Resilient deep reinforcement learning architecture for task offloading in autonomous IoT systems

Abdel Kader Chabi Sika Boni, Youssef Hablatou, Hassan Hassan, Khalil Drira

Resilient deep reinforcement learning architecture for task offloading in autonomous IoT systems

Abdel Kader Chabi Sika Boni, Youssef Hablatou, Hassan Hassan and Khalil Drira
LAAS-CNRS, Université de Toulouse, CNRS, UPS, Toulouse, France
{akchabisik, yhablatou, hhassan, khalil}@laas.fr

***Abstract*—Autonomous IoT systems require the development of good automation algorithms capable of handling a huge number of IoT devices such as in smart cities. Deep Reinforcement Learning (DRL) is a powerful automation technique that can be used in massive systems thanks to its ability to deal with big state spaces. Moreover, it adapts quickly to changes in the system by reinforcement learning, making the automation algorithm very flexible. However, using DRL relies generally on centralized agent architecture making it more exposed to communication failures. In this paper, we propose a distributed architecture to solve the task offloading problem in autonomous IoT systems where learning is achieved in a master agent while decision making is delegated to IoT devices. This architecture is more resilient as decisions are made locally and interactions between IoT devices and the master agent are less frequent and not blocking. We tested this architecture in the ns3-gym environment and our results show very good resilience of this architecture.**

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

I. Introduction

IoT devices are widely present in modern systems and are used in urban areas, for agriculture, in industry... The advent of autonomous IoT systems is a natural evolution of the increasing use of IoT device in all aspects of our modern life. However, the development of autonomous IoT systems is conditioned by the ability of providers to propose automation techniques capable of handling their complexity and managing their resources with minimum human interaction. In addition, these techniques should be able to scale and handle millions of devices in massive IoT systems. Deep Reinforcement Learning (DRL) is one of those new techniques capable of dealing with huge space states while offering powerful solutions to meet the challenges raised by autonomous IoT systems. But DRL implementations are often based on centralized architectures where an agent gathers observation from the whole system, trains an neural network, and takes actions before transmitting them to local entities. While this architecture offers many advantages such as providing enough computing power for learning, it suffers from the position of the central agent as a single point of failure (SPOF). Any communication problem between the agent and local entities can seize up the whole system.

In this paper, we propose a distributed architecture for DRL in order to solve the task offloading problem in autonomous IoT systems. Task offloading is a technique used to improve the efficiency of a computational task by delegating that task completely or partially to a remote entity [2]. Task offloading requires collaboration between IoT devices and remote servers (edge or cloud). Delegating the tasks generated by IoT devices can be ordered if resources are missing on the device and the cost of executing these tasks remotely is cheaper. However, numerous parameters should be taken into account, such as the bandwidth available to transmit tasks, the computation capacity of remote servers, and the energy cost of the operation to the device. We use a distributed DRL architecture to solve this problem. Our solution offers optimal task offloading algorithm, capable of handling big systems with very good scalability and moreover a robust decision making mechanism close to the devices.

The rest of this paper is organized as follows: in section 2 we discuss related work, in section 3 we present an overview of the problem, in section 4 we describe the proposed architecture and in section 5 we present evaluation results, finally we conclude in section 6 with future work.

II. Related work

Many recent works tried to provide solutions to the problem of efficient task offloading in different contexts. In [8], a three-layer infrastructure - vehicular fog (VF), fog server (FS) and central cloud (CC)- is presented in which a vehicular requester located in the VF has the option to send all or part of its task (divisible task) to: i) a vehicular server located in the same VF; ii) one of the fixed fog servers located along the route or; iii) the cloud. Considering the task processing time, the energy consumed and the financial cost of using the resources, a mathematical formulation based on probabilities of choosing an execution option is proposed and solved by the ADMM-PSO algorithm that transforms the problem into unconstrained sub-problems. Zhao et al. [16] adopted a similar approach to [8] focusing only on task offloading from one vehicle to another via a Multi-access Edge Computing (MEC) server. They solve their formulation by the Newton iteration algorithm. They were able to demonstrate a minimal average delay of their system as well as a lower probability of exceeding the deadline of a task.

In cellular networks, as in [10], strategies of task offloading are studied. Authors considered energy constrained smart devices (SDs) all connected to a base station attached to a MEC platform with servers. The tasks arriving to the SDs are put in their respective buffers for i) local processing or ii) processing on the MEC servers. They formulated the problem to minimize the overall end-to-end network delay, to ensure stability of

the SDs and MEC servers buffers and to minimize the energy consumption of the SDs. They solve it using an online version of Bandit Optimization, a technique that is not adapted for online scenarios with time-varying resources.

To cope with scenarios having high time-varying resources, other works use deep learning techniques. In [13] a collaboration between a cloud server and an edge server is set where tasks are received from mobile devices (MDs). An MD makes a double decision about each task to be executed: i) process it locally or offload it, ii) in case of offloading, send it to the edge server or to the cloud. The authors introduced a deep learning based optimization where S neural networks take as input information about the tasks and the available resources and generate binary decisions used to evaluate the formulated multi-objective optimization function. A database stores these evaluations and is used to train each of the neural networks. Although it isn't explicitly stated in [13], this approach is very similar to the principle of reinforcement learning. Unfortunately, the authors do not give any information about where the training will be done or where the database will be stored. Since the neural networks share the same database, it is difficult to have a distributed architecture at the risk of over-consuming bandwidth due to the transit of data on the network.

Zhou et al. [17] propose a distributed task offloading approach where a unidirectional (i.e. source-to-destination) and permanent offloading session is set up between pairs of devices via a device-to-device (D2D) communication in the coverage area of a base station. Having time divided into time-slots of fixed $\Delta\tau$ lengths, the source device decides at beginning of each time-slot whether to send its current task to the destination device or to execute it locally, depending on the quality of the allocated bandwidth. Authors adopted a game-theoretic approach using an improved version of Lagrangian Optimization embedded in each device allowing it to make the offloading decision autonomously. However, in this approach, not only all devices make the intense computations of the Lagrangian Optimization but in case of loss of the session, there is no possibility to offload tasks. The distributed aspect is also explored in [15] where authors propose TODG, a distributed online task offloading algorithm. The edge servers used in their approach deploy three types of virtual machines (VM). Each VM handles a specific task from a device. Thus, devices are also subdivided into 3 categories, each generating tasks executed by a type of VM. The communication between the devices and edge servers is ensured by a wireless link with L channels. The problem is formulated with the objective of maximizing the system long-term utility while satisfying the worst-case delay constraints. It is decomposed into sub-problems solved by TODG where each device computes and finds the worst-case queuing delay and sends it to one server. Servers then communicate between themselves to allocate appropriate channels to devices who have just to set their transmission rate and choose a server. The work in [7] has a similar approach for task assignment in Industrial IoT using classic Q-learning algorithm, but it neglects the distributed aspect. Besides, numerous calculations are imposed to energy constrained devices.

To automate task offloading, several works adopt the Deep Reinforcement Learning technique. In [1], to maximize the completed tasks number in a timely manner and minimize the power consumption of user devices, a centralized deep reinforcement learning agent based on double deep Q-network (DDQN) [12] is proposed. It retrieves the network state, communicates with MEC servers to obtain their workload levels and with devices to obtain their offload task profiles then decides which task should be executed by which MEC server as well as the computational frequency to be allocated to each task. It should be noted that the consistency of the whole model relies on the centralized agent which should never fall down otherwise the whole offloading system stops and no more tasks are processed.

With DMRO, [9], Guanjin et al. employ a centralized reinforcement learning system consisting of two models: the first is used to track the dynamic of the MEC environment; the second to generate offloading decisions for tasks received from IoT devices. The training of the agent is based on s parallel dense neural networks (DNN). However, there is no mention of the node that runs this agent which makes it difficult to deploy their solution in a real scenario as training multiple parallel DNNs requires a lot of computing resources.

Most DRL algorithms have a centralized operating mechanism. This constitutes a certain additional latency due to the need to transmit observations and wait for offloading decisions to be returned. A distributed DRL approach has therefore been considered in [6] and [18]. Goudarzi et al. [6] use a distributed version of Advantage Actor Critic (A2C) [14]. The proposed architecture includes three layers (IoT, Edge and Cloud layers). Access Points -brokers- located in Edge layer allow IoT devices to choose the servers on which the IoT application tasks should be executed. The actors are implemented on these brokers and the learner is centralized. During the training phase, each actor continuously sends pre-stored experiences to the learner that uses them to update its policy, and sharing it back with the actors. This approach forces all actors (brokers) to store their experiences in their internal buffers, thus considerably reducing the number of processed tasks. Moreover, actors being on brokers, it is not clear whether, for local executions, the task is first sent to the broker which sends it back to the IoT device, neither under which condition the device will execute its task locally without going through a broker.

In [11], a distributed DNN training is set where an edge server helps one or more mobile devices to train their embedded neural networks. That allows devices to have reliable way of choosing offloading decisions. Nevertheless, each training is done independently from others leading to an overload of edge nodes training two or more DNNs. Moreover, authors use LSTM layers in DNNs architectures making it necessary to have enough computing power on edge servers otherwise training becomes too long.

In our solution, we deal with the above mentioned problems

by: 1) implementing a cloud-based agent in charge exclusively of training and learning optimal task offloading policy; 2) embedding a neural network in each IoT device giving it ability to always take accurate offloading decisions internally; 3) allowing IoT devices to share their experiences less frequently in order to avoid energy waste due to data transmission; 4) copying weights and biases from cloud-based agent to neural networks in IoT devices so they can act without having to train locally; 5) providing a Generative Adversarial Network (GAN) to the cloud-based agent allowing it to grow its training database and hence mitigating the effects of less frequent sharing of experiences by IoT devices.

III. Overview

A. IoT environment

We consider an IoT system composed of N IoT devices connected wirelessly via a smart access point (SAP) to Z servers as shown in figure 1. Each IoT device has M computation tasks of different sizes, and can execute only one task at a time. For each task the IoT device has two possibilities: either to execute it locally or to offload it to a remote server (an edge server or the cloud). The offloading decision is based on the available capacity of the device, its energy level and the available bandwidth. Devices tasks are independent and the decision concerning one task cannot be changed during its execution.

B. Task offloading model

The tasks generated by IoT devices can be modeled by their data size S_i or the computational resources necessary for their execution D_i. These values are proportional $S_i = \theta * D_i$
The execution time of a task depends on the CPU processing delay. Let f be the number of CPU cycles required to process one bit. The execution time of a Task can be then expressed as in equation (1):

$$T_i = \frac{D_i}{f} \tag{1}$$

The value of f^{IoT} on an IoT device is generally greater than the value of f^{Server} on an edge/cloud server.
The energy consumption cost E_i to execute one task can be expressed as in equation (2):

$$E_i = D_i * e \tag{2}$$

Where e is the energy consumed to process one bit. This value is noted respectively e^{IoT}, e^{Server} for the IoT device, or an edge/cloud server.
When a task is offloaded, the transmission delay to the remote server $T_{i,t}^{Server}$ should be added to the processing delay on the remote server $T_{i,p}^{Server}$ to obtain the total execution time of the task.

$$T_i^{Server} = T_{i,t}^{Server} + T_{i,p}^{Server} \tag{3}$$

C. Problem statement

According to previous notations let the cost function of executing M tasks in N IoT devices with Z servers be $C(x_{mnz})$, this functions can be expressed as in equation (4):

$$C(x_{mnz}) = \sum_{z=1}^{Z}\sum_{n=1}^{N}\sum_{m=1}^{M}[\alpha_{mnz}(E_{mnz}^{IoT}(1-x_{mnz}) + E_{mnz}^{Server}x_{mnz}) + \beta_{mnz}(x_{mnz}T_{mnz}^{Server} + (1-x_{mnz})T_{mnz}^{IoT})] \tag{4}$$

x_{mnz} denotes the execution choice of the task; its value is 0 for local execution and 1 for remote computing. α_{mnz} and β_{mnz} are weighing factors. All variables in this equation are positive values. Minimizing this cost function $min(C(x_{mnz})$ defines the optimization problem to resolve in order to get the best offloading decision.

IV. Proposed architecture

The studied IoT system contains a huge number of IoT devices that interact with a big number of local edge servers and remote cloud server. Minimizing the cost equation (4) involves an important state space that can be efficiently resolved with a deep reinforcement learning approach. We implemented an algorithm based on double deep Q-network (DDQN) [12]. Usually the DDQN is implemented as a central agent. In such case, decision making will be centralized and constitutes a single point of failure (SPOF) of the solution. We choose to implement it differently using a distributed approach with centralized learning mechanism and localized decision making logic. Our solution offers a resilient architecture well adapted to massive IoT systems as explained in next sections.

A. Double deep Q-network

The processing of DRL algorithm is based on three variables: the observations sent from the environment to the agent as state values, the actions decides by the agent to apply to the environment and the reward that the agent will get as a consequence of these actions.

1) Observations: for each IoT device, the observations reflect the state of that device in its environment. It is composed of two parts:

- IoT device state: computation capacity per bit, energy consumption per bit, generated task size and link capacity between the device and the smart access point (SAP).
- Remote servers state: server computation capacity, energy consumption if applicable, and link capacity between the SAP and the server.

2) Actions: For each task, the agent will take a decision based on the transmitted observation, either to execute the task locally or on a remote server. Actions are applied to the ns-3 environment and new state observations are gathered.

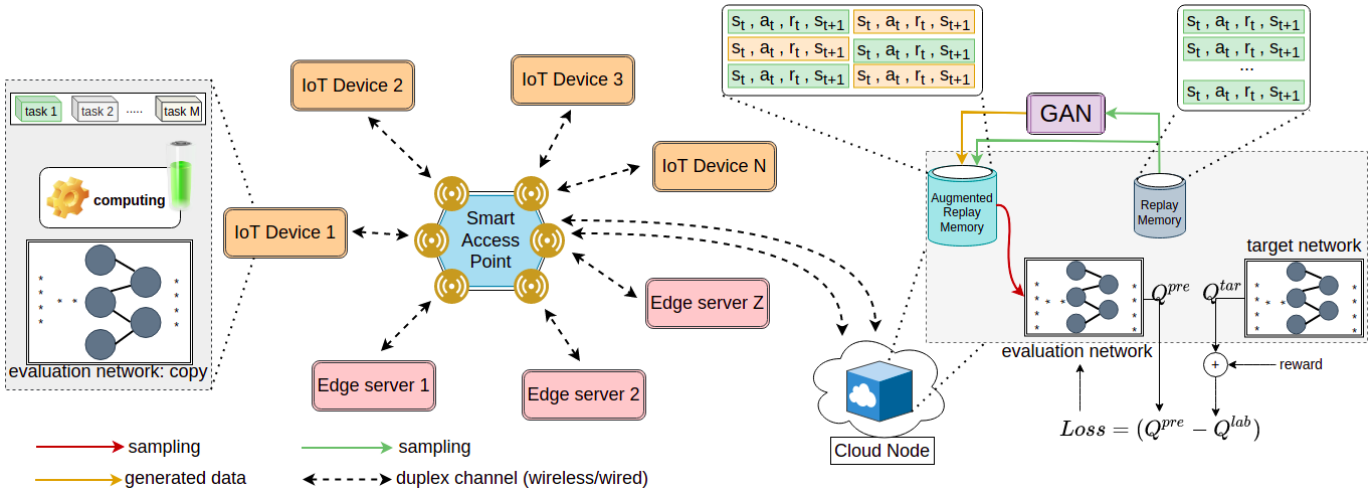

Fig. 1. IoT system

3) Reward function: Once the actions are applied, the environment measures the consequence of these actions and delivers a reward to the agent. The goal of the agent is to minimize the cost function defined in (4). We use the same function to quantify the reward. If the actions taken by the agent succeed to reduce the cost function, those actions need to be reinforced so the reward should be bigger. On the other hand, if the cost function increases, the reward must be smaller. In other words the reward is inversely proportional to the value of the cost function. It is defined in equation (5)

$$R = (-1) * C(x_{mnz}) \tag{5}$$

We implement this algorithm in two parts: cloud based-agent and IoT devices internal decision making mechanism.

B. Cloud-based agent

The cloud based-agent is responsible of 1) processing tasks sent to it and sending back results; 2) ensuring the agent's DDQN learning logic. The agent is equipped with basic replay memory (BRM) and augmented replay memory (ARM). It receives experiences from IoT devices and stores them into BRM. Then it uses tabular GAN [3] (a variant of GAN [5]) to generate more experiences which are stored together with original ones into ARM. The cloud agent focuses only on the DDQN neural networks training part. Hence, by using samples from ARM, the agent trains its neural networks through gradient descent algorithm. After T training iterations, the agent sends its evaluation network parameters to the SAP which in turn shares them with IoT devices. The architecture of the implemented agent is illustrated in figure 1 and we provide in algorithm 1 the pseudo-code for the DRL agent on the cloud.

C. IoT device internal decision mechanism

We embedded in each IoT device a deep neural network with the same architecture as the cloud-based agent's evaluation network. Each device is then capable of making offloading decisions locally. The device receives periodically from the SAP the state of the environment i.e. bandwidths, delays and servers computing levels. Those information combined with the task size are the input of the local neural network. When the decision taken is to compute locally, the task is processed locally using the device CPU with energy consumption as expressed in equations (1) and (2), otherwise, the task is sent to a remote server. Then the IoT device computes the reward based on equation (5) setting $M = 1$ and $N = 1$ as the reward is for current task generated in current device.

The device receives periodically the cloud agent's evaluation network parameters, then it updates its local neural network to enhance accuracy of future decisions. This way even if the connection is lost with cloud agent, the device still has the possibility to take offloading decisions. We provide in algorithm 2 the pseudo code of the internal decision mechanism.

D. Smart access point

The SAP plays an important role in the proposed autonomous IoT system. Not only it ensure connectivity between all devices, but it's also responsible for sending devices experiences to cloud agent and sharing back its evaluation network parameters with IoT devices. It tracks major changes in the system resources and notifies them to each IoT device. This notification includes loss of connection with a node. When an IoT device decides to offload its task to a server, the SAP ensures the task transmission to the chosen server and result retrieval. To strengthen the resilience of the system, we use a redundant connection between the SAP and the cloud agent.

V. Evaluation

A. Evaluation framework

We use the ns3-gym framework [4] to implement both our IoT system scenario and the DDQN algorithm. ns3-gym is a toolkit that introduces two middle-ware components provided as add-ons to the ns-3 network simulator and OpenAI Gym

Algorithm 1: Pseudo-code of cloud based-agent

Initialize BRM (basic replay memory) capacity
Initialize ARM (augmented replay memory) capacity
Initialize evaluation network with random weights
Clone evaluation network into target network
Initialize BS = batch Size
Initialize T = frequency of weights transfer to the SAP
Initialize t = number of training iterations
Continuously:
  If newExperiences received from smart acces point:
    Store newExperiences in BRM
  If size of BRM $>$ 0:
    Sample random batch from BRM
    Generate Experiences via Input of batch to tabGAN
    Store random batch + gen. Experiences into ARM
  If size of ARM $>=$ BS:
    Sample random batch from ARM
    Preprocess states from batch
    Pass preprocessed states to evaluation network
    Calculate loss between output (Q^{pre}) and target (Q^{tar}) Q-values
    Update evaluation network weights through Gradient Descent
    After time steps, copy weights of evaluation network to target network
    Set t = t + 1
  If t equals T:
    Send evaluation network weights to the SAP
    Set t = 0

Algorithm 2: Pseudo-code of internal decision mechanism

Initialize local network
Initialize SF = frequency of sending experience
Initialize cSF = counter for SF
Retrieve RES = servers + bandwidth resources level
Generate task
While battery level $>$ 0:
  If new weights received:
    Update local network weights
  Input RES+task size to local network
  Retrieve offloading decision from local network
  If decision equals local computing:
    Execute task locally
  If decision different from local computing:
    Send task+chosen server index to the SAP
  Generate new task
  If cSF equals SF:
    Generate R = reward based on equation (5)
    If received new RES:
      Update RES
    Send experience to the SAP
    Set cSF = 0
  Else
    Set cSF = cSF + 1

framework. These components ensure communication between the scenario implemented as an environment inside ns-3 and the learning agent implemented in OpenAI Gym. The environment provides state values (observations) while the agent calculates appropriate actions based on these observations. Applying the actions to the environment results in a reward to return to the agent to reinforce those actions or discard them.

The scenario is implemented in ns-3 while the agent is implemented OpenAI gym using Pytorch. The local copy of the evaluation network in IoT devices is implemented in C++ inside the ns-3 simulator.

B. Convergence

The convergence of the DRL agent is measured by the reward sum. As the goal of the agent is to minimize the cost function, its reward is inversely proportional to the cost. So during training when it converges its rewards will be better and the cumulative reward sum will increase until reaching a maximum indicating that the agent always takes the best decision. In figure 2 we show the reward sum curve for real data (no GAN is used) compared to using GAN with different frequencies. For example, when $GAN = 1/5$, that means the observations are sent from the environment only once of each five iterations. So the GAN compensates for the 4 missing entries. Our simulations show that for a GAN frequency greater than $1/10$ the agent does not converge anymore. Figure 2 depicts the evolution of error during training, using real data leads to fast convergence but using GAN we can reduce real data while the agent can always converge. Figure 3 illustrates the evolution of error during the learning phase of the agent.

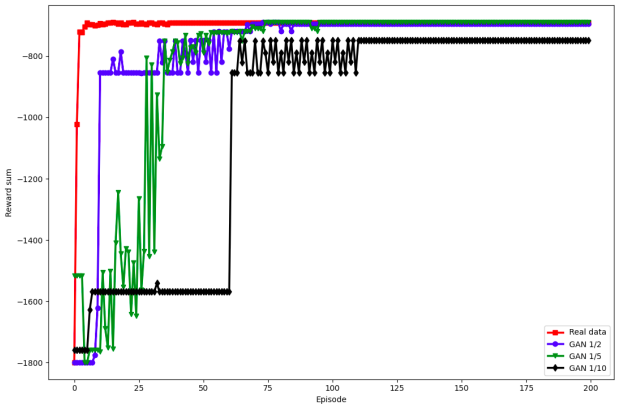

Fig. 2. Cumulative reward sum

C. Resilience

The advantage of our solution is that decisions are done locally by IoT devices. To illustrate the benefit of this architecture we compare the number of tasks that will be dropped in both architectures, normal centralized implementation and

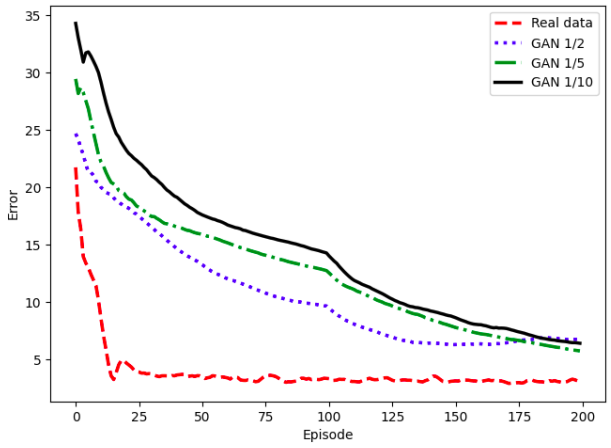

Fig. 3. Evolution of error during learning

our distributed implementation, against the DRL loop latency that represents the duration of communication failure between the agent and IoT devices. The result is shown in figure 4. While in a centralized architecture the number of dropped tasks explodes when the latency of the loop increases, using our solution IoT devices can decide autonomously and tasks are executed without loss.

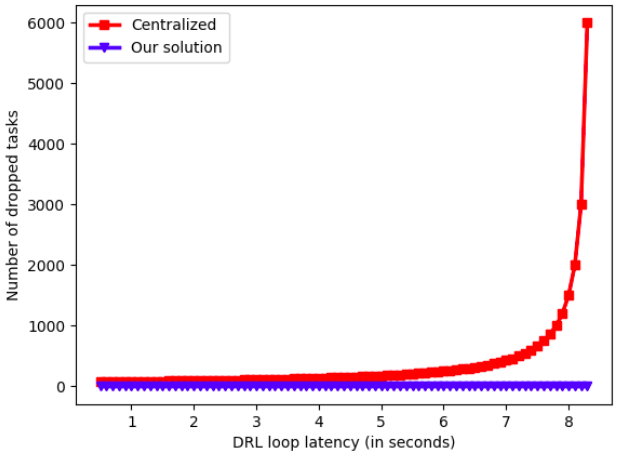

Fig. 4. Dropped tasks vs DRL loop latency

VI. Conclusion

In this paper we presented a distributed architecture to implement a deep reinforcement agent capable of solving the task offloading problem in an autonomous IoT system. Our solution can perform task offloading with an agent based on the cloud while task offloading decisions are made locally in IoT devices. It shows very good resilience to communication failures between the IoT devices and the agent. The execution of the agent on the cloud ensures efficient learning while copying the evaluation network in IoT devices allows local decision making. The use of GAN compensates for the missing data in the replay memory used to train the agent. In our future work, we would like to compare other DRL algorithms with the double deep Q-network and deploy the solution on a real platform.

References

[1] Laha Ale, Ning Zhang, Xiaojie Fang, Xianfu Chen, Shaohua Wu, and Longzhuang Li. Delay-aware and energy-efficient computation offloading in mobile-edge computing using deep reinforcement learning. *IEEE Transactions on Cognitive Communications and Networking*, 7(3):881–892, 2021.

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

[3] Insaf Ashrapov. Tabular gans for uneven distribution, 2020.

[4] 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.

[5] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[6] Mohammad Goudarzi, Marimuthu S Palaniswami, and Rajkumar Buyya. A distributed deep reinforcement learning technique for application placement in edge and fog computing environments. *IEEE Transactions on Mobile Computing*, pages 1–1, 2021.

[7] 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.

[8] Zongkai Liu, Penglin Dai, Huanlai Xing, Zhaofei Yu, and Wei Zhang. A distributed algorithm for task offloading in vehicular networks with hybrid fog/cloud computing. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, pages 1–14, 2021.

[9] Guanjin Qu, Huaming Wu, Ruidong Li, and Pengfei Jiao. Dmro: A deep meta reinforcement learning-based task offloading framework for edge-cloud computing. *IEEE Transactions on Network and Service Management*, 18(3):3448–3459, 2021.

[10] Zhenfeng Sun and Mohammad Reza Nakhai. An online learning algorithm for distributed task offloading in multi-access edge computing. *IEEE Transactions on Signal Processing*, 68:3090–3102, 2020.

[11] Ming Tang and Vincent W. S. Wong. Deep reinforcement learning for task offloading in mobile edge computing systems, 2020.

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

[13] Huaming Wu, Ziru Zhang, Chang Guan, Katinka Wolter, and Minxian Xu. Collaborate edge and cloud computing with distributed deep learning for smart city internet of things. *IEEE Internet of Things Journal*, 7(9):8099–8110, 2020.

[14] Yuhuai Wu, Elman Mansimov, Shun Liao, Roger Grosse, and Jimmy Ba. Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation, 2017.

[15] Sheng Yue, Ju Ren, Nan Qiao, Yongmin Zhang, Hongbo Jiang, Yaoxue Zhang, and Yuanyuan Yang. Todg: Distributed task offloading with delay guarantees for edge computing. *IEEE Transactions on Parallel and Distributed Systems*, 33(7):1650–1665, 2022.

[16] Haitao Zhao, Qixing Zhu, Yue Chen, and Yinyang Zhu. A research of task-offloading algorithm for distributed vehicles. In *2020 IEEE International Conference on Communications Workshops (ICC Workshops)*, pages 1–5, 2020.

[17] Jianshan Zhou, Daxin Tian, Zhengguo Sheng, Xuting Duan, and Xuemin Shen. Distributed task offloading optimization with queueing dynamics in multiagent mobile-edge computing networks. *IEEE Internet of Things Journal*, 8(15):12311–12328, 2021.

[18] Xiaoyu Zhu, Yueyi Luo, Anfeng Liu, Md Zakirul Alam Bhuiyan, and Shaobo Zhang. Multiagent deep reinforcement learning for vehicular computation offloading in iot. *IEEE Internet of Things Journal*, 8(12):9763–9773, 2021.