEE Transactions on Cloud Computing* **9**(3): 1050–1060.

Chiang, W.-L., Liu, X., Si, S., Li, Y., Bengio, S. and Hsieh, C.-J. (2019). Clustergcn: An efficient algorithm for training deep and large graph convolutional networks, *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 257–266.

Dab, B., Aitsaadi, N. and Langar, R. (2019). Q-learning algorithm for joint computation offloading and resource allocation in edge cloud, *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pp. 45–52.

Doweck, J., Kao, W.-F., Lu, A. K.-y., Mandelblat, J., Rahatekar, A., Rappoport, L., Rotem, E., Yasin, A. and Yoaz, A. (2017). Inside 6th-generation intel core: New microarchitecture code-named skylake, *IEEE Micro* **37**(2): 52–62.

Guo, H., Liu, J. and Lv, J. (2020). Toward intelligent task offloading at the edge, *IEEE Network* **34**(2): 128–134.

Habibi, P., Farhoudi, M., Kazemian, S., Khorsandi, S. and Leon-Garcia, A. (2020). Fog computing: A comprehensive architectural survey, *IEEE Access* **8**: 69105–69133.

Hosaagrahara, M. and Sethu, H. (2008). Max-min fair scheduling in input-queued switches, *IEEE Transactions on Parallel and Distributed Systems* **19**(4): 462–475.

Islam, A., Debnath, A., Ghose, M. and Chakraborty, S. (2021). A survey on task offloading in multi-access edge computing, *Journal of Systems Architecture* **118**: 102225.

Jiang, C., Cheng, X., Gao, H., Zhou, X. and Wan, J. (2019). Toward computation offloading in edge computing: A survey, *IEEE Access* **7**: 131543–131558.

McKinney, W. et al. (2011). pandas: a foundational python library for data analysis and statistics, *Python for high performance and scientific computing* **14**(9): 1–9.

Montoya, O. D., Gil-González, W. J. and Grisales-Noreña, L. F. (2020). An exact minlp model for optimal location and sizing of dgs in distribution networks: A general algebraic modeling system approach, *Ain Shams Engineering Journal* **11**: 409–418.
URL: *https://api.semanticscholar.org/CorpusID:209068690*

Pham, Q.-V., Fang, F., Ha, V. N., Piran, M. J., Le, M., Le, L. B., Hwang, W.-J. and Ding, Z. (2020). A survey of multi-access edge computing in 5g and beyond: Fundamentals, technology integration, and state-of-the-art, *IEEE access* **8**: 116974–117017.

Souza, P. S., Ferreto, T. and Calheiros, R. N. (2023). Edgesimpy: Python-based modeling and simulation of edge computing resource management policies, *Future Generation Computer Systems* .