Task offloading optimization in Mobile Edge Computing based on Deep Reinforcement Learning

C. Silva[1]
[1]*Instituto Superior Técnico,*
University of Lisbon
Lisbon, Portugal
carlos.m.a.silva@tecnico.ulisboa.pt

***Abstract*—The Cloud Computing (CC) paradigm has risen in recent years as a solution to a need for computation and battery constrained User Equipment (UE) to run increasingly intensive computation tasks. Given its seemingly infinite amount of resources and pay as you go nature this paradigm has brought several advantages: 1) Extended battery life of User Equipment (UE)s by offloading computation; 2) Enables new types of applications intractable with User Equipment (UE)'s computation capabilities; 3) In an ever more data focused world it allows for unlimited storage capacity. Nevertheless, given the centralized nature of the CC paradigm, this option introduces significant network congestion problems and unpredictable communication delays not suitable for real-time applications. To cope with these problems, the Mobile Edge Computing (MEC) concept has been introduced, which proposes to bring computation resources closer to the edge of the mobile networks in a distributed way. However, given that these edge computation resources are limited, this paradigm comes with its set of challenges that need to be solved in order to make it viable. This work proposes to innovate by presenting a network management agent capable of making offloading decisions from a heterogeneous network of UEs to a heterogeneous network of MEC servers. This agent represents an orchestrator of a group of 5G Small Cells (SCeNBs), enhanced with computation and storage capabilities. In order to solve this high complexity problem, an Advantage Actor-Critic (A2C) agent is implemented and tested against several baselines. The proposed solution is shown to beat the baselines by making intelligent decisions taking into account computation, battery, delay and communication constraints ignored by the baselines. The solution is also shown to be scalable, data-efficient, robust, stable and adjustable to address not only overall system performance but to take into account the worst case scenario.**

***Index Terms*—Mobile Edge Computing, Computation Offloading, Energy and Performance Optimization, Delay Sensitivity, Deep Reinforcement Learning, Cloud Computing**

I. Introduction

A. Context

There has been a continuous exponential increase in the number of User Equipment (UE), such as smartphones, laptops and Internet of Things (IoT) devices. Their need to run ever more complex applications, given their energy and computation constrained environment, has led to the rise of Cloud Computing (CC) as an alternative to offload computation and storage needs. In the CC paradigm, computation resources are located in centralized data centers that can be considered infinite. This paradigm brings several advantages [1]:

1) By offloading computation, it extends the battery life of UE;
2) Enables computationally complex applications intractable with UE computation capabilities;
3) Provides seemingly unlimited storage capacity.

However, there are also shortcomings. The two main problems introduced by this paradigm are:

1) The distance from UEs to these servers introduces an unpredictable communication latency that makes some delay-constrained applications unviable;
2) This increase in UEs and their communication needs with CC servers leads to ever more congested network links decreasing Quality-of-Service (QoS) for everyone in the network.

These shortcomings gave rise to a new emerging concept known as Mobile Edge Computing (MEC). The main idea of MEC is to bring computation resources closer to the edge of the mobile network enabling offloading of complex computation tasks with strict delay requirements. As defined by the European Telecommunications Standards Institute (ETSI) in [2], this can be achieved by allocating computing nodes at the network's edge in a fully distributed manner to reduce communication overhead and execution delay for UEs.

B. Motivation

Edge computing nodes come with limited radio, storage and computational resources, which raise three main challenges [1]:

- The **decision of which computing tasks** are profitable for the UE to offload to the MEC servers in terms of energy consumption and execution delay.
- How to efficiently **allocate the limited computation resources** within the MEC servers in order to minimize response delay and load balance the computing resources and communication links.
- **Mobility management** to guarantee MEC service continuity and efficiency for UEs roaming the network.

Given the heterogeneous and stochastic nature of network topologies, traditional optimization techniques lack the scalability and adaptability to deal with unknown network conditions. Recent breakthroughs in machine learning algorithms, showcasing their ability to solve and adapt to complex

problems previously thought impossible to be solved by a computer, led many researchers to explore applying these methods to the MEC challenges.

The problem of making offloading and resource allocation decisions in MEC can be simulated and performance benchmarks are easily defined. This makes their definition as a Markov Decision Process (MDP) straightforward and allows the use of promising algorithms, like Deep Neural Networks (DNN) trained using Reinforcement Learning (RL) or Deep Reinforcement Learning (DRL) algorithms for short.

C. Contributions

The main contributions of this work are:

- The release of an open-source realistic network simulation environment as an OpenAI Gym, [3], enviroment. Allowing for easy experimentation with different DRL algorithms;
- The implementation of an open-source, A2C agent, capable of making intelligent offloading decisions that take into account battery, computation and communication constraints overperforming the baselines in a heterogeneous network of several UEs and MEC servers;

The code for the simulator, baselines and agent, as well as environment configurations can be found in the following code repository: **https://github.com/Carlos-Marques/rl-MEC-scheduler**.

II. Overview of MEC architectures

One of the first efforts to bring computation resources closer to the edge was the cloudlet concept presented in [4]. The main idea behind it was to allocate powerful computers at WiFi hotspots that could sell their Infrastructure as a Service (IaaS) through the use of Virtual Machines (VMs).

Another concept to bring computation closer to UEs is the idea of an *ad-hoc* cloud like the one presented in [5]. The idea proposes that the computation power of a network of non-exclusive and sporadically available hosts can be harvested and abstracted as a service to which UEs can offload their computational needs.

Finally, the Mobile Edge Computing (MEC) concept presented by the Industry Specification Group (ISG) within the European Telecommunications Standards Institute (ETSI) in [2] is focused on integrating computation resources within the Radio Access Network (RAN) in very close proximity to mobile subscribers.

As presented in paper [1], several MEC architectures have been proposed in order to integrate and manage computation resources at the RAN level such as Small Cell Cloud (SCC) [6], Mobile Micro Cloud (MMC) [7], Fast Moving Personal Cloud (MobiScud) [8], Follow Me Cloud (FMC) [9] and CONCERT [10].

III. Reinforcement Learning (RL)

There are three main machine learning paradigms, supervised learning, unsupervised learning and reinforcement learning. While supervised learning needs a lot of labeled input/ouput data and unsupervised learning learns patterns from unlabeled data, Reinforcement Learning (RL) is used in problems where an agent learns to solve a closed-loop problem by changing its future actions based on a reward resulting from its actions on the environment and its effects on its state.

This paradigm has shown great potential in solving environments that can be simulated and easily benchmarked which is the reason most research on RL is made by solving video games, which offer a perfect environment in terms of reproducibility, easy simulation and baked in benchmarks that can be used to compute a reward signal (points, levels, etc).

Several algorithms have been proposed from Deep Q-Network (DQN) in [12], Double Deep Q-Network (DDQN) in [13], Dueling Deep Q-Network (Dueling DQN) in [14] and Asynchronous Advantage Actor-Critic (A3C) in [15].

IV. Exploitation vs Exploration

One major challenge a RL agent faces is the exploration vs exploitation dilemma. Given a fixed set of resources that must be allocated between competing choices in a way that maximizes expected reward in an unknown transition probability and reward signal environment, the agent must choose to perform actions that exploit known strategies versus performing unknown actions to explore the environment for better strategies. These types of problems have been studied extensively as multi-armed bandit problems.

In order to solve this problem several algorithms have been proposed from ϵ-greedy algorithm, Boltzmann exploration in [16] and Adaptive Genetic Algorithm (AGA) in [17].

V. System proposition

A. Problem statement

The goal of this thesis is the design of a management agent capable of making offloading decisions from a heterogeneous network of UEs to a heterogeneous network of MEC servers. This work proposes to innovate by presenting a more complete system and exploring a network topology not present in related works. The proposed network manager could be seen as the conductor in the CONCERT architecture proposed in [10] or the small cell manager in the SCC architecture, [11]. This manager would be deployed locally and would manage a group of M MEC servers and N UEs making offloading decisions on which UE tasks to compute locally, which tasks to offload and where tasks should be offloaded to. This decision should take into account communication delays, computation constraints and battery consumption.

Building upon the system presented in [20], the proposed system plans address its shortcomings by:

- Expanding the number of MEC servers that must be orchestrated from one to M servers, better representing a 5G network with several SCeNBs enhanced with computation capabilities;
- Testing the management agent in a heterogeneous enviroment, with UEs and MEC servers of different computation capabilities;

- Taking into account the delay and energy cost of downloading the processed result from the MEC server to the UE;
- Exploring modern DRL algorithms to solve the increased complexity problem.

B. System Model

The proposed network model considers *N* UEs and *M* MEC servers, as represented in Figure 1.

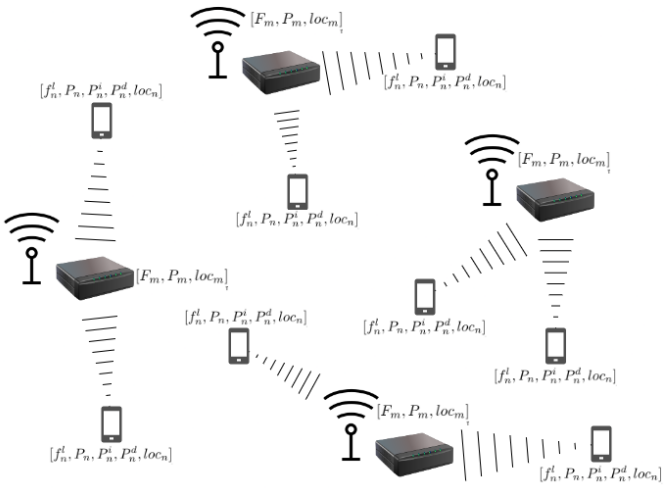

Figure 1. Proposed network model.

The set of UE is denoted as $\mathcal{N} = \{1, 2, ..., N\}$ while the set of MEC servers can be denoted as $\mathcal{M} = \{1, 2, ..., M\}$. For simplicity MEC servers are assumed to be connected to the power grid so their energy consumption is ignored.

- Each UE, $n \in \mathcal{N}$ can be described by its computing capacity, f_n^l (CPU cycles per second), transmit power, P_n, idle power consumption, P_n^i, download power consumption, P_n^d, and location, $loc_n = (x_n, y_n, z_n)$. The UE, n, can then be described by the vector, $u_n = [f_n^l, P_n, P_n^i, P_n^d, loc_n]$, and the system's UEs can be defined as the vector $\mathcal{U} = [u_1, u_2, ..., u_n]$.
- Each MEC server, $m \in \mathcal{M}$, can be described by its computing capacity, F_m (CPU cycles per second), its transmit power, P_m, and its location $loc_m = (x_n, y_n, z_n)$. The MEC server, m, can then be described by the vector, $s_m = [F_m, P_m, loc_m]$ and the system's MEC servers can be defined as the vector $\mathcal{S} = [s_1, s_2, ..., s_n]$.
- At each time step each UE is assumed to have a computation task to be completed. This task can either be computed locally or offloaded to one of the available MEC servers. The offloading decision of each computation task is denoted as $\alpha_n \in \{0, 1, ..., M\}$, where $\alpha_n = 0$ means local computation and $\alpha_n \in \mathcal{M}$ means offloading the task to MEC server $m = \alpha_n$. The total offloading decision can be defined as the decision vector $\mathcal{A} = [\alpha_1, \alpha_2, ..., \alpha_N]$ with a decision for each task.
- Each computation task, R_n, can be defined by an input data amount, B_n (bits), an output data amount, B_d (bits), the total number of CPU cycles required to compute it, D_n and the importance weights of time and energy costs, I_n^t and I_n^e. The importance weights of the task must satisfy $0 \leq I_n^t \leq 1$, $0 \leq I_n^e \leq 1$ and $I_n^t + I_n^e = 1$. The task, R_n, can then be described by the vector, $R_n = [B_n, B_d, D_n, I_n^t, I_n^e]$ and the system's tasks can be defined as the vector $\mathcal{R} = [R_1, R_2, ..., R_N]$.

If the network manager decides to compute the task, R_n, of UE n locally then a local computation model can be defined by a local execution delay T_n^l and an energy consumption E_n^l:

$$T_n^l = \frac{D_n}{f_n^l}, \tag{1}$$

$$E_n^l = z_n D_n, \tag{2}$$

where z_n represents the energy consumption per CPU cycle and is set to $z_n = 10^{-27}(f_n^l)^2$ according to practical observations made in [21].

Based on the computation delay and energy consumption of task, R_n, a local cost can be calculated according to:

$$C_n^l = I_n^t T_n^l + I_n^e E_n^l. \tag{3}$$

If the network manager decides to offload the computation to the MEC server $m = \alpha_n$, then the offload computation model can be defined by an upload delay, $T_{n,t}^m$, an upload energy consumption, $E_{n,t}^m$, an offload execution delay, $T_{n,p}^m$, an idle energy consumption, $E_{n,p}^m$, a download delay, $T_{n,d}^m$ and its corresponding download energy consumption, $E_{n,d}^m$.

Firstly, the UE n must upload the input data, B_n from the task R_n to the decided MEC server m. This upload has an associated delay defined as:

$$T_{n,t}^m = \frac{B_n}{r_u}, \tag{4}$$

where r_u is the uplink rate of UE n.

This upload has an associated energy consumption:

$$E_{n,t}^m = P_n T_{n,t}^m = \frac{P_n B_n}{r_u}. \tag{5}$$

After the data is uploaded the MEC server then computes the task resulting in a offload execution delay:

$$T_{n,p}^m = \frac{D_n}{f_m}, \tag{6}$$

where f_m is the amount of the MEC server, m, computation capacity, F_m, allocated to the offloaded task. To simplify the system the computation capacity of a MEC server is equally divided by all tasks offloaded to it:

$$f_m = \frac{F_m}{N_m}, \tag{7}$$

where N_m is the number of tasks offloaded to MEC server, m.

While the UE waits for the task to be computed it stays idle which has an associated energy consumption:

$$E^m_{n,p} = P^i_n T^m_{n,p} = \frac{P^i_n D_n}{f_m}. \tag{8}$$

Finally the computation results are downloaded to the UE n with an associated delay:

$$T^m_{n,d} = \frac{B_d}{r_d}, \tag{9}$$

where B_d is the size of the computation output and r_d is the download rate of UE n.

This download step has an associated energy consumption that can be calculated according to:

$$E^m_{n,d} = P^d_n T^m_{n,d}. \tag{10}$$

By taking into account the delays defined in Equations (4), (6) and (9), we can compute the total offload delay, T^m_n as the sum of all delays:

$$T^m_n = T^m_{n,t} + T^m_{n,p} + T^m_{n,d}. \tag{11}$$

The total energy consumption of offloading to the MEC server m, can be calculated by adding all energy consumptions defined in Equations (5), (8) and (10):

$$E^m_n = E^m_{n,t} + E^m_{n,p} + E^m_{n,d}. \tag{12}$$

Based on the computation delay and energy consumption of offloading task R_n to MEC server m, a cost can be calculated according to:

$$C^m_n = I^t_n T^m_n + I^e_n E^m_n. \tag{13}$$

The cost of the offloading decision $\alpha_n \in \{0, 1, ..., M\}$ can be computed according to:

$$C_n = \begin{cases} C^l_n & \alpha_n = 0 \\ C^m_n & \alpha_n \in \mathcal{M} \end{cases}. \tag{14}$$

The system's costs can be defined as the vector:

$$\mathcal{C} = [C_1, C_2, ..., C_N]. \tag{15}$$

The mean cost of the MEC system at each iteration can then be defined as:

$$C_{mean} = \frac{\sum_{n=1}^{N} C_n}{N}. \tag{16}$$

The worst case cost of the MEC system at each iteration can then be defined as:

$$C_{max} = \max\ \mathcal{C}. \tag{17}$$

This corresponds to the UE that has the highest cost at each iteration.

C. Problem formulation

The offloading decision needs to be formulated in a way that can both optimize overall system performance while considering the worst case serviced UE.

Given this, the weighted cost at each iteration can be defined as:

$$C = W_{mean} * C_{mean} + W_{max} * C_{max}. \tag{18}$$

By picking the importance weights of the costs that satisfy:

$$\begin{aligned} 0 \leq W_{mean} \leq 1, \\ 0 \leq W_{max} \leq 1, \\ W_{mean} + W_{max} = 1. \end{aligned}$$

The priorities of the system can be adjusted to satisfy both overall system performance and consider the worst case cost of the system.

The optimization function can then be defined with the objective of finding the offloading decision vector $\mathcal{A} = [\alpha_1, \alpha_2, ..., \alpha_n]$ that minimizes the weighted average of the mean and max cost of the system at each iteration.

D. Solution

Given the complex nature of the proposed problem and the lack of perfect knowledge of network conditions this thesis proposes to use a model-free DRL agent to manage the network. This means that the network manager does not have access to the transition probability, $P(s_{t+1}|s_t, a_t)$, nor the reward function $R(s, a)$ and must learn them by experimenting on the environment.

To do this the problem is formulated as an MDP, $< S, A, P, R >$:

- $S = \{s = (\mathcal{R})\}$ is the state space, which contains all requested tasks, $\mathcal{R}$;
- $A = \{a = (\mathcal{A})\}$ is the action space, which contains the offloading decision vector for all tasks, $\mathcal{A}$;
- $P : S \times A \times S \rightarrow [0, 1]$ is the transition probability distribution $P(s_{t+1}|s_t, a_t)$;
- $R = -C(s, a)$ is the reward function, which is defined as the inverse of the system cost, $C(s, a)$, given that the agent will be trying to maximize the reward.

At each time step, t, this network manager takes a state, s_t, makes a decision, a_t, that results in the state, s_{t+1}, and a reward r_t.

The DRL algorithm chosen for this agent was A2C. In order to solve the exploration vs exploitation dilemma, the A2C algorithm uses Boltzmann exploration.

As a function approximator for $V(s_t|\theta_v)$ and $\pi(a_t|s_t; \theta)$ a single neural network with two dense hidden layers was used. The hidden layers consist of 256 neurons each using the $tanh$ activation function.

The input layer of this model consists of the flattened state vector, S. Since the state vector is a vector of length N, for the number of tasks and each task is described by a 5 variable

vector, $R_n = [B_n, B_d, D_n, I_n^t, I_n^e]$, the total length of the input vector is $5 \times N$.

The output of the model consists of two parts. First a single linear output representing the value function, $V(s_t|\theta_v)$. Secondly a softmax output for each action dimension with one entry per action, representing the probability of selecting the action, $\pi(a_t|s_t;\theta)$.

Since the agent must decide for each task (N), between local computation or offloading to one of the MEC servers (M), there are N softmax functions with $M+1$ entries each, representing the possible actions. A diagram of the model is shown in Figure 2.

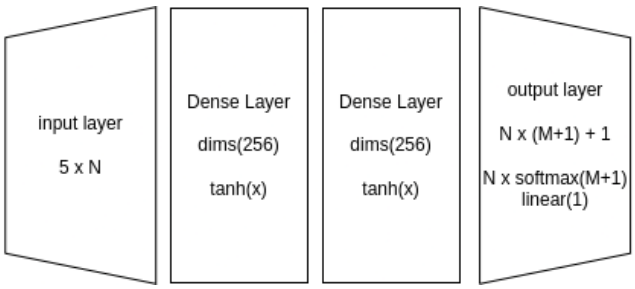

Figure 2. Model architecture.

VI. Results

A. Baselines

In order to benchmark the various DRL algorithms and test the implemented simulator, several baseline algorithms were developed:

- Full Local: all UEs execute their tasks locally, in terms of the decision vector $\mathcal{A} = [0_1, 0_2, ..., 0_N]$. This represents the case where the system has no offloading capabilities.
- Random Offload: where the offloading decision is completely random, each task is either executed locally or offloaded to one of the available MEC servers randomly;
- Full nearest MEC: all UE tasks are offloaded to the nearest MEC server according. This represents the case of offloading to the nearest node without regarding their computation constraints and other offloading decisions. This baseline also requires knowledge of the network topology which the agent does not have.

The plan is to create several system configurations of increasing complexity. The baseline algorithms will be used in order to benchmark the quality of trained agents. It is expected that the proposed network manager will surpass their performance in all situations by making intelligent decisions taking into account computation, battery, delay and communication constraints, which are ignored by the baselines.

B. Simple test

To put these baselines, the agent and the simulator to the test, a simple test case was devised. In this test, there are five UEs and five MEC servers. The UEs and MEC servers were randomly distributed in a 2D plane with $x \in [0, 200]$ and $y \in [0, 200]$, resulting in the distribution seen in Figure 3.

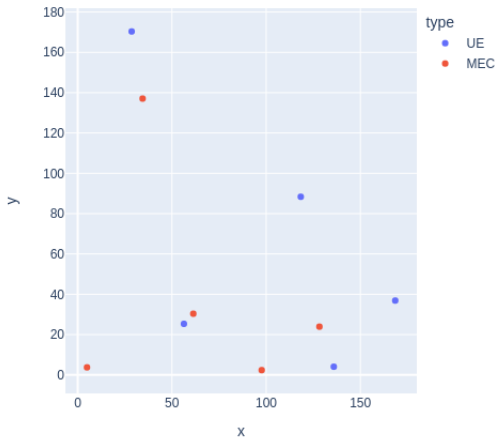

Figure 3. Example layout for the simple test.

For simplicity, all UEs and MEC servers are considered to have the same specifications. The system's hyper-parameters were set according to the values present in Table I. In order to test this configuration, the task parameters, B_n, B_d and D_n were sampled from uniform distributions between (300, 500) Kbits, (10, 15) Kbits and (900, 1100) Megacyles respectivly.

Variable	Value	Variable	Value
B	10×10^6	f_n^l	1×10^9
n	10	I_n^t	0.5
β	-4	I_n^e	0.5
h_ul	100	P_m	200
h_dl	100	P_n	500×10^{-3}
g_ul	1	P_n^i	100×10^{-3}
g_dl	1	P_n^d	200×10^{-3}
N_0	5×10^{-5}	F_m	5×10^9
W_{mean}	1	W_{max}	0

Table I
System hyper-parameters.

The agent described in Section V-D was trained with the following hyper-parameters:

Variable	Value
episode length	10
learning rate, α	0.0001
discount factor, γ	0.99
batch size	200

Table II
Training hyper-parameters.

Each algorithm ran for 100 episodes and an average per episode of offloading decisions is shown in Table III.

Algorithm	Average Reward (R)	Reduction
Full Local	-49.47	96.65%
Random Offload	-9.68	82.89%
Full nearest MEC	-2.18	24.06%
Agent (A2C)	-1.66	-

Table III
Results of simple test.

As expected, the Full Local baseline performed the worst given that it does not make use of the MEC servers computing capabilities, leading to an increased cost due to increased energy consumption and delay of the tasks. The Random Offload baseline is the second worst, given that although it gains from offloding to MEC servers, it does so without any consideration for the system's configuration. From the baselines, Full nearest MEC performed the best since it takes into consideration the distance between the UE and the MEC server but it comes with the disadvantage of not only requiring knowledge of the network topology (positions of UEs and MECs) but also not taking into account anything else.

As expected, the proposed agent outperformed the baselines in all situations, which demonstrates that the agent is able to learn not only the topology of the network by trying out different offloading decisions but also make intelligent decisions based on computation, battery, delay and communication constraints ignored by the baselines.

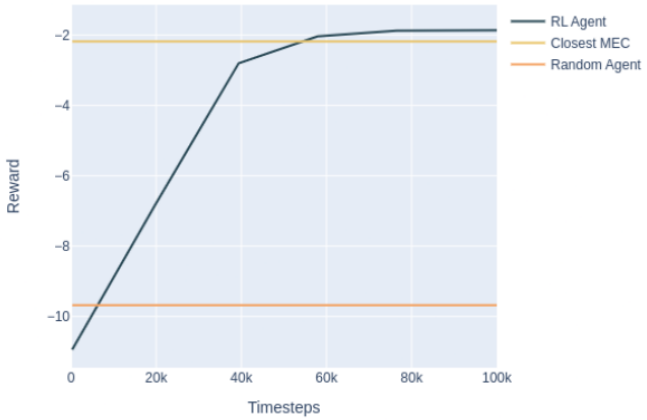

Figure 4. Agent reward while training.

As shown in Figure 4, the agent starts with an equivelent reward to the Random Offload, which is expected since the agent has not yet learned the environment. As the agent interacts with the environment making offloading decisions (actions), it recieves state transitions and rewards. This allows it to quickly learn the system and the reward increases rapidly in the first iterations. After about 55K steps in the environment, the agent surpasses the best offloading baseline, Full nearest MEC. As expected, the reward starts to stabilize as the agent learns the final details of the system increasing slightly the final performance.

C. Scalability and Data Efficiency

In this section the scalability and data efficiency of our agent is explored. Scalability in this context can be defined as the ability of the agent to learn and outperform the baselines in higher complexity problems.

Data efficiency in this context can be defined as the ability of the agent to learn in as few possible steps as possible. While it is expected that the agent takes longer to learn in more complex problems, the complexity of the underlying policy should not grow linearly with the state and action spaces. The reason for this is that it is expected that the agent is able to generalize concepts instead of brute forcing a solution for each problem set.

Since our state space is defined as the requested tasks at each step and each task is defined by a set of 5 parameters, the complexity of the state space grows linearly with the amount of UEs, *N*. The size of the state space should be $N \times 5$.

On the other side, the action space is defined as the offloading decision of each task. Given that each task can be computed locally or offloaded to one of the MEC servers, *M*, this gives us the option of $M+1$ actions per task. This results in the action space growing exponentially with the amount of UEs, *N*, since there are $(M+1)^N$ possible actions at each decision step.

With this in mind, the stability and data efficieny of the system can be tested by setting the system's hyper-parameters and amount of MEC servers to be the same as in the simple test case and create two new test scenarios by increasing the number of UEs to 10 and 20, respectively.

The UEs and MEC servers were randomly distributed in a 2D plane with $x \in [0, 200]$ and $y \in [0, 200]$, resulting in the distribution seen in Figure 5 and Figure 6.

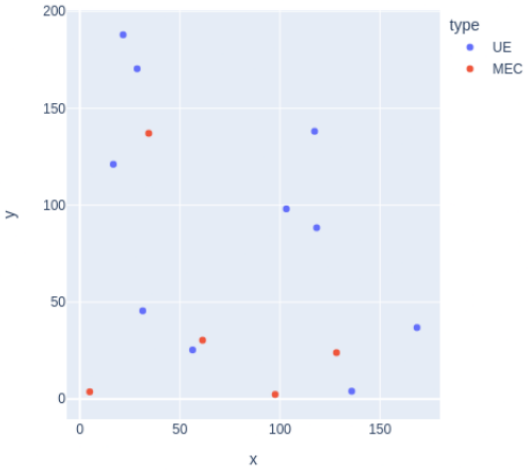

Figure 5. Test with 10 UEs.

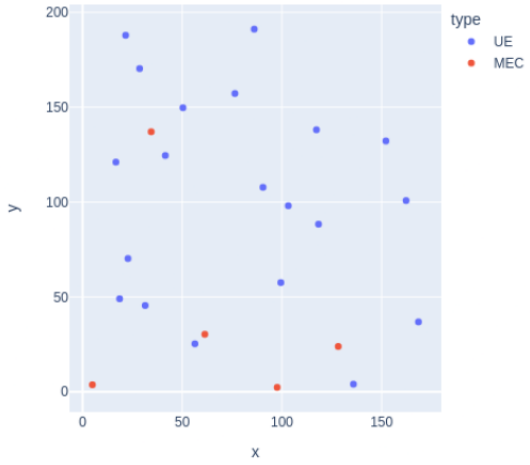

Figure 6. Test with 20 UEs.

Each algorithm ran for 100 episodes and an average per episode of offloading decisions is shown in Table IV and Table V.

Algorithm	Average Reward (R)	Reduction
Full Local	-49.50	94.40%
Random Offload	-10.32	73.14%
Full nearest MEC	-3.56	22.17%
Agent (A2C)	-2.77	-

Table IV
TEST RESULTS WITH 10 UES

Algorithm	Average Reward (R)	Reduction
Full Local	-49.49	90.42%
Random Offload	-11.67	59.36%
Full nearest MEC	-7.23	34.38%
Agent (A2C)	-4.74	-

Table V
TEST RESULTS WITH 20 UES

As expected, the reward of the Full Local baseline stays almost the same between test environments. This happens because these tests set W_{mean} to 1 and W_{max} to 0, so the cost is only composed by C_{mean}, defined in Equation 16. Since C_{mean} is the average cost of each UE's task, this should not scale with the amount of UEs.

The same cannot be said of the other baselines and the agent. Since the number of MECs stays the same and the number of tasks increases, the overall cost of the system increases with the amount of UEs. This represents the computation constraint of the MEC servers.

These results demonstrate that the agent scales with the complexity of the system.

In order to study the data efficiency of the agent, the agent's performance while training must be analysed.

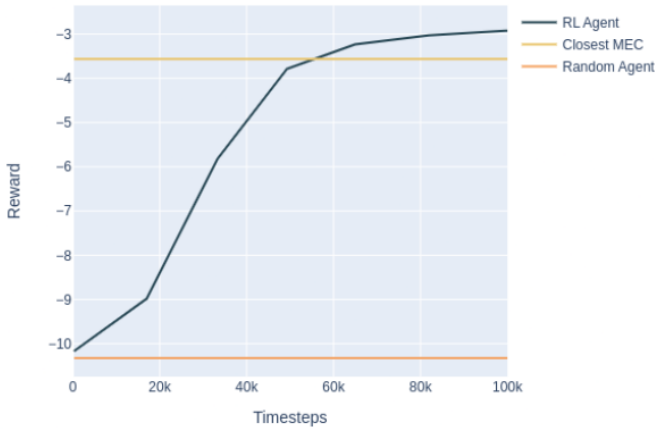

Figure 7. Training with 10 UEs.

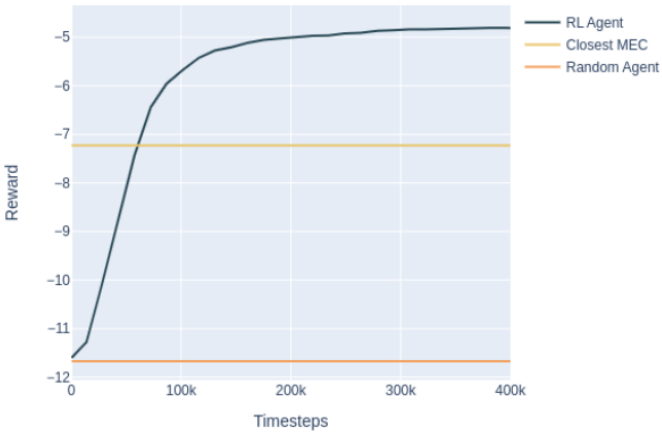

Figure 8. Training with 20 UEs.

The same trend as the simple test can be observed with the two new tests in Figure 7 and Figure 8.

This confirms that the agent is data efficient, meaning that even when the system's state space increases linearly and the action space increases exponentially, the agent is able to generalize concepts achieving a good performance. It does this with almost the same number of interactions with the system proving it is finding a strategy without brute forcing a solution.

D. Robustness and Stability

In this section the robustness and stability of our agent is explored. Robustness in this context can be defined as the ability of the agent to learn and outperform the baselines in environments with different network conditions and heterogeneous computation capabilities.

Stability in this context can be defined has the ability of the agent to not diverge or collapse while training. This means that its performance should improve as it interacts with the environment and after learning a strategy it should not forget it, collapsing the system's performance.

In order to test the system's robustness to changing network conditions, a new test case is devised by setting the system hyper-parameters to the values present in Table VI.

Variable	Value	Variable	Value
B	10×10^6	f_n^l	0.75×10^9
n	10	I_n^t	0.25
β	-4	I_n^e	0.75
$h_u l$	50	P_m	100
$h_d l$	50	P_n	250×10^{-3}
$g_u l$	0.5	P_n^i	50×10^{-3}
$g_d l$	0.5	P_n^d	100×10^{-3}
N_0	3×10^{-5}	F_m	2.5×10^9
W_{mean}	1	W_{max}	0

Table VI
NEW SYSTEM HYPER-PARAMETERS.

By setting the number of UEs to 10 and the number of MECs to 5, they were randomly distributed in a 2D plane with $x \in [0, 200]$ and $y \in [0, 200]$, resulting in the distribution seen in Figure 9.

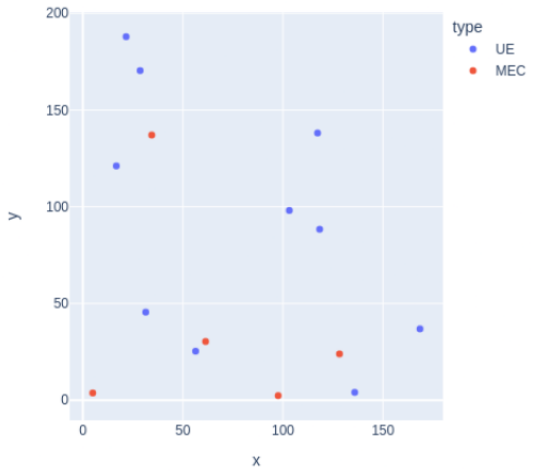

Figure 9. Robustness test layout.

Each algorithm ran for 100 episodes and an average per episode of offloading decisions is shown in Table VII.

Algorithm	Average Reward (R)	Reduction
Full Local	-31.31	84.92%
Random Offload	-9.14	48.35%
Full nearest MEC	-6.81	30.62%
Agent (A2C)	-4.72	-

Table VII
ROBUSTNESS TEST RESULTS.

As expected the reward values of all the algorithms differ from previous tests since the system hyper-parameters that are used to calculate the reward function are different.

The same cost reduction trends can be observed in this new environment, showing that the agent is able to learn the system's behaviour independently of the network conditions.

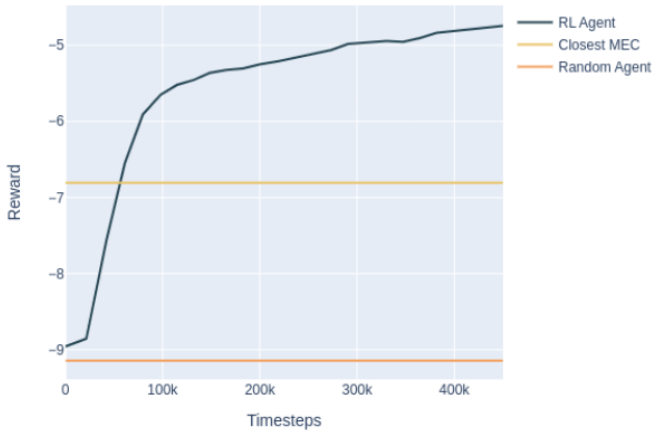

Figure 10. Training in robustness test.

As shown in Figure 10, the agent's performance while training follows a similar trend to previous tests.

The other component of robustness that must be tested is robustness to heterogeneous computation capabilities of UEs and MEC servers. To test this, a new test case is devised by resetting the system hyper-parameters to the values present in Table I but instead of setting f_n^l and F_m to a fixed value, they are set to a value randomly sampled from a set of values, $f = \{0.25, 0.5, 0.75, 1\} \times 10^9$ and $F = \{2.5, 5, 7, 10\} \times 10^9$, respectively.

By setting the number of UEs to 10 and the number of MECs to 5, they were randomly distributed in a 2D plane with $x \in [0, 200]$ and $y \in [0, 200]$, resulting in the distribution seen in Figure 11.

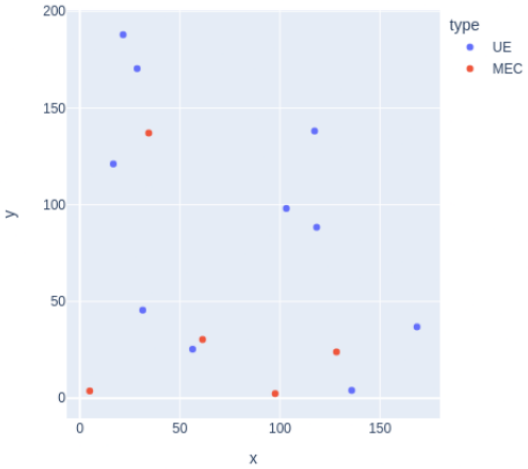

Figure 11. Heterogeneous test layout.

Each algorithm ran for 100 episodes and an average per episode of offloading decisions is shown in Table VIII.

Algorithm	Average Reward (R)	Reduction
Full Local	-33.56	89.55%
Random Offload	-8.80	60.16%
Full nearest MEC	-5.04	30.39%
Agent (A2C)	-3.51	-

Table VIII
HETEROGENEOUS TEST RESULTS.

As presumed, the reward values of all the algorithms differ from previous tests since the system computation capabilities that are used to calculate the reward function are different.

Although the ordering of the algorithm performances stays the same in this test case, the cost reduction of our agent in comparison to the Full nearest MEC is higher. With a reported 30.39% improvement, while the equivelant homogeneous test had an improvement of 22.17%. This makes sense given that the Full nearest MEC only takes into account the distance of UEs to MECs and not their computation capabilities. This in turn leads to worse load balancing when the computation capabilities of the UEs and MECs are not all the same.

The agent on the other hand is able to learn the system's network topology and make offloading decisions that better load balance the required tasks.

By analysing all training curves, it is clear that the agent is stable and its reward does not diverge or collapse while trainnig, even when ran for millions of iterations.

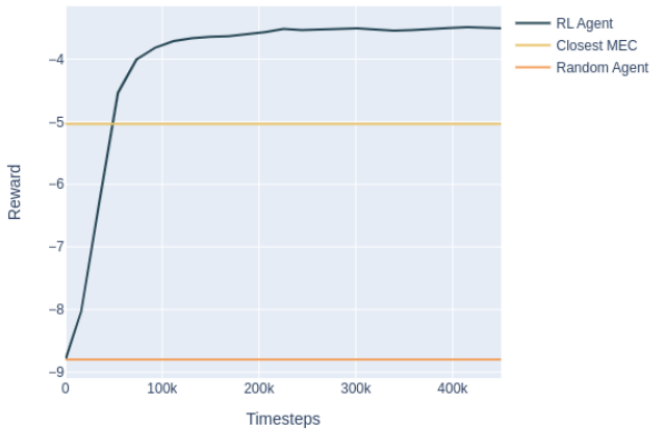

Figure 12. Training in heterogeneous environment.

As shown in Figure 12, the agent's performance while training follows a similar trend to previous tests.

E. Reward function weights

This section focuses on studying the effect of the cost function importance weights on the agent's overall and worst case performance.

To test this, five new test cases were created by setting the system hyper-parameters to the values present in Table I but instead of setting W_{mean} to 1 and W_{max} to 0, they are set to $(1, 0.75, 0.5, 0.25, 0)$ and $(0, 0.25, 0.5, 0.75, 1)$, respectively.

By setting the number of UEs to 10 and the number of MECs to 5, they were randomly distributed in a 2D plane with $x \in [0, 200]$ and $y \in [0, 200]$, resulting in the distribution seen in Figure 13.

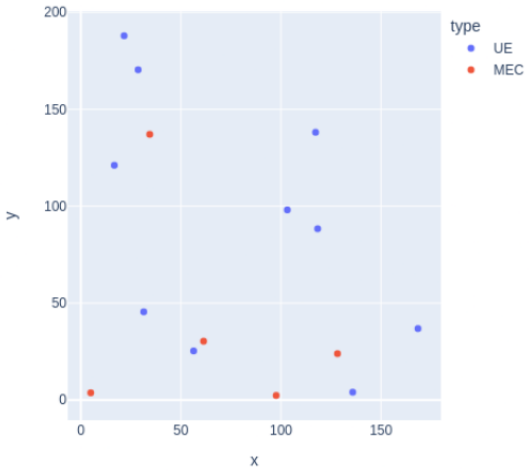

Figure 13. Weight test layout.

Each combination of the cost function ran for 100 episodes and an average per episode of offloading decisions is shown in Table IX.

$\{W_{mean}, W_{max}\}$	Overall Cost	Worst Cost	Difference
$\{1, 0\}$	2.72	4.04	1.32
$\{0.75, 0.25\}$	2.97	3.84	0.87
$\{0.5, 0.5\}$	3.22	3.53	0.31
$\{0.25, 0.75\}$	3.52	3.81	0.29
$\{0, 1\}$	3.81	3.78	-0.03

Table IX
WEIGHT TEST RESULTS.

As the weights shift from prioritizing overall performance to prioritizing worst case performance, the agent's overall cost increases while the worst case cost decreases. While the agent outperforms the best baseline with every weight distribution, it learns to make offloading decisions that prioritize the worst case performance at the expense of overall system performance. With this in mind, the process of picking the best weight distribution would involve trying to determine the maximum cost allowed per UE and shifthing the weight distribution in favour of W_{max}, trying to maximize W_{mean} while still meeting the maximum allowed cost.

VII. CONCLUSION

A. Summary

As presented in Section II several architectures have been proposed in the MEC context that present complex optimization problems of managing how to distribute tasks between UEs and MEC servers. Due to the high dimensional complexity and uncertain networking conditions, classical offline optimization algorithms fail to effectively manage these types of problems. Based on the work done by papers [18] and [20] a more complete system that takes into account the possibility of offloading between a network of heterogeneous UEs to a

network of heterogeneous MEC servers is proposed. As far as the candidate knows, this is the first time that this extended optimization problem is addressed. To deal with the increased complexity of taking into account computation, battery, delay and communication constraints in an N to N problem a network manager agent is proposed in Section V-D. In order to evaluate the performance of the proposed agent, several baselines are described in Section VI-A. Finally, the agent's capabilities are put to the test in Sections VI-B, VI-C, VI-D and VI-E.

In Section VI-B, a simple test case is used to demonstrate the simulator, baselines and the agent's capacity to learn. As expected, the Full Local baseline performs the worst, then the Random Offload baseline and finally the Full nearest MEC. The proposed agent is shown to learn, overperforming the baselines by 96.65%, 82.89% and 24.06% respectively. The agent achieves this while not requiring any information of the network topology by simply interacting with the environment.

In Section VI-C, the agent's scalability and data efficiency is put to the test by increasing the complexity of the system with two new tests. Given that the system complexity is tightly related with the number of UEs in the network, the agent was tested in an environment with 10 and 20 UEs. As showcased, the agent learned to overperform the best baseline with a 22.17% and 34.38% improvement, respectively, showcasing its ability to scale with problem complexity. The agent also achieved this in 55K steps, demonstrating it's data efficiency.

Section VI-D serves to test the agent's robustness and stability. In order to test robustness, two new test cases were implemented. First, the agent robustness to changing network conditions was tested and proven by overperforming the baseline with a 30.62% improvement. Secondly, the agent robustness to an environment with heterogeneous computation capabilities of UEs and MEC servers was put to the test. The agent not only was able to learn and overperform the best baseline, but it did so in a greater faction than with simpler tests, demonstrating a reduction of 30.39% over the Full nearest MEC baseline, compared with the 22.17% of the same test without heterogeneous computation capacity. Stability was analysed by looking at the different learning curves of the agent. Over all previous tests, the agent was shown to be stable independently of the changing conditions, never showing regression or collapse of its reward after it starts learning the environment.

In the end, Section VI-E presents the effects of changing the importance weight distribution from the overall performance to the worst case performance. The adjustability of the reward function is shown to allow the improvement of the worst case cost at the expense of the overall system cost.

All the simulator, baselines and agent code, as well as environment configurations can be found in the following code repository: **https://github.com/Carlos-Marques/rl-MEC-scheduler**.

B. Future Work

Future steps of research should include:

- Expanding the proposed simulator to include more dynamic environments, implementing the ability to change the following as simulation progresses:
  - UE locations to simulate mobility;
  - The number of UEs and MEC servers;
  - Network conditions.
- Creating a more complete test set, testing environments for specific scenarios. For example, a Smart city environment or a autonomous vehicle environment.
- Exploring the use of this orchestration agent with choreography algorithms like the one proposed in [19], by simulating an environment where several groups of MEC servers would be orchestrated by agents like the one proposed in this work, while choreographing with each other.
- Exploring the use of model based RL algorithms to take advantage of the extensive domain knowledge of telecomunication physics. By leveraging this knowledge, the agent could converge to a better solution more quickly, without having to learn this through experimentation.

REFERENCES

[1] A. Shakarami, M. Ghobaei-Arani, and A. Shahidinejad, "A survey on the computation offloading approaches in mobile edge computing: A machine learning-based perspective," *Computer Networks*, vol. 182, p. 107496, 2020.

[2] M. Patel, B. Naughton, C. Chan, N. Sprecher, S. Abeta, A. Neal, *et al.*, "Mobile-edge computing introductory technical white paper," *White paper, mobile-edge computing (MEC) industry initiative*, vol. 29, pp. 854–864, 2014.

[3] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *CoRR*, vol. abs/1606.01540, 2016.

[4] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *IEEE Pervasive Computing*, vol. 8, no. 4, pp. 14–23, 2009.

[5] G. A. McGilvary, A. Barker, and M. Atkinson, "Ad hoc cloud computing," in *2015 IEEE 8th International Conference on Cloud Computing*, pp. 1063–1068, 2015.

[6] F. Lobillo, Z. Becvar, M. A. Puente, P. Mach, F. Lo Presti, F. Gambetti, M. Goldhamer, J. Vidal, A. K. Widiawan, and E. Calvanesse, "An architecture for mobile computation offloading on cloud-enabled lte small cells," in *2014 IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*, pp. 1–6, 2014.

[7] S. Wang, G.-H. Tu, R. Ganti, T. He, K. Leung, H. Tripp, K. Warr, and M. Zafer, "Mobile micro-cloud: Application classification, mapping, and deployment," in *Proc. Annual Fall Meeting of ITA (AMITA)*, 2013.

[8] K. Wang, M. Shen, J. Cho, A. Banerjee, J. K. V. der Merwe, and K. Webb, "MobiScud: a fast moving personal cloud in the mobile network," 2015.

[9] A. Aissioui, A. Ksentini, and A. Gueroui, "An efficient elastic distributed sdn controller for follow-me cloud," in *2015 IEEE 11th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, pp. 876–881, 2015.

[10] J. Liu, T. Zhao, S. Zhou, Y. Cheng, and Z. Niu, "Concert: a cloud-based architecture for next-generation cellular systems," *IEEE Wireless Communications*, vol. 21, no. 6, pp. 14–22, 2014.

[11] H. E. Project, "Small cells coordination for multi-tenancy and edge services (sesam).." Online, 2015. Last access on 28/05/2021.

[12] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, Feb 2015.

[13] H. v. Hasselt, "Double q-learning," in *Proceedings of the 23rd International Conference on Neural Information Processing Systems - Volume 2*, NIPS'10, (Red Hook, NY, USA), p. 2613–2621, Curran Associates Inc., 2010.

[14] Z. Wang, N. de Freitas, and M. Lanctot, "Dueling network architectures for deep reinforcement learning," *CoRR*, vol. abs/1511.06581, 2015.
[15] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," *CoRR*, vol. abs/1602.01783, 2016.
[16] N. Cesa-Bianchi, C. Gentile, G. Lugosi, and G. Neu, "Boltzmann exploration done right," *CoRR*, vol. abs/1705.10257, 2017.
[17] X. Qiu, L. Liu, W. Chen, Z. Hong, and Z. Zheng, "Online deep reinforcement learning for computation offloading in blockchain-empowered mobile edge computing," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 8, pp. 8050–8062, 2019.
[18] Y. Gong, C. Lv, S. Cao, L. Yan, and H. Wang, "Deep learning-based computation offloading with energy and performance optimization," *EURASIP Journal on Wireless Communications and Networking*, vol. 2020, p. 69, Mar 2020.
[19] L. P. de Matos Morgado Ferreira, "Fog computing task offloading optimization based on deep reinforcement learning," Master's thesis, Instituto Superior Técnico, 2021.
[20] J. Li, H. Gao, T. Lv, and Y. Lu, "Deep reinforcement learning based computation offloading and resource allocation for mec," in *2018 IEEE Wireless Communications and Networking Conference (WCNC)*, pp. 1–6, 2018.
[21] Y. Wen, W. Zhang, and H. Luo, "Energy-optimal mobile application execution: Taming resource-poor mobile devices with cloud clones," in *2012 Proceedings IEEE INFOCOM*, pp. 2716–2720, 2012.