Task Offloading in Autonomous IoT Systems using Deep Reinforcement Learning and ns3-gym

Abdel Kader Chabi Sika Boni, Hassan Hassan, Khalil Drira

Task Offloading in Autonomous IoT Systems using Deep Reinforcement Learning and ns3-gym

Abdel Kader Chabi Sika Boni
LAAS-CNRS
University of Toulouse
Toulouse, France
akchabisik@laas.fr

Hassan Hassan
LAAS-CNRS
University of Toulouse
Toulouse, France
hhassan@laas.fr

Khalil Drira
LAAS-CNRS
University of Toulouse
Toulouse, France
khalil@laas.fr

***Abstract*—IoT systems grow quickly and are massively present in urban areas. Their successful deployment requires autonomy that can be built on automated learning technologies such as Deep Learning. The IoT applications require important computational resources, rarely available on devices. Autonomous IoT systems require the computation power available on the edge and cloud servers in order to offload some tasks related to the supported applications and the underlying platforms. Task offloading constitutes a big challenge in autonomous IoT systems due to the huge number of IoT devices for scenarios of the family of smart cities. Managing task offloading in such contexts requires adaptive strategies capable of taking into consideration the rapid evolution of available resources and proposing efficient offloading solutions to all received requests. In this paper we use a Deep Reinforcement Learning (DRL) approach capable of handling large state spaces, and resolve the optimization problem in this context, where other techniques can not scale efficiently. Our solution is based on a DRL agent that was developed in the ns3-gym framework and was tested on IoT system scenario implemented in the NS3 simulator. The results obtained show that the DRL agent can adapt quickly to resource evolution in the IoT system and can handle big number of demands fulfilling scalabilty requirements of autonomous IoT systems.**

***Index Terms*—Autonomous IoT systems, Task offloading, Deep Reinforcement Learning, optimization**

I. Introduction

IoT devices are used widely in many environments: industrial IoT, smart cities, transport networks, etc. New applications deployed in these environments constitute a big challenge as IoT devices are very constrained in resources (small CPUs, limited memory, batteries...). Besides, new IoT systems expand very quickly including a large number of devices with heterogeneous capabilities and using smart applications based on new techniques such as Deep Learning. The deployment of modern IoT systems require an overall automation and resources management. Autonomous IoT systems should be able to propose new strategies of resource allocation and management to deploy applications on all available devices in the system. One of the strategies that can be used to achieve this goal is task offloading.

Task offloading is the process or technique that is used to improve the performance, quality or efficiency of a computational task by delegating that task completely or partially to a remote computational machine that usually has a more powerful computational capacity than the local machine [4]. This technique requires a close collaboration between IoT devices, edge and cloud servers present in the autonomous IoT system. IoT devices can rely on available servers in order to offload heavy tasks. At the same time, servers should manage requests from a huge number of IoT devices. Handling such a complex problem requires adapting techniques capable of managing an increasing number of demands based on available resources. Although many techniques can be used to solve this problem, a few are capable of managing large state spaces such as in smart cities. In this work we explore the use of Deep Reinforcement Learning technique to handle the task offloading problem in autonomous IoT systems using ns3-gym [13] simulator.

The rest of this paper is organized as follows: in section 2 we give a brief overview of related work, in section 3 we state the problem, in section 4 we present the DRL model and in section 5 the simulation tool, in section 6 we present the simulation results and we conclude in section 7 with future work.

II. Related work

Several recent works have proposed solutions to solve the task offloading problem. The authors in [27], [11], [24], [17], [6], [5], [30], [2], [33] and [1] formulated task offloading as an optimization problem that they solved using heuristic methods. However, these methods are not very efficient or with an unacceptable resolution time when considering offloading scenarios with very large state space. This approach is the one adopted also in [10], [22], [9], [25], [19], [31] and [20], as the authors have treated task offloading in conjunction with the problem of resource allocation in resource constrained networks. Task offloading can be approached as a Markov Decision Process (MDP) problem as suggested by the authors in [8], [34], [29], [32], [26], [7], [8], [33] and [15]. To determine the optimal policy for the MDP problem several works, [32], [26], [15], have implemented strategies based on Q-learning, a classical reinforcement learning algorithm. Meanwhile this algorithm suffers from the same deficits of resolution time as the heuristic methods. The authors of [8], [35], [16], and [29] have exploited the power of neural networks with Deep Q-Networks to estimate the value functions in a Deep Q-learning resolution. This is a

fairly promising approach and is considered robust. However, in [8], an architecture is implemented where a mobile user generates tasks following a Bernoulli distribution. These tasks are independent of each other and can be executed locally on mobile device or on a cloud. The transmission of a task to the cloud is done via one of the N available base stations. The energy model is based on wireless charging of the mobile user, but this model is not very realistic. The choice of offloading to a remote cloud is also questionable because edge computing could be more efficient. Finally, the scalability of the solution has not been demonstrated; this makes it difficult to adopt in massive IoT systems. Zhang et al. [35] designed an architecture where mobile devices, in addition to executing locally, have the choice to offload their tasks to a multitude of edge servers. Those servers do not collaborate with each other. The authors have jointly addressed the delay sensitivity problem by defining two types of tasks, i.e., delay-constrained tasks and delay-sensitive tasks. However, the authors did not address the energy consumption problem at all, which is very important given the energy constraints of mobile devices. Moreover, the formulation of their task offloading model does not mention the size of the tasks as a parameter and the scalability of their model has been proven for a maximum of 500 mobile devices, which is very low at the scale of large IoT scenarios i.e. smart cities. Huang et al. [16] addressed the task offloading problem together with resource allocation. They implemented an architecture composed of wireless devices capable of performing tasks locally or on a server. They equipped each device with a time-division-multiplexing (TDD) circuit that allows it to perform its task while charging itself via wireless power transfer (WPT) technology. This makes their solution very specific to WPT enabled devices. They implement a reinforcement learning agent based on a single neural network. This agent allowed them to demonstrate the scalability of their solution for only about 30 devices. In the state of the art of deep reinforcement learning, single neural network agents are considered to be less efficient than dual neural network agents (Double Deep Q-learning) [28].
The authors in [33] formulated the task offloading as a MDP problem which they solved by a dynamic programming heuristic algorithm. An idea that we had discarded in our work especially since the efficiency of such algorithms is less compared to Deep Q-learning in large state spaces.

From an architectural point of view, the majority of the works ([16], [22], [25], [6], [2], [19], [31], [7], [8], [33], [20], [15] and [1]) adopt a centralized architecture while some others [36], [10], [6], [30], [32], [26] have implemented a decentralized architecture. In the centralized architecture, offloading decisions are made at a central point, whereas in a decentralized architecture each IoT device implements the decision making algorithm. Therefore, the decentralized architecture seems more realistic. However, IoT devices that lack resources to run the algorithm may be discarded from the proposed solution. An IoT device that is discarded in this way and operates its offloading decisions poorly could hinder the choices of other devices that run the algorithm. The authors of [21], [17], [9] and [5] have conducted their work on task offloading in a Software-Defined Networks (SDN) context. In a task offloading problem, the type of collaboration between the different entities can be horizontal or vertical. This allows for different offloading scenarios [3]. A collaboration between IoT devices and an edge server is implemented in [34], [23], [16], [9], [25], [32], [2], [19], [31], [7], [8], [20], [15] and [1]. This is a vertical collaboration. The authors demonstrated gain in response time as edge servers are considered closer to IoT devices. Whereas in [27], [18] and [30] the authors instead have IoT devices collaborate directly with cloud servers. This means that a task is always processed because cloud resources are considered as unlimited. However, the priority is no longer put on the response time. An intermediate collaboration between the horizontal and vertical approach is used in [36] where an IoT device can offload its task to an edge server which in turn can distribute the task between itself and other edge servers. Adding a central cloud server to this operation results in the collaboration employed in [10], [29], [22], [6], [5], [14] and [17].
We consider that the resolution of task offloading and resource allocation problems can be done independently of each other with the objective of simply adapting to the available resources in order to make the right task offloading choices. It is therefore not always relevant to manage resource allocation in parallel with task offloading. In our approach, we have therefore focused only on the task offloading problem based on a Deep Q-Network approach. Moreover, to expand the solution to all IoT devices in the system we adopted a centralized architecture. Thus, the ability of the IoT device to run the algorithm is no longer a hindrance. We chose an IoT device-Edge server collaboration in order to have a reasonable response time. To the best of our knowledge, our work is the only one to use the ns3-gym framework [13] to address the task offloading problem by deep reinforcement learning. It is also one of the few works to consider the resources of IoT systems as being very dynamic with frequent random variations. The numerous simulations carried out have shown that our solution converges faster and guarantees good offloading decisions in near-real time compared to existing solutions.

III. Problem statement

A. IoT system model

We consider an IoT system consisting of an edge server with which IoT devices communicate wirelessly via an AP access point as depicted in figure 1. The system is composed of N IoT devices each with M tasks to execute. We consider a discrete time scale $D_t = t_i$ with $i \in \{1, 2, ..., M\}$. At each time t_i, each IoT device executes one and only one task among the M tasks it has.
The n_{th} IoT device has two possibilities of executing its m_{th} task at the discrete time t_i : a local execution on the IoT

device or an offloading of the task to the edge server. A local execution uses the limited device CPU and energy source. We assume that the computing power of IoT devices is much lower than edge server. The choice of execution type is made at the beginning of each t_i time and we assume that this choice does not change during the same time t_i. However, the choice can be different for two different times t_i.

B. Task model

We use S_n to represent the task in terms of the transferable data size and D_n the size of the task expressed as the total number of CPU cycles required to complete it. In other words, D_n is the amount of computational resources needed to complete the task. The relationship between S_n and D_n is expressed as in equation (1)

$$S_n = \theta * D_n \tag{1}$$

We assume that a local computing or an offloading to the edge server has no impact on D_n.
In our model, we assume that one task is indivisible and can not be processed on different devices, i.e., an IoT device must choose exclusively between local and edge computing to execute its task.

C. Local computing

We denote by T_n^{loc} the local execution delay for the nth IoT device when it decides to execute its task locally. This delay is considered as the processing delay by the local CPUs and is expressed as in (2)

$$T_n^{loc} = \frac{D_n}{f_n^{loc}} \tag{2}$$

Next, we define E_n^{loc} as the energy consumption during the local execution of the task

$$E_n^{loc} = D_n * e_n^{loc} \tag{3}$$

where e_n^{loc} is the energy consumption per task bit processed locally.

D. Edge computing

When the current task is to be offloaded, we denote by T_n^{edg} the temporal cost of offloading (edge computing) for the nth IoT device. It is composed of the transmission delay $T_{n,t}^{edg}$ i.e., the time needed for the task to be transmitted from the IoT device to the edge server and the processing delay $T_{n,p}^{edg}$ depending on the processing speed by the edge server CPUs. These expressions are explained in equations (4), (5) and (6)

$$T_{n,t}^{edg} = min(\frac{S_n}{d_n}, \frac{S_n}{b_n}) \tag{4}$$

$$T_{n,p}^{edg} = \frac{D_n}{f_n^{edg}} \tag{5}$$

$$T_n^{edg} = T_{n,t}^{edg} + T_{n,p}^{edg} \tag{6}$$

Then, as for the local computing, we define E_n^{edg} as the energy consumption during the execution of the task on the edge server and express it by (7)

$$E_n^{edg} = D_n * e_n^{edg} \tag{7}$$

where e_n^{edg} is the energy consumption per bit of task processed on the edge server.
Note that we consider the return time of the task execution result as negligible.

E. Problem formulation

According to the previous notations, at time t_i, the total cost J of processing the current m_{th} task of N IoT devices is expressed as (8)

$$J(x_{nm}) = \sum_{n=1}^{N}[E_n^{loc}(1-x_n) + E_n^{edg}x_n + w_n(x_n T_n^{edg} + (1-x_n)T_n^{loc})] \tag{8}$$

where x_n represents the execution choice of the current task; $x_n = 0$ for local execution and $x_n = 1$ for edge computing and w_n is a priority parameter for energy or computing power. Extending the cost J to the discrete time scale D_t yields the overall cost of executing all M IoT device tasks. Minimizing this cost with constraints on energy consumption, computation time and variable x_{nm}, defines the optimization problem (9) related to the IoT system depicted in figure 1

$$\begin{aligned} min \quad & J(x_{nm}) = \sum_{n=1}^{N}[\sum_{m=1}^{M}[E_{nm}^{loc}(1-x_{nm}) + \\ & E_{nm}^{edg}x_{nm} + w_n(x_{nm}T_{nm}^{edg} + (1-x_{nm})T_{nm}^{loc})]] \\ & \text{s.t.} \quad E_{nm}^{loc} > 0 \\ & \quad T_{nm}^{loc} > 0 \\ & \quad E_{nm}^{edg} > 0 \\ & \quad T_{nm}^{edg} > 0 \\ & \quad x_{nm} \in \{0,1\} \end{aligned} \tag{9}$$

All the notations useful for understanding the formulation of the problem are shown in table I. To solve the above problem, we introduce a resolution technique based on deep reinforcement learning. This technique is presented in the next section.

IV. DRL Model

We solve the problem of task offloading (9) using a deep reinforcement learning (DRL) agent. Our agent is based on a double deep Q-Network. This structure constitutes the state of the art in reinforcement learning [28].
The DRL agent is in charge of mapping between the state of the IoT system resources and the optimal execution choices. To do so, it learns the policy defined by (10).

$$\pi : s_t \rightarrow X^N \tag{10}$$

where $s_t = \{v_i\}, i \in \{1, 2, 3, ...\}$ with v_i values characterizing the current state of the IoT system resources. We detail the

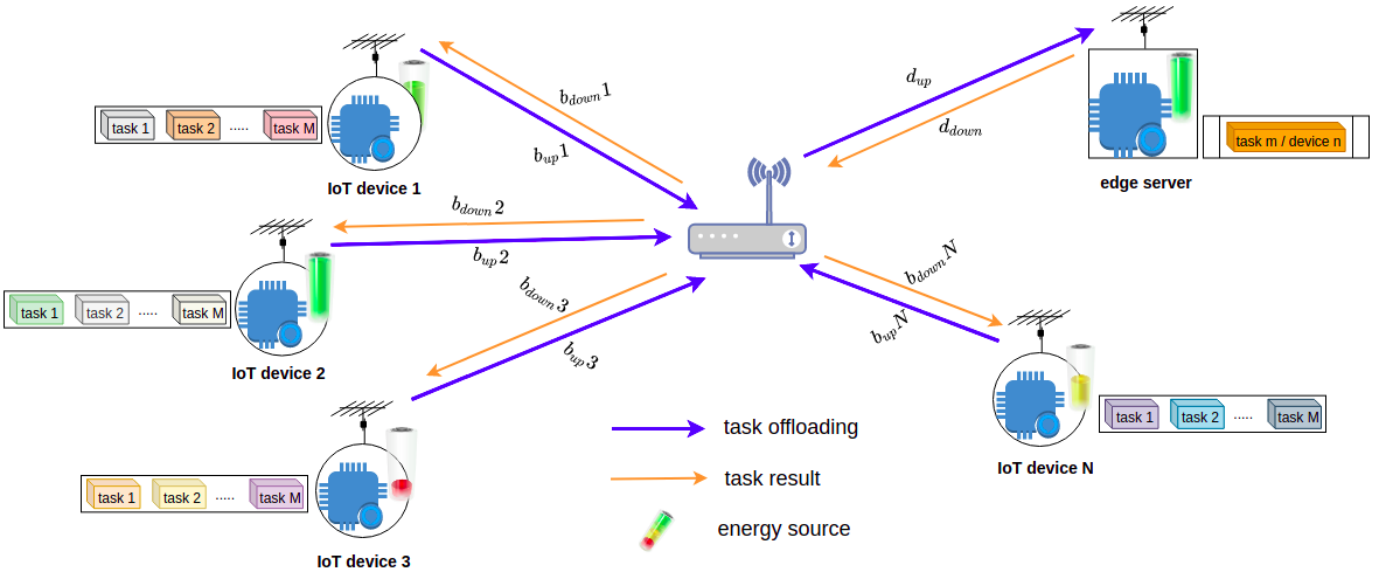

Fig. 1. IoT system

S_n	size of the task in terms of transferable data
D_n	size of the task expressed in terms of the total number of CPU cycles required to perform it
T_n^{loc}	local computing delay
T_n^{edg}	cost in time of an execution on the edge server when the task is offloaded
E_n^{loc}	energy consumed in performing the task locally
E_n^{edg}	energy consumed by running the task on the edge server
f_n^{loc}	number of CPU cycles required to process one bit of the task locally
f_n^{edg}	number of CPU cycles required to process one bit of the task on the edge server
e_n^{loc}	energy consumed to process one bit of the task locally
e_n^{edg}	energy consumed to process a fragment of the task on the edge server
d_n	average available bandwidth between the access point AP and the edge server
b_n	average available bandwidth between the IoT device and the access point AP
θ	definition parameter of proportionality between task size and number of CPU cycles required
N	number of IoT devices with tasks to perform in the IoT system
M	number of tasks per IoT device
x_{nm}	decision to offload the m_{th} task from the n_{th} IoT device 0 = local computing 1 = edge computing

TABLE I
NOTATIONS OF THE OPTIMIZATION PROBLEM

v_i values in the following sections. And $X^N = \{x_n\}, n \in \{1, 2, ..., N\}$ where x_n is the current task execution decision of the n_{th} IoT device. From a reinforcement learning perspective, s_t and X^N represent the state space and the action space, respectively. The π policy of mapping between these two spaces is exploited at each t_i. We associate a reward r_t with each of the x_n values of X^N. The reward indicates to the agent whether the decided action was right or not. This reward is based on an evaluation at time t_i of the cost J and represents an important part of the efficiency of the agent, especially since it ensures its convergence. We express this reward function as in (11).

$$\begin{aligned} &\texttt{If } J(x_{nm}) < J(x_{nm,alt}) \\ &\quad r(t) = -1 \\ &\texttt{Else If } J(x_{nm}) > J(x_{nm,alt}) \\ &\quad r(t) = +1 \\ &\texttt{Else} \\ &\quad r(t) = 0 \end{aligned} \tag{11}$$

where $J(x_{nm,alt})$ represents the cost J for alternative action. We denote by e_t (12) the tuple containing the state of the environment s_t, the action $a_t \in X^N$ performed from this state, the reward r_{t+1} given to the agent at time $t+1$ following the previous state-action pair (s_t, a_t), and the next state of the environment s_{t+1} . This tuple indeed gives us a summary of the agent's experience at time t_i.

$$e_t = (s_t, a_t, r_{t+1}, s_{t+1}) \tag{12}$$

Figure 2 shows the structure of the agent and the learning process based on experiences. The successive experiments are indeed stored in the replay memory. At each time t_i, a sample is then taken from this memory for training. The structure is composed of two neural networks, the evaluation network and the target network. In the Double Deep Q-Network algorithm as presented in [28], the evaluation network is used only to select the optimal action $a_t \in X^N$ and the target network to generate the right output labels Q^{lab} for neural network training. The generation of labels can be summarized by the Bellman's equation (13) where θ_t and θ'_t represent respectively the weights of the evaluation network and the target network.

$$Q^{lab} = r_t + \gamma Q(s_{t+1}, arg\max Q(s_t, a_t; \theta_t); \theta'_t) \tag{13}$$

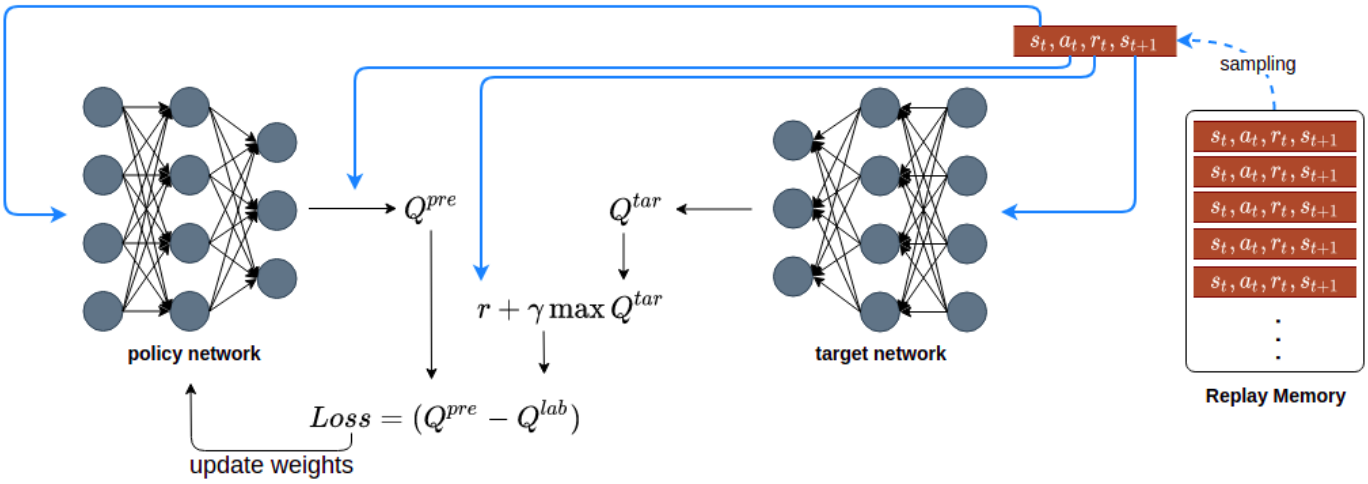

Fig. 2. Double Deep Q-Network-based agent structure

We provide in the algorithm 1 the pseudo-code for the DRL agent learning policy π.

Algorithm 2 Pseudo-code of DRL agent learning policy π

```
Initialize replay memory capacity
Initialize the policy network with random weights
Clone the policy network, and call it the target network
For each episode:
    Initialize the starting state
    For each time step:
        Select an action
            Via exploration ou exploitation
        Execute selected action in an emulator
        Observe reward and next state
        Store experience in replay memory
        Sample random batch from replay memory
        Preprocess states from batch
        Pass batch of preprocessed states to policy network
        Calculate loss between output Q-values (Q^pre) and
        target Q-values (Q^tar)
            Requires a pass to the target network for the next
            state
        Gradient descent updates weights in the policy net-
        work to minimize loss
        After time steps, weights in the target network are
        updated to the weights in the policy network
```

V. Simulation Tool

To implement the proposed scenario of the IoT system in figure 1 and to validate the effectiveness of the DRL agent, we use the ns3-gym framework [13]. ns3-gym is a toolkit that consists of two software components: Environment Gateway written in C++ and the Environment Proxy written in Python. Both components are add-ons to the existing ns-3 and OpenAI Gym frameworks as depicted in figure 3. The main objective is to facilitate and shorten the time required for prototyping new RL-based networking solutions. In the framework, the ns-3 simulator implements environments, while the OpenAI Gym unifies their interface. As ns-3 and OpenAI Gym are some already existing frameworks, the implementation of a generic interface between OpenAI Gym and ns-3 allows seamless integration of these two frameworks. That generic interface, figure 3, interconnects ns-3 network simulator and OpenAI Gym framework by transferring states and actions between the Gym agent and the simulation environment. The main strength of this framework is that it makes it possible to turn any ns-3 scenario into an OpenAI Gym compatible environment and use the flexibility of RL agent implementation in OpenAI Gym to interact with the ns-3 scenario. We implement our IoT system scenario in ns-3.

A. NS3-based environment

We classify the nodes in the IoT system in figure 1 into three categories. The first one is the set of IoT devices having tasks to execute. Each of these devices can execute a task locally with dynamic f_n^{loc} CPU cycles per task fragment; this generates an energy consumption of e_n^{loc} per executed task bit. The total cost of a task execution by the IoT device is equal to the execution time T_n^{loc} added to an energy consumption E_n^{loc}. Figure 4 illustrates the template used to instantiate each of the IoT devices. The second category is the edge server, which is able to receive, execute and return the execution result of tasks sent to it. Each received task is executed with dynamic f_n^{edg} CPU cycles per bit with an energy consumption per bit of e_n^{loc}. An execution on the server generates an execution time of T_n^{edg} and an overall energy consumption of E_n^{edg}. The third category includes the wireless access point AP. It was modeled using a standard ns-3 simulator node implementing the TCP/IP protocol stack and capable of forwarding tasks from IoT devices to the edge server and vice versa.
Note the state of the entire IoT system is a composed of the states of its nodes. The ns-3 simulator provides node state

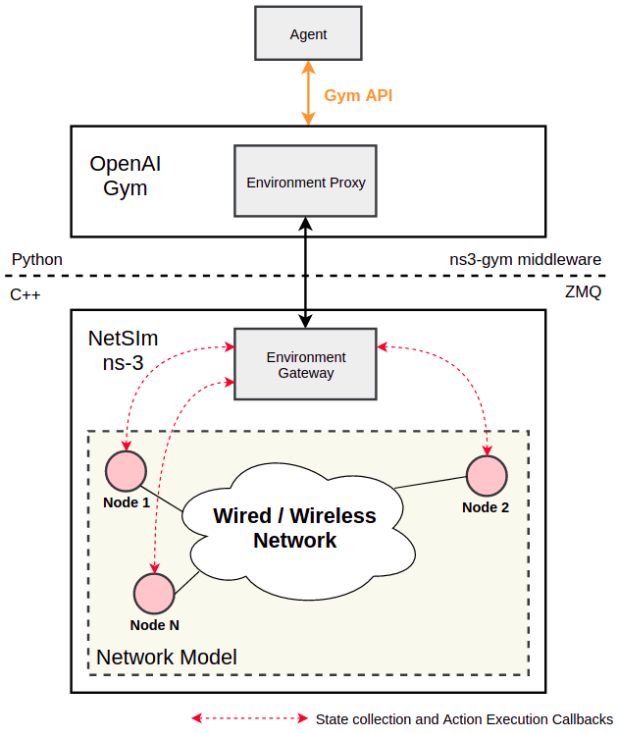

Fig. 3. ns3-gym framework architecture

retrieval functions by default. We made use of them to speed up the characterization of the state s_t of the IoT system.

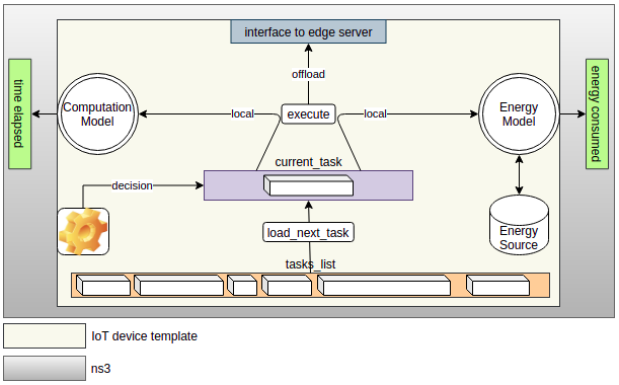

Fig. 4. Template for instantiating IoT devices

B. OpenAI Gym-based agent

Using the ns3-gym framework, we implement the deep reinforcement learning agent. With ns3-gym we can work with any learning framework. Thus, in order to build the Deep Q-Networks, we use tensorflow 2.2.0-rc1. We then pre-processed the raw data of the s_t states using scikit-learn 0.24.1 and numpy 1.20.1.

We implemented the replay memory using the data structure *deque* of the Python library *collections* [12]. *Deque* is a generalization of stacks and queues: it is possible to add and remove elements from both ends. This data structure handles multiple, memory-efficient additions and deletions with approximately the same performance *O(1)* in both directions.

In the next section, we will provide details about the Deep Q-Networks hyper parameters and the size of the data structure used.

VI. Experimentation

We use the ns-3 simulator to implement the IoT system. This simulator provides state retrieval functions for each node. To make explicit the v_i values of the s_t state of the IoT system, we noted, among the metrics recoverable from ns-3, those that change at each time t_i and thus can characterize the state of the resources during the same time. We realize that considering an IoT subsystem consisting of the n_{th} IoT device, the AP wireless access point and the edge server, eight values manage to characterize it: the execution capacity per task bit on the IoT device, the energy consumption per task bit on the IoT device, the size of the task to be executed, the packet transmission rate of the IoT device, the available bandwidth between the IoT device and the AP wireless access point, the available bandwidth between AP wireless access point and the edge server, the execution capacity per task bit on the edge server, and the energy consumption per task bit on the edge server. Those values constitutes the state $s_t = \{v_i\}$, with $i \in \{1, 2, ..., 8\}$.

In the Deep Q-Network implementation of our DRL agent structure, we use *Dense* neural networks with an input layer, three hidden layers and an output layer of 8, 120, 50, 50 and 2 neurons respectively. We use the Adam optimizer with lr=0.001 for back-propagation of neural networks with a loss function *mse*.

A. Convergence performance

Unlike supervised learning techniques, in reinforcement learning there is no distinction between the training phase and the test phase. Thus, one of the most used methods to measure performance is to follow the evolution of the rewards accumulated by the agent during several simulations. Each simulation is then considered as an episode. We consider $M = 29$ computational tasks per IoT device and a performance condition of our DRL agent when it perfectly realizes a good execution of about 28 tasks during 30 successive episodes. The simulations conducted generated the cumulative reward curve illustrated in figure 5.

We see that at the beginning of the simulations, the agent, less efficient, accumulates rewards between 1 and 26 during the first 20 episodes. From the $20th$ episode, a global convergence is observed until the $40th$ episode. This is explained by the beginning of self-learning thanks to the experiences stored in the replay memory. The filling of this memory from the $110th$ episode onwards has allowed an efficient learning of the agent implying a stability of the rewards accumulation. The defined stopping condition is thus satisfied, translating the convergence of our DRL agent.

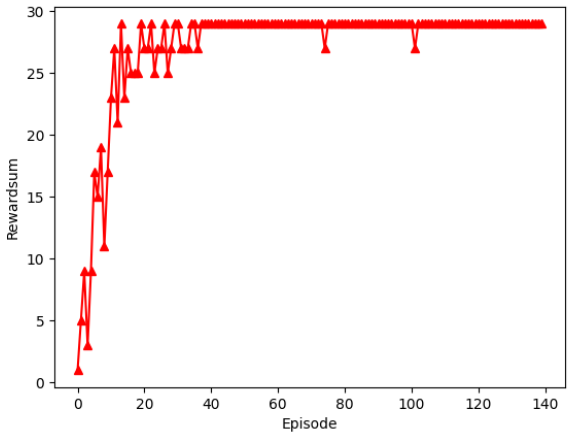

Fig. 5. Cumulative reward curve for the DRL agent

B. Scalability

To measure the scalability of our solution, we vary the number N of IoT devices and observe the time required for the DRL agent to apply the π offloading policy to each IoT device. The results obtained for this batch of simulations can be seen in figure 6. We see that for a number of IoT devices

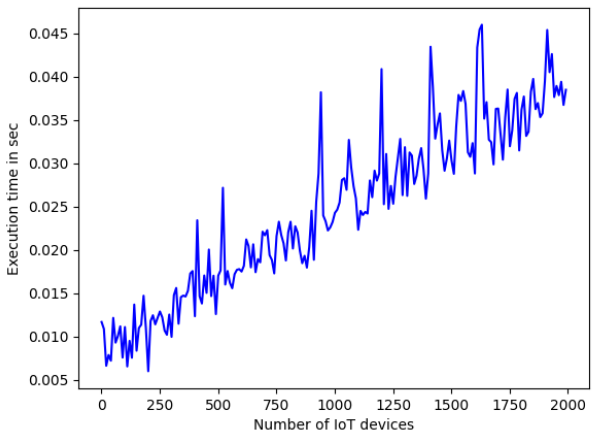

Fig. 6. Decision making time vs number of IoT devices

of about 500, the average time of application of the policy and thus of generations of offloading decisions by the agent is $20ms$. And even multiplying this number by 4, which is conceivable at the scale of large IoT system, we obtain an average application time less than $45ms$, showing that our proposed DRL agent scale quite well with the increase of IoT devices involved.

C. Computing performance

In previous simulations, we evaluate the gain in task execution time over existing execution strategies namely always local execution - strategy 1 - (on the IoT device) and always execution on the edge server - strategy 2. We define the optimal execution time T_{opt} of formula (14) as the best feasible execution time. We make explicit the execution time formulas for strategy 1, strategy 2 and our solution by equations (15), (16) and (17) respectively.

$$T_{opt} = \sum_{m=1}^{M} [\min(T_n^{loc}, T_n^{edg})] \tag{14}$$

$$T_{strg1} = \sum_{m=1}^{M} [T_n^{loc}] \tag{15}$$

$$T_{strg2} = \sum_{m=1}^{M} [T_n^{edg}] \tag{16}$$

$$T_{agt} = \sum_{m=1}^{M} [(1 - x_{nm})T_n^{loc} + x_{nm}T_n^{edg}] \tag{17}$$

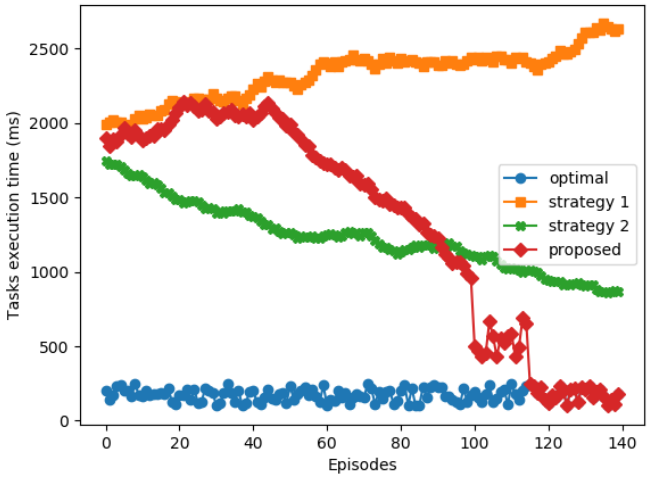

Fig. 7. Task execution time device-only vs edge-only vs agent

The plot of these curves is shown in figure 7. We can see that although at the beginning the agent is not performing very well almost as good as executing all tasks locally, it ends up being in phase with the optimal solution after about 110 episodes outperforming by far the solution of all offloading to the edge server. This is due to the self-adapting ability of the DRL agent. It is dynamic and develops the best offloading strategy.

VII. Conclusion

In this paper, we used a Deep Reinforcement Learning agent in order to solve the task offloading problem in autonomous IoT systems. Our simulations were done using the framework ns3-gym which make it possible to simulate very realistic IoT environments while deploying efficient DRL agents in the Gym framework. Our results show that this technique is

very promising and scale very well in large IoT environments such as smart cities. The architecture deployed is based on a centralized DRL agent handling task offloading requests from IoT devices to edge servers. In our future work, we would like to generalize the task offloading to horizontal and vertical collaborations, making it possible to offload tasks between IoT devices besides offloading to servers. A more distributed DRL architecture based on multi-agents collaboration in different parts of the autonomous IoT system will also be explored.

REFERENCES

[1] Hyame Assem Alameddine, Sanaa Sharafeddine, Samir Sebbah, Sara Ayoubi, and Chadi Assi. Dynamic task offloading and scheduling for low-latency iot services in multi-access edge computing. *IEEE Journal on Selected Areas in Communications*, 37(3):668–682, 2019.

[2] Ibrahim Alghamdi, Christos Anagnostopoulos, and Dimitrios P. Pezaros. On the optimality of task offloading in mobile edge computing environments. In *2019 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6, 2019.

[3] Saif Aljanabi and Abdolah Chalechale. Improving iot services using a hybrid fog-cloud offloading. *IEEE Access*, 9:13775–13788, 2021.

[4] Majid Altamimi. *A Task Offloading Framework for Energy Saving on Mobile Devices using Cloud Computing*. PhD thesis, University of Waterloo, 2014.

[5] Min Chen and Yixue Hao. Task offloading for mobile edge computing in software defined ultra-dense network. *IEEE Journal on Selected Areas in Communications*, 36(3):587–597, 2018.

[6] Weiwei Chen, Dong Wang, and Keqin Li. Multi-user multi-task computation offloading in green mobile edge cloud computing. *IEEE Transactions on Services Computing*, 12(5):726–738, 2019.

[7] Xianfu Chen, Honggang Zhang, Celimuge Wu, Shiwen Mao, Yusheng Ji, and Medhi Bennis. Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning. *IEEE Internet of Things Journal*, 6(3):4005–4018, 2019.

[8] Xianfu Chen, Honggang Zhang, Celimuge Wu, Shiwen Mao, Yusheng Ji, and Mehdi Bennis. Performance optimization in mobile-edge computing via deep reinforcement learning. In *2018 IEEE 88th Vehicular Technology Conference (VTC-Fall)*, pages 1–6, 2018.

[9] Sukjin Choo, Joonwoo Kim, and Sangheon Pack. Optimal task offloading and resource allocation in software-defined vehicular edge computing. In *2018 International Conference on Information and Communication Technology Convergence (ICTC)*, pages 251–256, 2018.

[10] Chongwu Dong and Wushao Wen. Joint optimization for task offloading in edge computing: An evolutionary game approach. *Sensors*, 19(3):1–23, 20.

[11] Mohamed El Ghmary, Tarik Chanyour, Youssef Hmimz, and Mohammed Ouçamah Cherkaoui Malki. Efficient multi-task offloading with energy and computational resources optimization in a mobile edge computing node. *International Journal of Electrical and Computer Engineering*, 9(6):4908–4919, 2019.

[12] Python Software Foundation. deque.

[13] Piotr Gawłowicz and Anatolij Zubow. Ns-3 meets openai gym: The playground for machine learning in networking research. In *Proceedings of the 22nd International ACM Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, MSWIM '19, page 113–120, New York, NY, USA, 2019. Association for Computing Machinery.

[14] Yixue Hao, Yingying Jiang, Tao Chen, Donggang Cao, and Min Chen. itaskoffloading: Intelligent task offloading for a cloud-edge collaborative system. *IEEE Network*, 33(5):82–88, 2019.

[15] Md. Sajjad Hossain, Cosmas Ifeanyi Nwakanma, Jae Min Lee, and Dong-Seong Kim. Edge computational task offloading scheme using reinforcement learning for iiot scenario. *ICT Express*, 6(4):291–299, 2020.

[16] Liang Huang, Suzhi Bi, and Ying-Jun Angela Zhang. Deep reinforcement learning for online computation offloading in wireless powered mobile-edge computing networks. *IEEE Transactions on Mobile Computing*, 19(11):2581–2593, 2020.

[17] Mingfeng Huang, Wei Liu, Tian Wang, Anfeng Liu, and Shigeng Zhang. A cloud–mec collaborative task offloading scheme with service orchestration. *IEEE Internet of Things Journal*, 7(7):5792–5805, 2020.

[18] Abdullah Lakhan and Xiaoping Li. Mobility and fault aware adaptive task offloading in heterogeneous mobile cloud environments. *ICST Transactions on Mobile Communications and Applications*, 5:159947, 01 2019.

[19] Chen-Feng Liu, Mehdi Bennis, Mérouane Debbah, and H. Vincent Poor. Dynamic task offloading and resource allocation for ultra-reliable low-latency edge computing. *IEEE Transactions on Communications*, 67(6):4132–4150, 2019.

[20] Chen-Feng Liu, Mehdi Bennis, and H. Vincent Poor. Latency and reliability-aware task offloading and resource allocation for mobile edge computing. In *2017 IEEE Globecom Workshops (GC Wkshps)*, pages 1–7, 2017.

[21] Sudip Misra and Niloy Saha. Detour: Dynamic task offloading in software-defined fog for iot applications. *IEEE Journal on Selected Areas in Communications*, 37(5):1159–1166, 2019.

[22] Mithun Mukherjee, Suman Kumar, Mohammad Shojafar, Qi Zhang, and Constandinos X. Mavromoustakis. Joint task offloading and resource allocation for delay-sensitive fog networks. In *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, pages 1–7, 2019.

[23] Samrat Nath and Jingxian Wu. Deep reinforcement learning for dynamic computation offloading and resource allocation in cache-assisted mobile edge computing systems. *Intelligent and Converged Networks*, 1(2):181–198, 2020.

[24] Salman Raza, Wei Liu, Manzoor Ahmed, Muhammad Rizwan Anwar, Muhammad Ayzed Mirza, Qibo Sun, and Shangguang Wang. An efficient task offloading scheme in vehicular edge computing. *Journal of Cloud Computing*, 9, 2020.

[25] Jinke Ren, Guanding Yu, Yunlong Cai, and Yinghui He. Latency optimization for resource allocation in mobile-edge computation offloading. *IEEE Transactions on Wireless Communications*, 17(8):5506–5519, 2018.

[26] Yuxuan Sun, Xueying Guo, Jinhui Song, Sheng Zhou, Zhiyuan Jiang, Xin Liu, and Zhisheng Niu. Adaptive learning-based task offloading for vehicular edge computing systems. *IEEE Transactions on Vehicular Technology*, 68(4):3061–3074, 2019.

[27] Sowndarya Sundar and Ben Liang. Offloading dependent tasks with communication delay and deadline constraint. In *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, pages 37–45, 2018.

[28] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning, 2015.

[29] Jin Wang, Jia Hu, Geyong Min, Albert Y. Zomaya, and Nektarios Georgalas. Fast adaptive task offloading in edge computing based on meta reinforcement learning. *IEEE Transactions on Parallel and Distributed Systems*, 32(1):242–253, 2021.

[30] Yi Yang, Yeli Geng, Li Qiu, Wenjie Hu, and Guohong Cao. Context-aware task offloading for wearable devices. In *2017 26th International Conference on Computer Communication and Networks (ICCCN)*, pages 1–9, 2017.

[31] Zhaohui Yang, Cunhua Pan, Jiancao Hou, and Mohammad Shikh-Bahaei. Efficient resource allocation for mobile-edge computing networks with noma: Completion time and energy minimization. *IEEE Transactions on Communications*, 67(11):7771–7784, 2019.

[32] Bingxin Zhang, Guopeng Zhang, Weice Sun, and Kun Yang. Task offloading with power control for mobile edge computing using reinforcement learning-based markov decision process. In *Mobile Information Systems*, pages 1–6, 2020.

[33] C. Zhang, Bo Gu, Zhi Liu, K. Yamori, and Y. Tanaka. Cost- and energy-aware multi-flow mobile data offloading using markov decision process. *IEICE Trans. Commun.*, 101-B:657–666, 2018.

[34] Cheng ZHANG, Zhi Liu, Bo Gu, Kyoko Yamori, and Yoshiaki TANAKA. A deep reinforcement learning based approach for cost- and energy-aware multi-flow mobile data offloading. *IEICE Transactions on Communications*, E101.B, 01 2018.

[35] Tianyu Zhang, Yi-Han Chiang, Cristian Borcea, and Yusheng Ji. Learning-based offloading of tasks with diverse delay sensitivities for mobile edge computing. In *2019 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6, 2019.

[36] Zhenjiang Zhang, Chen Li, ShengLung Peng, and Xintong Pei. A new task offloading algorithm in edge computing. *EURASIP Journal on Wireless Communications and Networking*, 2021(1):17, Jan 2021.