Deep Learning Method for Task offloading in a Fog-Cloud environment

Doha KHTATBA
ETIS, UMR 8051 CY Cergy Paris University, ENSEA, CNRS F-95000
doha.khtatba@ensea.fr

Sonia YASSA
ETIS, UMR 8051 CY Cergy Paris University, ENSEA, CNRS F-95000
sonia.yassa@cyu.fr

July 27, 2023

Abstract

As the demand for computation-intensive applications grows, existing task offloading algorithms struggle to effectively optimize Quality of Service (QoS) metrics in a heterogeneous Fog-Cloud environment. This paper addresses the challenges of latency, energy efficiency, and cost in such an environment by proposing a deep learning solution for multi-objective optimization of total QoS metrics. The solution is a modified version of Deep Q-learning with Batch Learning and Dropout layers to prevent overfitting. The simulation results show the efficiency of our algorithm. A comparison with a well-known metaheuristic — the Genetic Algorithm is performed. The results highlight the advantages and limitations of our approach.

Keywords— task offloading, fog, cloud, optimization, deep learning

1 Introduction

1.1 Context and issues

The increasing demand for computation-intensive applications such as interactive gaming, augmented reality and the growth of IoT has led to a need for more powerful computational resources than the local devices can provide. Task offloading is the process of sending data from a user device to a Fog server or a Cloud server for processing, typically when the device is not able to handle the processing itself. This can be done continuously or as needed, based on various factors [1].

While Cloud computing is a popular solution, as it offers numerous advantages such as on-demand access, resource pooling, self-service provisioning, device and location independence, scalability, and the efficient use of virtualization technology [2]. It also has limitations in terms of latency, traffic hot-spots, unstable and intermittent network connectivity, costly and lossy backhaul networks, and bandwidth issues due to its centralized nature, which significantly impact user experience and hinder real-time applications.

Fog computing, introduced as an extension of cloud computing to the edge of the network, addresses these limitations by bringing the computations closer to the end users [3], [4]. By installing localized computing facilities at the user's premises, processing workloads and services locally on Fog devices, Fog computing provides low-latency and high-speed services, supporting applications that do not suit the cloud [5]. Typical applications combined with Fog computing are mostly IoT related, cache networks, and immersive media services [6].

However, there are challenges in providing low-latency and energy-efficient computations in a heterogeneous Fog-Cloud environment. These challenges include effective resource allocation, managing the trade-offs between latency, energy consumption, and monetary cost, as well as dealing with the dynamic nature of resource availability and task requirements. The existing task offloading algorithms do not effectively address the optimization of the Quality of Service (QoS) metrics of such applications and the constraints of such a problem.

In this paper, we identify the challenges associated with task offloading in Fog-Cloud environments and present our research goals, which include developing a solution that optimizes QoS metrics, balance trade-offs, and ensure efficient resource utilization.

1.2 Goals

The goals of this research are to develop a task offloading solution specifically designed for Fog-Cloud environments, to optimize total QoS metrics of the overall tasks, and to balance the trade-offs between latency, energy consumption and monetary cost in task offloading decisions. We also aim to ensure efficient utilization of fog and cloud resources, and to adapt our solution to the heterogeneous nature of Fog-Cloud environments, including varying resource availability and task requirements.

1.3 Contributions

In this paper, we adapt a Deep Q-learning algorithm designed specifically for Fog-Cloud environments. Our algorithm aims to optimize the total QoS metrics by making the best offloading decisions.

1.4 Paper organisation

The paper is organized as follows — Section 2 discusses existing algorithms and their limitations to our problem. We then describe our system model and formulate our multi-objective optimization problem in Section 3. Section 4 describes in depth our proposed approach. Section 5 showcase our solution's performance while being compared to the genetic algorithm. Finally, Section 6 gives a summary on our paper and future improvement that need to be implemented in the future.

2 Related work

The literature on task offloading, particularly in Fog-Cloud environments, is extensive and continually growing. In this section, we review the key research studies that have influenced the development of our deep learning solution for multi-objective optimization of total QoS metrics in such environments. Table **??** summarizes the techniques, the considered QoS, and the objectives of the studied related works.

- **Task offloading in Heterogeneous Environments**

Several studies have tackled the task offloading problem in heterogeneous environments, focusing on the optimization of either one or several QoS metrics. Kumar and Lu [7] discussed the potential for energy minimization that can be achieved by offloading computation to the cloud for mobile users. Zhang et al. [1] proposed a new task offloading in edge computing that aims to balance delay and energy consumption. Xia et al. [8] designed an objective function that would minimize the average latency of tasks offloaded to fog nodes in the fog layer.

- **Heuristics for Task Offloading**

Heuristic approaches have been employed to optimize various QoS metrics in task offloading. These techniques typically focus on optimizing a single QoS metrics, such as latency, energy efficiency or cost. Ait Salaht et al. [9] proposed a service offloading strating based on a mutli-criteria decision-making algorithm, which optimized multiple QoS metrics but lacked the flexibility of more advanced optimization techniques. While heuristics can provide satisfactory solutions, their limitations lie in the inability to adapt and optimize complex problems effectively.

- **Metaheuristics for Task Offloading**

Metaheuristic approaches, such as Genetic Algorithm, have been widely used for task offloading optimization. Mebrek et al. [10] put forward a GA-based approach for optimizing task offloading in a fog infrastructure by minimizing the energy consumption while still meeting the delay requirements for each task.

- **Deep Learning for Task Offloading**

Recently, deep learning techniques have been explored for optimizing task offloading. Liu et al. [11] address the challenge of selecting the optimal edge server for offloading computationally intensive applications in Mobile Edge Computing (MEC) enviornlents, considering the limited coverage of edge servers and frequent user movement. They modelled the problem of continuous server selection as a Markov Decision Process (MDP) and propose a Deep Reinforcement Learning (DRL) based algorithm to learn the selection policy based on observed performance of past server selections. The algorithm uses a Long Short-Term Memory (LSTM) based neural network to encode historical information. The practical implications of the paper include providing a solution to optimize resource allocation and improve the quality of service for MEC users. The algorithm is evaluated using trace-driven simulations, and the results demonstrate that it has the lowest overall cost compared to existing solutions. However, limitations include reliance on simulations, assumptions about system states, and lack of consideration for energy consumption and scalability.
Zhang et al. [1] developed a deep learning-based algorithm for energy-efficient task offloading, which optimized energy efficiency and system stability. However, these studies focused primarily on optimizing one or two QoS metrics, leaving room for improvement in addressing multiple QoS metrics simultaneously. Our goal in this paper is to develop a deep learning-based solution, incorporating elements from Deep Q-learning, Batch Learning, and Dropout layers, A solution that will overcomes the related work limitations by optimizing the total QoS metrics, thus contributing to a novel approach to the field of task offloading.

- **Other methods for Task Offloading**

Du et al. [12] presents a joint optimization approach for computation offloading and resource allocation in mixed fog/cloud computing systems, aiming to minimize the maximal weighted cost of delay and energy consumption among all user equipment (UEs) while guaranteeing user fairness and maximum tolerable delay. The authors propose a suboptimal algorithm with low complexity, which employs semidefinite relaxation and randomization techniques to determine offloading decisions and fractional programming theory and Lagrangian dual decomposition to allocate resources. There are regardless some limitations in terms of the suboptimal nature of the algorithm, the simulation is performed on a single fog node and cloud server, and the results may not be generalizable to larger and more complex systems. Additionally, the paper assumes that the network conditions and user require-

ments are known a priori, which may not always be the case in real-world scenarios.

2.1 Discussions

Most existing task offloading strategies try to optimize a single objective of the QoS parameters. However, single objective optimization may reduce the performance and efficiency of the fog node or cloud data center. Today, most applications or tasks are generated from real-time devices, making it challenging to run them in a fog environment alone. The purpose of this study is to develop a multi-objective offloading strategy that would take into account the three major objectives of IoT applications: energy consumption, execution time, and latency, and offload them to a Fog-Cloud environment. A Fitness function based on weighted sum approach is used to evaluate the quality of the solution in terms of these multiple objectives.

3 System Model and Problem Formulation

3.1 System model

The system architecture of our approach is composed of three layers as shown in Figure 1. The first layer, called the physical layer, represents the end users environment where all IoT devices are located for each end user. The fog layer holds a set of interconnected fog nodes, forming the fog network. Each fog node is a local node that provides computing services through one virtual machine. The cloud layer is located above the Fog layer, and houses several cloud data centers with several virtual machines. Each fog and cloud node is characterized by its CPU rate in Million instructions per second (MIPS), memory, bandwidth, execution power, transmission power, geographical position as well as different costs — CPU usage cost, Memory usage cost and Bandwidth usage cost. The physical layer generates tasks that can either be processed by a nearby fog node, or sent to a cloud data center. It depends on the characteristics of the task and the capabilities of the fog node.

3.2 Task Model

We have T set of tasks that need to be offloaded. Each $T_i \in T$ is part of an application generated by the ith IoT object. A task is characterized by a computing size S_i which represents the workload of the task (i.e. the number of instruction lines), and a size S_i which represents the amount of data to be stored for the execution of the task.
The location of each connected IoT device is determined by its longitude and latitude values.

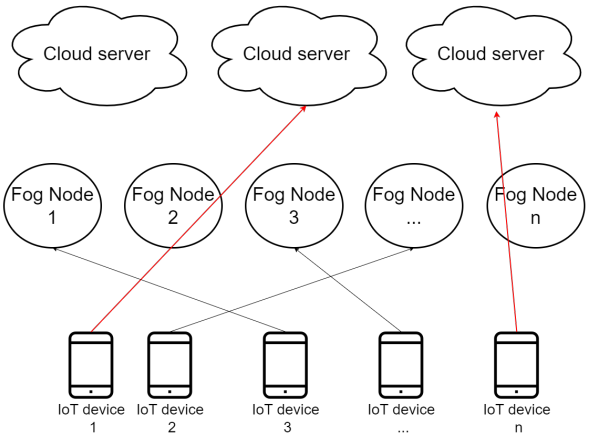

Figure 1: System Model

3.2.1 QoS metrics

One approach to optimizing QoS metrics for task offloading is to consider the various metrics that researchers have separately treated for the problem. In our case, we choose to optimize the following metrics:

- Execution Time

The execution time of a task T_i is measured through the equation 1, where S_i is the size/workload of the task and VM_{Mips} is the number of instructions per second (Mips) of each Virtual Machine i.e. number of CPU cycles to complete the execution of the task.

$$ET(T_i) = \frac{S_i}{VM_{Mips}} \tag{1}$$

- Latency

The total latency is measured through the equation (2) — the sum of the transmission time (equation 3 with the execution time (equation 1) for each task T_i of the system, most formulas down below are found in [17].

$$Latency = \sum_{i=1}^{T} TT(T_i) + ET(T_i) \tag{2}$$

with:

$$TT(T_i) = \frac{Size_{Input}(T_i)}{B_{VM}} \frac{(MB)}{(MB/s)} \tag{3}$$

  - $Size_{transf}$: Input size of the data that must be transferred to the fog/cloud
  - B_{VM} : Bandwidth of the Virtual Machine/server i.e. the amount of data that can be transferred per second

- Energy consumption

The total energy consumption is measured through the equation (4) — the sum of the execution energy (equation 5) and the transmission energy (equation 6)

$$E_{total} = \sum_{i=1}^{T} E_{exec}(T_i) + E_{transm}(T_i) \tag{4}$$

$$E_{exec} = ET_i * P_i \tag{5}$$

Table 1: Classification of related works

Work	Env.	Technique	QoS	App. type	Objective	Sim. Env.
[8]	Fog	Heuristic: Exh. Search, Naive Search	Latency	Smart Home	Min. latency, reduce exec. times and scalability	Simgrid
[10]	Hybrid	Meta-heuristic: Genetic Alg.	Energy, delay	Task	Min. energy and delay with multi-obj. optimization	Not mentioned
[13]	Hybrid	Energy Consum. Oriented Offloading Alg.	Energy	Task	Min. energy cost of task offloading	Matlab
[14]	Cloud Comp.	Neural Network Scheduling (NNS) Alg.	Makespan	Workflow	Min. makespan based on RL	Neural Networks
[15]	Hybrid	Energy Eff. Comp. Offloading Alg.	Energy, delay	Task	Min. energy consum. in 5G het. network	Not mentioned
[9]	Fog Comp.	Constraint Prog. Alg.	Delay, deploy. cost	DAG	Min. placement descr. effort and solving time	Choco-solver
[11]	MEC	DRL with LSTM	Comm. delay, comp. delay, switching cost	DAG	Address limited coverage and user movement in MEC	Not mentioned
[1]	Edge	Deep Reinforcement Learning	user delay	Task	improve user experience and resource utilization	Not mentioned
[16]	Fog Comp.	Deep Recurrent Q-network Alg.	Delay, model accuracy	Task	Solve joint task offloading and res. allocation	Python-Tensorflow
[12]	Hybrid	SDR, randomization, fractional prog., Lagrangian dual decompos.	Delay, ensuring consum.	Comp. Offloading, Res. Alloc.	Min. maximal weighted cost of QoS metrics	Matlab
Our work	Fog and Cloud	Batch Deep Q-learning with Dropout layers	Latency, Energy consum., Monetary Cost	Task	Balance optimization of three QoS metrics	Python 3.9 Tensorflow 2.0

– P_i : Execution power of Virtual Machine VM_i

$$E_{transm}(T_i) = TT(T_i) * P_s \tag{6}$$

– P_s : Used Power for sending the task to the fog nodes/cloud data centers

- Total Cost

The total cost is measured through the equation (7) — the sum of the execution time multiplied by the CPU cost, the memory required to execute each task multiplied by the memory cost of every node and finally the sum of the input output size of each task multiplied by the bandwidth of each node.

$$C_{total} = \sum_{i=1}^{T} ET_i * C_{CPU} + M_i * C_M + (S_i^i + S_i^O) * B_{VM} \tag{7}$$

– C_{CPU} : CPU cost of the VM executing task i

– M_i : Memory required to execute each task

– C_M : Memory cost of VM executing task i

– S_i^i : Input data size of each task i

– S_i^O : Output data size of each task i

3.3 Problem formulation

We formulate the task offloading problem as a multi-objective optimization problem where we need to minimize the presented conflicting Quality of Service metrics.

Objective: Minimize

$$\sum_{i=1}^{T} [\omega_1 Latency + \omega_2 E_{total} + \omega_3 C_{total}] \tag{8}$$

– $\omega_1, \omega_2, \omega_3$: weights representing the importance of QoS metrics, used to aggregate our multi-objective optimization problem where $\sum \omega_i = 1$

Subject to:

- **Resource constraints for mapping tasks:**

- Let x_{ij} be a binary decision variable, where $x_{ij} = 1$ if task T_i is offloaded to Virtual Machine (VM) j, and 0 otherwise

  – CPU constraint: The CPU requirement of task T_i must not exceed the available CPU capacity of VM_j

  $$\sum_{i=1}^{T} x_{ij} * CPU_{T_i} \leq CPU_{VM_j}, \quad \forall j \tag{9}$$

  – Memory constraint: The memory required for task T_i must not exceed the available memory capacity of VM_j

  $$\sum_{i=1}^{T} x_{ij} * M_i \leq M_{VM_j}, \quad \forall j \tag{10}$$

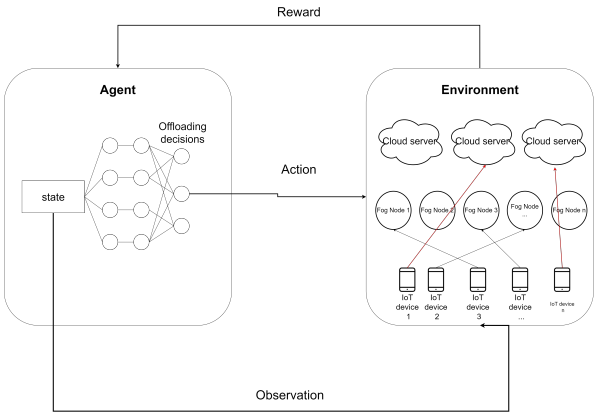

Figure 2: Solution Architecture

- Task Offloading Constraint:

$$\sum_{j=1}^{N} x_{ij} = 1, \quad \forall i \tag{11}$$

This constraint ensures that each task T_i is offloaded to exactly one VM

4 Proposed Approach: Batch Deep Q-Learning for Task Offloading [BDQL-TO]

In this work, we adapt a deep learning model based on deep reinforcement learning (DRL) that combines the principles of Deep Q-Learning (DQL) with batch learning [18] to handle the multi-objective optimization problem. The model takes into account all available task information at once to make offloading decisions. The input of the model is the concatenated TaskDetails and NodeDetails datasets [19], while the output is a matrix representing the offloading decisions for all tasks on all nodes.

Our BDQL-TO model, shown in Figure 2 consists of several layers, including an input layer, hidden layers, and an output layer. The input layer receives all the tasks and nodes information, while the hidden layers extract relevant features and learn the optimal offloading decisions. The output layer provides the final offloading decisions for all tasks on all nodes. Based on the offloading decision the agent outputs, it receives a reward from the environment, depending on how well the agent performed the decisions.

4.1 Deep Q-Learning

DQL is a reinforcement learning technique that uses deep neural networks to approximate the Q-function, which estimates the expected future rewards for each action taken in a given state. This technique enables the agent to learn a policy for making decisions that maximize the cumulative reward in the environment.

4.2 Batch Learning

The agent trains on a batch of experiences sampled from its memory, rather than updating its Q-function estimate based on single experiences. This approach can improve the stability and efficiency of the learning process by reducing the correlation between consecutive updates [18].

The agent uses Batch DQL to learn a policy that minimizes the total latency, energy consumption, and cost in the environment. The agent's neural network is trained to approximate the Q-function, which guides the agent in selecting the best actions given the current state. By iteratively improving its Q-function estimates through experience replay and batch learning, the agent converges to an optimal policy for allocating tasks to the fog and cloud nodes. The algorithm 1 showcases the environment initialization as well as the action selection process of the agent.

Algorithm 1 BDQL-TO for Task Offloading on Fog-Cloud

Initialize environment *env* with task data, fog nodes, and cloud nodes
Initialize agent with state size, action size, batch size, number of fog nodes, number of cloud nodes, and number of tasks
Initialize total rewards and episode lengths
for each episode **do**
  Reset the environment state
  Anneal the agent's exploration rate
  Initialize total reward and episode length
  while not done **do**
    Agent selects an action based on the current state
    Take action, receive next state, reward, and done flag from the environment
    Store the experience in the agent's memory
    if memory size $\geq$ batch size **then**
      Agent updates its neural network model using a batch of experiences
    end if
    Update total reward and episode length
  end while
  Update the target model
  Store the calculated metrics for the episode
end for=0

- **Modified Batch Deep Q-Learning with Dropout Layers**

The algorithm 4.2 initializes the **Modified BDQL-TO Agent** with hyperparameters and builds the neural network model with 4 dense layers and 2 dropout layers, which help prevent **overfitting**.

Algorithm 2 Modified BDQL-TO Network Architecture

Initialize Modified BDQL-TO Agent with state_size, action_size, discount_factor, learning_rate, memory_size, batch_size, exploration_max, exploration_min, exploration_decay
function BUILDMODEL
Create a neural network model with the following architecture:
  - Dense layer with 256 neurons, input_dim = state_size, activation = 'relu'
  - Dropout layer with rate = 0.2
  - Dense layer with 512 neurons, activation = 'relu'

  - Dropout layer with rate = 0.2
  - Dense layer with 256 neurons, activation = 'relu'
  - Dense layer with action_size neurons, activation = 'linear'
Compile the model with:
  - loss fuction : Mean squared error Loss = $\frac{1}{N}\sum_{i=1}^{N}(Q_{\text{predicted}}(s_i,a_i) - Q_{\text{target}}(s_i,a_i))^2$,
  - optimizer = Adam with learning_rate =0

5 Experiments and results analysis

In this section, we describe the experimental setup and evaluation of our proposed deep learning-based approach for multi-objective task offloading optimization in a Fog-Cloud environment. Our goal is to find the best offloading decisions for all tasks on all nodes while minimizing the total latency, total energy, and total cost. The settings of the experimental environment are shown in

System	Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz (12 CPUs), 2.6GHz
Memory	16 GB
Operating system	Windows 11
Programming language	Python 3.9
Tools used	Tensorflow, sklearn

Table 2: Simulation setup

5.1 Datasets

Two datasets are used in our experiments:[20] TaskDetails and NodeDetails. TaskDetails contains the following attributes: number of instructions (in 10^9 instructions), memory required (MB), input file

size (MB), and output file size (MB). NodeDetails includes attributes such as CPU rate (MIPS), CPU usage cost, memory usage cost, and bandwidth usage cost. Fog nodes have a CPU rate between 500 and 1500 MIPS, while cloud nodes have a CPU rate between 3000 and 5000 MIPS.

Since the NodeDetails dataset does not include execution power and transfer power attributes, we generate these values using a predefined distribution based on the nodes' types (Fog or Cloud) and their respective characteristics.

The datasets used for the training of our solution consist of 120 tasks and 13 nodes in total — 3 cloud nodes and 10 fog nodes.

5.2 Evaluation Metrics

To evaluate the performance of the batch deep Q-learning algorithm, we consider the following metrics and parameter values:

1. **Batch size (64)**: We use a batch size of 64, which is a common choice for training deep learning models. This means that 64 samples are randomly selected from the replay buffer and used to train the neural network at each step. Smaller batch sizes may lead to more noise in the gradient updates, while larger batch sizes may require more memory and computational resources.
2. **Learning rate (0.001)**: The learning rate of 0.001 is used for the optimizer to update the neural network's weights. This value is chosen because it is a common default value that provides a good balance between convergence speed and stability.
3. **Gamma (discount factor, 0.99)**: The discount factor, γ, is set to 0.99, which determines the importance of future rewards in the Q-value calculation. A value close to 1 means that the agent considers future rewards to be important, while a lower value means the agent prioritizes immediate rewards.
4. **Epsilon (exploration rate, 1.0 with a decay of 0.995)**: The initial exploration rate, ϵ, is set to 1.0, which means the agent starts with a 100% probability of taking a random action. The exploration rate decays by a factor of 0.995 per episode, which encourages the agent to gradually shift from exploration to exploitation as it gains more knowledge about the environment.
5. **Dropout rate (0.1)**: The dropout rate is set to 0.1, meaning that during training, 10% of the neurons in the dropout layers are randomly turned off. This helps in preventing overfitting and improves the model's generalization capability.
6. **Loss Function**: We use the Mean Squared Error (MSE) loss to measure the difference between the predicted Q-values and the target Q-values during training. The goal is to minimize this loss function [21].

$$\text{MSE Loss} = \frac{1}{N}\sum_{i=1}^{N}(Q_{\text{predicted}}(s_i, a_i) - Q_{\text{target}}(s_i, a_i))^2, \tag{12}$$

where $Q_{\text{predicted}}(s_i, a_i)$ is the predicted Q-value for action a_i in state s_i, $Q_{\text{target}}(s_i, a_i)$ is the target Q-value for action a_i in state s_i, and N is the total number of samples in the batch.

7. **Reward**: The reward is the objective function of our problem. The cumulative reward obtained during an episode is used to assess the agent's performance. The goal is to maximize the cumulative reward over time. the reward function is designed to represent the benefits and costs associated with offloading tasks to fog and cloud nodes. By maximizing the reward, the agent is essentially trying to find the best offloading decisions that balance the trade-offs between latency, energy consumption, and other factors.

$$Reward = -[\omega_1 Latency + \omega_2 E_{total} + \omega_3 C_{total}] \tag{13}$$

8. **Number of Episodes (500 episodes)**: The number of episodes required to converge or reach a satisfactory level of performance is another metric to evaluate the efficiency of the learning algorithm. Fewer episodes indicate faster convergence and more efficient learning.

5.3 Results and evaluation

In this section, we evaluated the performance of the BDQL-TO agent by analyzing the trend of the average reward over time in comparison to a Random agent.In Figure 3, We calculated the average reward over 10-episode intervals, and plotted these values against the number of episodes. This graph provided valuable insights into the agent's learning process.

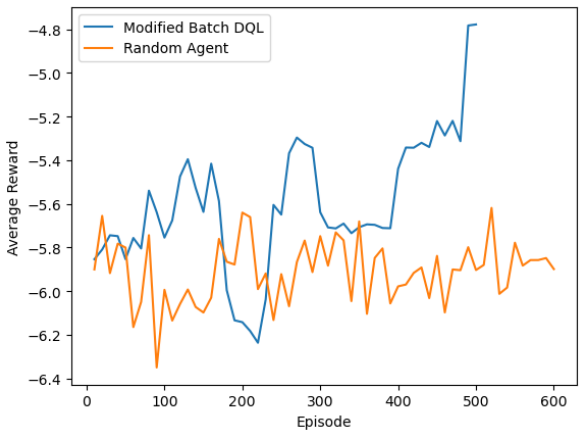

Figure 3: Average reward progression

The BDQL-TO agent demonstrated superior performance to the Random Agent in terms of average reward, with the exception of a brief period between episodes 170 and 230, during which the performance of the BDQL-TO agent was slightly lower due to factors such as the exploration-exploitation trade-off, adjustments in the learning rate, or other elements that may influence the agent's learning process. The results indicate that the BDQL-TO agent is an effective approach for the given offloading problem, outperforming the Random Agent in most of the episodes throughout the training process

From figure 4, The agent's learning progress appears to go through three distinct phases: an **initial exploration phase**, a **plateau phase** where the agent seems to be stuck in a local optimum, and an **improvement phase** where the agent starts learning better strategies.

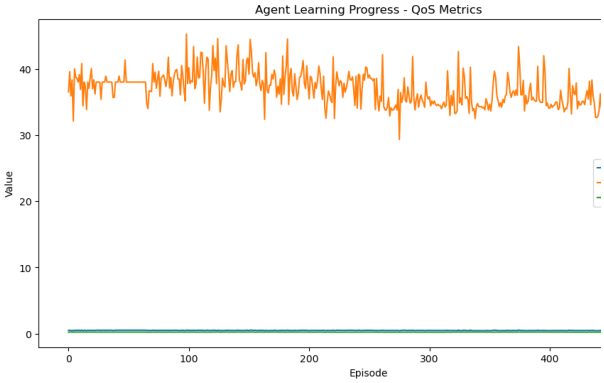

Figure 4: Agent Learning Progress over the Episodes.

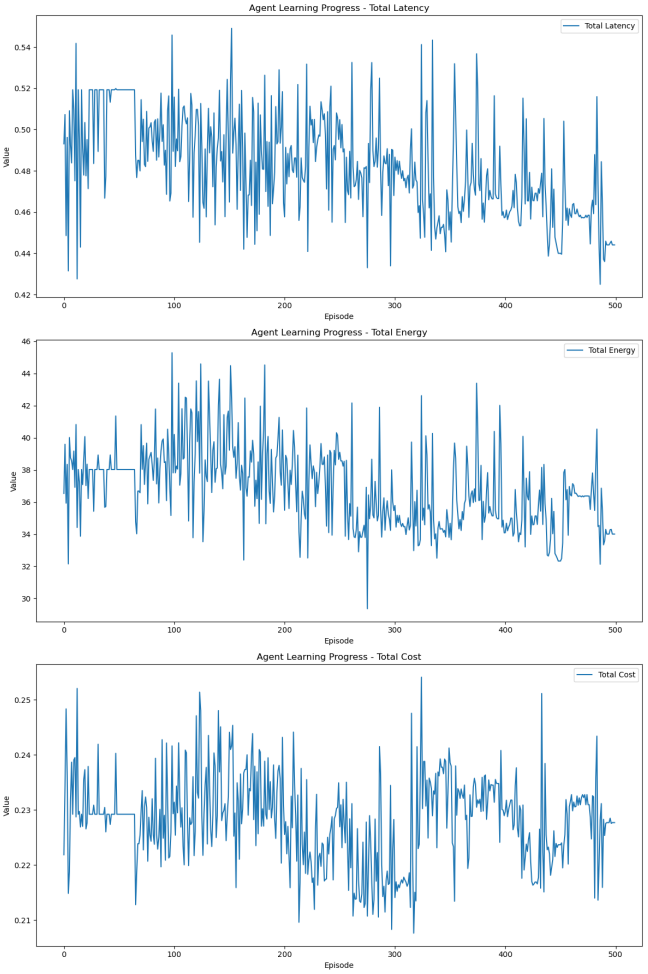

Figure 5: total QoS metrics learning progress

Figure 5 goes further in depth and shows that although the values for total latency, total energy and total cost seem to fluctuate in the first episodes because of the exploration phase, as our agent tries to understand the environment and adapt the actions (offloading decisions) accordingly, towards the end of the episodes the agent seems to manage to find a better solution after reaching a local optimum. We also compare the performance of the Batch Deep Q-Learning (BDQL-TO) algorithm with the Genetic Algorithm (GA) for task offloading optimization in Fog-Cloud environments in terms of total latency, total energy and total cost.

Metric	**Batch Deep Q-Learning**	**Genetic Algorithm**
Total Latency	0.4915	0.2834
Total Energy	17.9879	9.1515
Total Cost	0.2225	0.2462

Table 3: Comparison of Batch Deep Q-Learning and Genetic Algorithm

Table 3 shows that the GA achieved lower latency and lower energy consumption, while the BDQL-TO algorithm resulted in a lower average cost. In certain scenarios, when cost is a crucial factor, the BDQL-TO algorithm may be a more desirable choice for task offloading optimization.

Additionally, the BDQL-TO algorithm has the potential to scale well to larger problem sizes and adapt to complex and dynamic environments. As the number of tasks and nodes increases, the BDQL-TO algorithm may perform better than the GA due to its ability to learn and adapt to changes in the Fog-Cloud environment. This can be beneficial in real-world applications where environments are subject to constant changes.

Moreover, deep learning algorithms like BDQL-TO can handle incomplete or noisy data more effectively than traditional optimization algorithms like GA. In cases where the dataset for nodes is incomplete, the BDQL-TO algorithm can still learn and generalize from the available data to make better offloading decisions.

The Figure 6 for the modified DQL algorithm show

that approximately 67.6% of the tasks were offloaded to the fog nodes, with the remaining %32.4 being offloaded to the cloud nodes.

The Figure 7 for the GA show that approximately

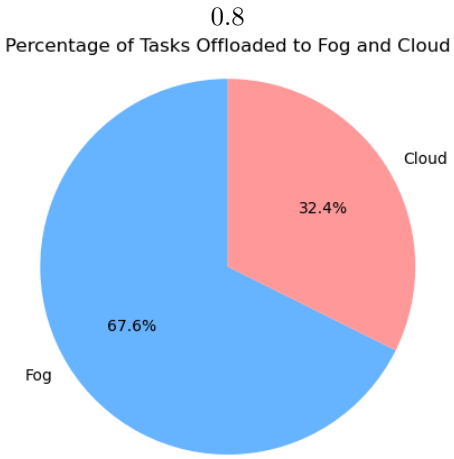

Figure 6: Percentage of tasks offloaded to the Fog and the Cloud in BDQL-TO

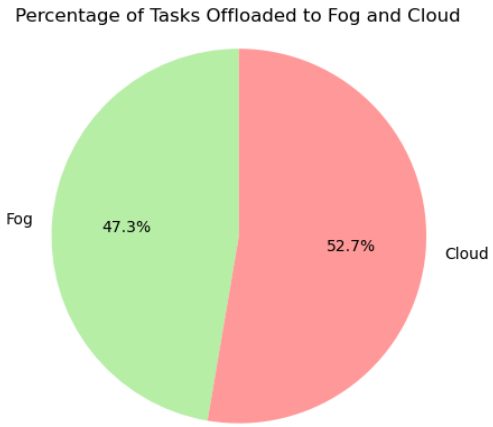

Figure 7: Percentage of tasks offloaded to the Fog and the Cloud in Genetic Algorithm

47.3% of the tasks were offloaded to the fog nodes, with the remaining 52.7% going to the cloud nodes.

The results suggest that the **BDQL-TO**agent is more likely to prefer offloading tasks to **fog nodes,** while the **genetic algorithm** is more likely to prefer offloading tasks to **cloud nodes**.
The different offloading percentages can be explained by the different algorithms used. The modified DQL algorithm learns a policy over time through **exploration and exploitation** of the environment based on the rewards it receives. This learning process involves training a deep neural network to approximate the Q-values (action-value function), which is more data-driven and adaptive. On the other hand, the genetic algorithm is a population-based search method that evolves a population of candidate solutions using selection, crossover, and mutation operators. It seeks a near-optimal solution by optimizing the fitness function, which in this case is the reward obtained from the environment.

6 Conclusion and prospects

6.1 Summary

In this paper, we have dealt with the problem of task offloading in Fog-Cloud environments. We adapted a Modified version of Deep Q-Learning with Batch learning to able to take all of the tasks information and added Dropout layers to prevent the overfitting. We implemented and evaluated our BDQL-TO algorithm in a simulated Fog-Cloud environment, and our results demonstrate its effectiveness in optimizing task offloading. We also conducted a comparative study with a known meta-heuristic algorithm, the Genetic Algorithm, to illustrate the advantages of our proposed approach. The BDQL-TO algorithm offers several advantages, such as cost optimization, scalability, adaptability, and the ability to handle incomplete or noisy data. Finally, we evaluated the performance of our algorithm against a Random Agent, which showcased the superiority of our approach in terms of average rewards, with only a brief period of slightly lower performance. This further solidifies the potential of the BDQL-TO algorithm for effective task offloading and QoS optimization in Fog-Cloud environments.

6.2 Future work

Our approach has several advantages over other existing methods, but there is room for improvement. One possibility is to introduce a double deep Q-learning architecture, which could potentially improve the stability and accuracy of the model's predictions. In our work, we focused on a static Fog-Cloud environment. Future research can explore the impact of dynamic environments, where the availability and capacity of Fog and Cloud nodes change over time, on the our proposed algorithm. It would also be interesting to compare the performance of our BDQL-TO model with other reinforcement learning algorithms, such as policy gradient methods and other meta-heuristics like the Particle Swarm optimization. This could provide valuable insights and help identify how to improve our approach.

- **Acknowledgement:**

This work was carried out in collaboration with PhD. Marwa MOKNI.

References

[1] Z. Zhang, C. Li, S. Peng, and X. Pei, "A new task offloading algorithm in edge computing,"

EURASIP Journal on Wireless Communications and Networking, vol. 2021, p. 17, Jan. 2021.

[2] P. Mell and T. Grance, "The NIST Definition of Cloud Computing,"

[3] "Fog computing and its role in the internet of things | Proceedings of the first edition of the MCC workshop on Mobile cloud computing."

[4] newsroom, "Cisco Delivers Vision of Fog Computing to Accelerate Value from Billions of Connected Devices."

[5] F. Bonomi, R. Milito, P. Natarajan, and J. Zhu, "Fog Computing: A Platform for Internet of Things and Analytics," in *Big Data and Internet of Things: A Roadmap for Smart Environments* (N. Bessis and C. Dobre, eds.), vol. 546, pp. 169–186, Cham: Springer International Publishing, 2014. Series Title: Studies in Computational Intelligence.

[6] M. Chiang and T. Zhang, "Fog and IoT: An Overview of Research Opportunities," *IEEE Internet of Things Journal*, vol. 3, pp. 854–864, Dec. 2016. Conference Name: IEEE Internet of Things Journal.

[7] K. Kumar and Y.-H. Lu, "Cloud Computing for Mobile Users: Can Offloading Computation Save Energy?," *Computer*, vol. 43, pp. 51–56, Apr. 2010. Conference Name: Computer.

[8] Y. Xia, X. Etchevers, L. Letondeur, T. Coupaye, and F. Desprez, "Combining hardware nodes and software components ordering-based heuristics for optimizing the placement of distributed IoT applications in the fog," in *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*, SAC '18, (New York, NY, USA), pp. 751–760, Association for Computing Machinery, Apr. 2018.

[9] F. Ait Salaht, F. Desprez, A. Lebre, C. Prud'homme, and M. Abderrahim, "Service Placement in Fog Computing Using Constraint Programming," in *2019 IEEE International Conference on Services Computing (SCC)*, pp. 19–27, July 2019. ISSN: 2474-2473.

[10] A. Mebrek, L. Merghem-Boulahia, and M. Esseghir, "Efficient green solution for a balanced energy consumption and delay in the iot-fog-cloud computing," in *2017 IEEE 16th International Symposium on Network Computing and Applications (NCA)*, pp. 1–4, 2017.

[11] H. Liu and G. Cao, "Deep Reinforcement Learning-Based Server Selection for Mobile Edge Computing," *IEEE Trans. Veh. Technol.*, vol. 70, pp. 13351–13363, Dec. 2021.

[12] J. Du, L. Zhao, J. Feng, and X. Chu, "Computation Offloading and Resource Allocation in Mixed Fog/Cloud Computing Systems With Min-Max Fairness Guarantee," *IEEE Transactions on Communications*, vol. 66, pp. 1594–1608, Apr. 2018.

[13] X. Zhao, L. Zhao, and K. Liang, "An Energy Consumption Oriented Offloading Algorithm for Fog Computing," in *Quality, Reliability, Security and Robustness in Heterogeneous Networks* (J.-H. Lee and S. Pack, eds.), (Cham), pp. 293–301, Springer International Publishing, 2017.

[14] M. Melnik and D. Nasonov, "Workflow scheduling using neural networks and reinforcement learning," *Procedia Computer Science*, vol. 156, pp. 29–36, 2019. 8th International Young Scientists Conference on Computational Science, YSC2019, 24-28 June 2019, Heraklion, Greece.

[15] K. Zhang, Y. Mao, S. Leng, Q. Zhao, L. Li, X. Peng, L. Pan, S. Maharjan, and Y. Zhang, "Energy-Efficient Offloading for Mobile Edge Computing in 5G Heterogeneous Networks," *IEEE Access*, vol. 4, pp. 5896–5907, 2016. Conference Name: IEEE Access.

[16] J. Baek and G. Kaddoum, "Heterogeneous Task Offloading and Resource Allocations via Deep Recurrent Reinforcement Learning in Partial Observable Multifog Networks," *IEEE Internet of Things Journal*, vol. 8, pp. 1041–1056, Jan. 2021. Conference Name: IEEE Internet of Things Journal.

[17] M. Marwa, S. Yassa, J. E. Hajlaoui, M. N. Omri, and R. Chelouah, *WORKFLOW SCHEDULING IN FOG-CLOUD COMPUTING ENVIRONMENT*. PhD thesis, 12 2021.

[18] S. Fujimoto, E. Conti, M. Ghavamzadeh, and J. Pineau, "Benchmarking batch deep reinforcement learning algorithms," 2019.

[19] B. M. Nguyen, H. Thi Thanh Binh, T. The Anh, and D. Bao Son, "Evolutionary algorithms to optimize task scheduling problem for the iot based bag-of-tasks application in cloud–fog computing environment," *Applied Sciences*, vol. 9, no. 9, 2019.

[20] "Cloud-fog computing dataset." `https://www.kaggle.com/datasets/sachin26240/vehicularfogcomputing`. Accessed on March 3, 2023.

[21] P. D, *Reinforcement Learning*. O'Reilly Media, 2020.