Sustainable Task Offloading in UAV Networks via Multi-Agent Reinforcement Learning

Alessio Sacco, *Student Member, IEEE*, Flavio Esposito, *Member, IEEE*, Guido Marchetto, *Senior Member, IEEE*, and Paolo Montuschi, *Fellow, IEEE*

***Abstract*—The recent growth of IoT devices, along with edge computing, has revealed many opportunities for novel applications. Among them, Unmanned Aerial Vehicles (UAVs), which are deployed for surveillance and environmental monitoring, are attracting increasing attention. In this context, typical solutions must deal with events that may change the state of the network, providing a service that continuously maintains a high level of performance. In this paper, we address this problem by proposing a distributed architecture that leverages a Multi-Agent Reinforcement Learning (MARL) technique to dynamically offload tasks from UAVs to the edge cloud. Nodes of the system co-operate to jointly minimize the overall latency perceived by the user and the energy usage on UAVs by continuously learning from the environment the best action, which entails the decision of offloading and, in this case, the best transmission technology, i.e., Wi-Fi or cellular. Results validate our distributed architecture and show the effectiveness of the approach in reaching the above targets.**

***Index Terms*—UAV, task offloading, multi-agent reinforcement learning.**

I. Introduction

UNMANNED aerial vehicle (UAV) systems have been experiencing a constantly increasing popularity during the last years, mainly thanks to their maneuverability, flexibility, and limited deployment costs. For example, nowadays, drone swarms can appear as a viable candidate for fast computation and communication if equipped with cameras, sensors, or civilian tablets and smartphones [1], [2]. Such a system is particularly suited for rapid disaster response and environmental monitoring, and systems to provide connectivity to ground stations. The role of drones, but in general of IoT devices, could become even more prominent in the near future as they enable, improve, and optimize novel and existing services [3]–[5]. Autonomous and semi-autonomous drones will surely continue to help humans in accomplishing many tasks, spanning from industrial inspection to survey operations, from rescue management systems to military or first responder support.

Manuscript received January 13, 2021; revised March 23, 2021; accepted April 12, 2021. Date of publication April 20, 2021; date of current version June 9, 2021. The work of Flavio Esposito was supported by NSF Awards CNS-1836906 and CNS-1908574. The review of this article was coordinated by Dr. Ai-Chun Pang. *(Corresponding author: Alessio Sacco.)*

Alessio Sacco, Guido Marchetto, and Paolo Montuschi are with the DAUIN, Politecnico di Torino, 10129 Turin, Italy (e-mail: alessio_sacco@polito.it; guido.marchetto@polito.it; paolo.montuschi@polito.it).

Flavio Esposito is with the Department of Computer Science, Saint Louis University, St. Louis, MO 63103 USA (e-mail: flavio.esposito@slu.edu).

Digital Object Identifier 10.1109/TVT.2021.3074304

In a drone fleet, the device computing power of the small mobile devices can be effectively enhanced if combined with the development of the multi-access edge computing (MEC) technology [6]–[8]. In such a scenario, the IoT device can offload computationally intensive tasks to nearby edge cloud to reduce computation latency and energy consumption [9]–[12]. For example, a network of drones can be used to collect a huge quantity of data in order to offload them at the edge for heavy audio/video processing.

Task offloading decision processes are generally modeled as mixed integer programming (MIP) problems, whose solution is often achieved by means of heuristics [13], [14], convex relaxation [15], [16], Markov approximation [17]. These approaches, however, require a considerable number of iterations to reach a satisfying local optimum, which makes them not suitable for real-time offloading decisions when environment conditions have fast and significant changes.

To enable learning in an unknown environment, reinforcement learning (RL) has been shown as a promising solution, which can help overcome the prohibitive computational requirements. Recent RL-based online offloading decisions solutions have demonstrated improvements compared to traditional approaches, e.g., [18]–[21]. However, none of them take full advantage of a possible collaborative framework and decisions are taken independently by each agent of the system.

In this paper, we propose the use of multi-agent reinforcement learning (MARL) to jointly improve the energy efficiency (EE) and task completion time of edge computing enabled UAVs swarms, while considering distributed offloading decision strategies. The proposed MARL algorithm can solve the computation offloading optimization problem in real-time by combining information coming from other devices, i.e., in a collaborative way, in order to decide if computing a task locally or offloading it to the closest edge cloud. In the case of offloading, the second decision entails the radio access technology (RAT) to consume, i.e., Wi-Fi or cellular, to transmit the task from the device to the edge cloud.

The presented decentralized algorithm leverages the actor-critic framework and is applicable to large-scale problems where both the number of states and the number of agents are massively large. Specifically, the actor step is performed individually by each agent with no need to communicate and infer the policies of other agents. On the other hand, for the critic step, each agent shares its estimate of the value function with its neighbors in order to achieve a consensual estimate, further used in the

subsequent actor step. In this regard, the local information at each agent is able to diffuse across the network, making the network-wide maximum reward achievable. As in standard distributed algorithms over networked systems, our algorithm provides the advantages of scalability to a large number of agents, robustness against malicious attacks, and communication efficiency.

The rest of the paper is structured as follows. Section II revises the related work on RL techniques and task offloading, while in Section III we describe some applications where our algorithm can be used. Despite being mainly inspired by the disaster response use case, our approach has indeed broader applicability. We present the system model in Section IV-B, along with the basic concepts about RL and MARL. Then, Section V describes the algorithm at the basis of our solution, which organizes resources and underpins the task offloading decision. Finally, in Section VI we present our experimental results, while Section VII concludes the paper.

II. Related Work

The problem of shortening task completion time by exploiting the close edge cloud is crucial for any type of IoT network in general, and robotic or drone networks in particular; so it is not surprising that there are several proposed solutions to tackle this problem. In the following, we first analyze the class for the RL model exploited by our solution and the differences between our implementation with previous approaches. Secondly, we cite a few representative (centralized and distributed) solutions to clarify our contributions to the decision task offloading problem.

Actor-Critic and Multi-Agent. The presented algorithms belong to the class of actor-critic framework, a special class of reinforcement learning (RL) problems. Actor-critic algorithms, which are based on the more general policy gradient theorem [22], have been widely studied in the literature since their birth [23], [24], also due to the proved convergence of algorithm with linear function approximation [25]. Recently, for deep reinforcement learning, where deep neural networks are used to approximate functions in RL settings, various actor-critic algorithms have been proposed. A variety of environments has entailed efficient actor-critic algorithms based on experience replay [26], off-policy learning [27], deterministic policies for continuous action spaces [28], [29], and the asynchronous actor-critic (A3C) algorithm [30], which became extremely popular. However, the vast majority of these approaches considers the single-agent setting of this algorithm, where the A3C deals with single-agent RL but with multiple parallel workers, and a central controller is required to coordinate the asynchronous update of the workers. In our system, in contrast, as the algorithm is based upon a policy gradient theorem for MARL, no central controller is necessary.

A more relevant and more recent trend is on MARL, which applies to the setting with both collaborative and competitive relationships among multiple agents. Many early algorithms [31]–[33], have been developed only for tabular cases where no function approximation is applied. When deep neural networks are used as function approximators, several MARL algorithms have gained increasing attention, e.g., [34]–[39]. Nonetheless, while some of them lack convergence guarantees, none of them has been designed to tackle the complexity and peculiarity of UAVs swarms, where task offloading optimizations are essential.

Task Offloading. In the last years, edge computing has been proved to be an effective method in supporting some latency-critical tasks [17], [40]. This paradigm can be particularly beneficial for UAV swarms, or in general unmanned aerial systems (UAS), e.g., self-driving vehicles, to conduct a computation offloading scheme with edge computing. Edge computing-based UAV swarms [41], are able to improve the latency and energy-efficiency issues caused by cloud computing [19]. In general, using ML/AI to optimize offloading process in vehicular environments has gained the attention in recent studies [42]–[44].

The minimization of transmission energy for single-user MEC systems, for instance, has been addressed under specific latency constraints in [45], [46]. Furthermore, in [11] the authors presented a game-theoretic approach to distributed offload computation among mobile device users, modeling the problem as a multi-user offloading game. You *et al.* [10] conceived a solution that determines the offloading data volume, the offloading duration, and the transmission resources of each user in an energy-efficient manner. Kalatzis *et al.* [47] decreased energy consumption in UAV based forest fire detection applications by adopting the edge and fog computing principles. However, such approaches fail in addressing the dynamicity of the environment, which is one of the main features of disaster scenarios, and hinders from high long-term performance.

Some researches studied the online computation offloading problem when edge computing resources are available. For instance, a task offloading solution built upon rent/buy problem aiming to minimize the task completion time in mobile clouds has been presented in [48]. At the same time, another recent trend is the utilization of RL in these circumstances, given its ability to adapt to highly dynamic environments [49], [50]. Huang *et al.* [18] proposed a deep reinforcement learning-based online offloading framework (DROO) to decide whether to offload tasks to the edge cloud and proportionally allocate wireless resources. Despite the similarity in the RL framework, our work differs from this class of solutions for the distributed nature that leads to multiple heterogeneous agents with potentially distinct policies and rewards, and the further improvements on protocol decisions. Besides, although distributed approaches in task offloading decisions leveraging deep RL exist, e.g., DDLO [51] and a hotbooting Q-learning based schema [52], these solutions use multiple parallel deep neural networks, rather than collaboratively take offloading decisions.

III. Motivating Applications

UAVs are often used for collecting data and sending them to the edge/fog, e.g., for data-intensive visual computing.

At network edges, indeed, there may be present more resources that can speed-up the processing. In particular, data-intensive visual computing requires seamless processing of imagery/video at the network-edge and resilient performance to guarantee adequate user Quality of Experience (QoE) expectations. This is particularly critical, e.g., in (natural or human-made) disaster scenarios, due to the poor bandwidth availability and the highly variable conditions. These applications should be able to provide rapid awareness through videos or audios collected at salient incident scenes in order to plan a proper response that can minimize disaster impact and/or save lives [53].

To meet such network-edge data-intensive computations and local storage requirements, *edge computing* is a valuable solution [54], by providing on-demand network, storage, and computational resources that compensate (scaling up and scaling down on demand) the insufficient local processing capabilities within a geographical area of interest. Edge computing extends the notion of cloud, but it is placed closer to the location of users and data sensors, reducing latency and enabling real-time decision making. A few examples of how edge computing could be of help in the above described scenarios are reported in the following.

Reconnaissance to save lives. A (very) large fleet of camera-equipped UAVs collect visible (or infrared) imagery, e.g., to recognize body temperatures or identify bodies under ruins or massive avalanches. In such environments, image processing is key, to first enhance the image, e.g., dehaze, stabilize, compress inputs for lower level image processing, and then apply computationally intensive computer vision algorithms. Transmitting such data to a remote cloud is thus unfeasible, given the poor connection bandwidth which dramatically increases the data transfer time.

Reuniting lost citizens and families. Online face recognition software runs at the edge and acts on imagery snapped from cameras onboard the UAVs. Face image feature extraction processing performed at the network edge would attempt to match against a database of missing people without encountering poor network or processing performance. The face detection and identification can gain great benefit from utilizing deep neural networks-based models, that are rapidly improving their performance in this field.

Property Surveillance. Alarms or other actuators may be triggered if the continuous monitoring performed by UAVs detects activities of concern, such as a fire, a human intrusion, or a broken window. A first video analytic pre-scanning phase is recommended to run at the edge, and only upon the completion data could be sent to the cloud core, where a more in-depth analysis can occur and the video can be shared with law enforcement for further investigation.

IV. Model and Problem Definition

In this section, we first present some preliminary notions on actor-critic and multi-agent reinforcement learning (Section IV-A), used in our UAV task offloading model (Section IV-B and Section IV-C) and problem definition (Section IV-D).

A. Background on Actor-Critic and Multi-Agent Reinforcement Learning

Before describing the details and the notation of our model, we first describe the actor-critic framework and the MARL concepts, where our system is built upon.

Actor-Critic Algorithm. The Actor-Critic belongs to the class of model-free, online, on-policy reinforcement learning methods. The goal of an agent is to optimize the policy (actor) directly and train a critic to estimate the return or future rewards. Hence, at the very basis, a Markov decision process exists and is characterized by a quadruple $\mathcal{C} = \langle S, A, P, R \rangle$, where S denotes the finite state space, A is the finite action space, $P(s'|s,a) : S \times A \times S \to [0,1]$ refers to the state transition probability from state s to state s' determined by action a, and $R(s,a) : S \times A \to \mathbb{R}$ is the reward function defined by $R(s,a) = \mathbb{E}[r_{t+1}|s_t = s, a_t = a]$, where r_{t+1} is the instantaneous reward at time t. The probability of choosing action a at state s is the policy of the agent, defined as the mapping $\pi : S \times A \to [0,1]$. The agent has the objective of finding the optimal policy that maximizes the expected time-average reward, i.e., the long-term return, which is given by $J(\pi)$:

$$J(\pi) = \lim_{T} \frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}[r_{t+1}] = \sum_{s \in S} d_\pi(s) \sum_{a \in A} \pi(s,a) R(s,a), \tag{1}$$

where $d_\pi(s) = \lim_{t \to \infty} \mathbb{P}(s_t = s|\pi)$ represents the stationary distribution of the Markov chain under policy π. Such a distribution $d_\pi(s)$ and the limit in (1) are well defined when the Markov chain resulting from the Markov Decision Process (MDP) is irreducible and aperiodic with any policy π.

Given any policy π, the action-value associated with the state s and action a, $Q_\pi(s,a)$, is thus defined, according to [55], as:

$$Q_\pi(s,a) = \sum_{t} \mathbb{E}[r_{t+1} - J(\pi)|s_0 = s, a_0 = a, \pi]. \tag{2}$$

Furthermore, the state-value associated with state s under policy π can be defined as $V_\pi(s) = \sum_{a \in A} \pi(s,a) Q_\pi(s,a)$. In the following, we simply refer to $Q_\pi(s,a)$ and $V_\pi(s)$ as *action-value* and *state-value* functions respectively. When the action or state spaces are massively large, these two functions are usually approximated by some parameterized functions $Q(\cdot,\cdot;\omega)$ and $V(\cdot;\nu)$, depending on the parameters ω and ν. Also the policy π can be parameterized by parameter θ in π_θ. For the sake of simplicity, hereafter we replace the subscript π_θ with just θ, e.g., V_{π_θ} to V_θ.

Actor-critic (AC) algorithms have been advocated to solve, with this parameterization, the optimal policy π_θ. Built on the well-known policy gradient theorem [23], AC algorithms are characterized by the gradient of the return $J(\theta)$ written as:

$$\nabla_\theta J(\theta) = \mathbb{E}_{s \sim d_\theta, a \sim \pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) \cdot (Q_\theta(s,a) - b(s))], \tag{3}$$

where the term $b(s)$ is commonly named baseline, and $\nabla_\theta \log \pi_\theta(s,a)$ is referred as the score function for policy π_θ. Also, let the advantage function be:

$$A_\theta(s,a) = Q_\theta(s,a) - V_\theta, \tag{4}$$

which specifies how much better it is to take a specific action compared to the average, general action at the given state. Indeed, it has been recognized, e.g., in [25], that the minimum variance baseline in the action-value function estimator is the state-value function $V_\theta(s)$. Defining $Q_t(\omega) = Q(s_t, a_t; \omega)$ at time t, and let A_t the sample at time t of the advantage function, we get:

$$A_t = Q(s_t, a_t; \omega_t) - \sum_{a \in A} \pi_{\theta_t}(s_t, a) Q(s_t, a; \omega_t), \tag{5}$$

Let then $\psi_t = \nabla_\theta \log \pi_{\theta_t}(s_t, a_t)$ be the sample of the score function. The AC algorithm based on the action-value function approximation is based on the following updates:

$$\begin{aligned} \mu_{t+1} &= (1 - \xi_{\omega,t}) \cdot \mu_t + \xi_{\omega,t} \cdot r_{t+1}, \\ \omega_{t+1} &= \omega_t + \xi_{\omega,t} \cdot \delta_t \cdot \nabla_\omega Q_t(\omega_t), \\ \theta_{t+1} &= \theta_t + \xi_{\theta,t} \cdot A_t \cdot \psi_t, \end{aligned} \tag{6}$$

where $\xi_{\omega,t}, \xi_{\theta,t} > 0$ are the stepsizes, μ_t tracks the unbiased estimate of the average return, and δ_t refers to the action-value temporal difference (TD) error and is defined as:

$$\delta_t = r_{t+1} - \mu_t + Q(s_{t+1}, a_{t+1}; \omega_t), \tag{7}$$

where action a_{t+1} is retrieved from the policy $\pi_{\theta_t}(s_{t+1}, \cdot)$. This TD error is used to evaluate the action just selected, i.e., the action a_t taken in state s_t. A positive TD error suggests that the tendency to select this action should be strengthened for the future, whereas a negative TD error suggests the tendency should be weakened.

The standard AC algorithm is defined as a two-time-scale algorithm, where the two stepsizes are set such that $\lim_{t\to\infty} \xi_{\omega,t} \cdot \xi_{\theta,t}^{-1} > 0$. The first two updates in (6) belongs to the *critic step*, which operates at a faster time scale; while the last update in (6) corresponds to the *actor step* that occurs at a slower time scale. The actor controls how our agent behaves by improving the policy along the gradient ascent direction; on the other hand, the critic measures how good is the action taken, by estimating the action-value function under policy π_{θ_t}.

Finally, actor-critic algorithms are able to achieve state-of-the-art performance in many complicated application domains, as shown in [24], [30], [56]. Inspired by these achievements, we further define a MARL algorithm based on the AC approach.

Multi-Agent Reinforcement Learning. We consider now a system of N agents operating in a common environment with no central controller that either collects rewards or makes the decisions for the agents. In this context, the set of agents is denoted by $\mathcal{N}_t$, whose cardinality is N, and each agent can communicate with each other. In general, the set of agents is a time-varying set, defined as $\mathcal{N}_t$ at time $t \in \mathbb{N}$.

A time-varying multi-agent MDP is defined as a tuple $(S, \{A^i\}_{i\in\mathcal{N}}, P, \{R^i\}_{i\in\mathcal{N}}, \{\mathcal{N}_t\}_{t\geq 0})$, where S denotes the global state space shared by all the agents in $\mathcal{N}_t$, and A^i is the action set that agent i can execute. Besides, let $A = \prod_{i=1}^N A^i$ be the joint action space of all agents, also referred to as global action profile. We then define $R^i : S \times A \to \mathbb{R}$ the local reward function of agent i, while $P : S \times A \times S \to [0, 1]$ is the state transition probability. In this system, we assume that the states and the joint actions are globally observable, while the rewards are observed only locally.

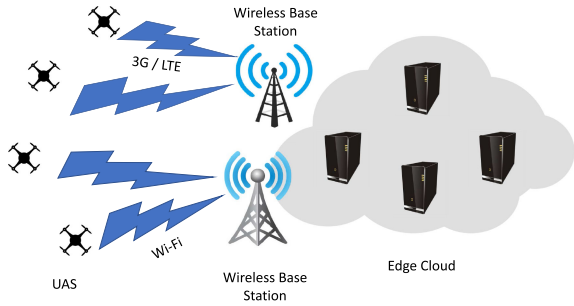


Fig. 1. System Overview: mobile devices, e.g., UAVs, interaction with the edge cloud via cellular and Wi-Fi network.

At time step t, assuming the global state space is $s_t \in S$ and the joint actions of agents are $a_t = (a_t^1, \ldots, a_t^N) \in A$, each agent will receive a reward r_{t+1}^i, which is a random value with $R^i_{(s_t,a_t)}$ as expected value. Also, the model shift to the new state $s_{t+1} \in S$ with probability $P(s_{t+1}|s_t, a_t)$. Our model is considered as fully decentralized since the reward is locally received and the action is performed locally by each agent.

As the state space S may be large, it is convenient to consider policies that are in a parametric function class, similar to the single AC. For agent i the local policy is then given by $\pi^i_{\theta^i}$, where $\theta^i \in \Theta^i$ is the parameter, and $\Theta^i \subseteq \mathbb{R}^{R_i}$ is a compact set. We then pack these parameters altogether in $\theta = [(\theta^1)^T, \ldots, (\theta^N)^T] \in \Theta$, where $\Theta = \prod_{i=1}^N \Theta^i$. Therefore, the joint policy is given by $\pi_\theta(s, a) = \prod_{i=1}^N \pi^i_{\theta^i}(s, a_i)$, and is often shortened as π_θ.

Joint objective of the agents is to collaboratively find the joint policy π_θ that maximizes the globally averaged long-term return based solely on local information. The optimization problem to solve is:

$$\begin{aligned} \max_\theta J(\theta) = \lim_T \frac{1}{T} \mathbb{E}\left[\sum_{t=0}^{T-1} \frac{1}{N} \sum_{i\in\mathcal{N}} r_{t+1}^i\right] = \\ \sum_{s\in S} d_\theta(s) \sum_{a\in A} \pi_\theta(s, a) \cdot \overline{R}(s, a), \end{aligned} \tag{8}$$

where $\overline{R}(s, a) = N^{-1} \cdot \sum R(s, a)$ is the globally averaged reward function. Further, given $\overline{r}_t = N^{-1} \cdot \sum_{i\in\mathcal{N}} r_t^i$, it yields $\overline{R}(s, a) = \mathbb{E}[\overline{r}_{t+1}|s_t = s, a_t = a]$. Hence, the global expected action value function for a state-action pair (s, a) under policy π_θ is:

$$Q_\theta(s, a) = \sum_t \mathbb{E}\left[\overline{r}_{t+1} - J(\theta)|s_0 = s, a_0 = a, \pi_\theta\right], \tag{9}$$

Finally, the global state-value function $V_\theta(s)$ is given by $V_\theta(s) = \sum_{a\in A} \pi_\theta(s, a) Q_\theta(s, a)$.

B. System Model

As shown in Fig. 1, we consider a UAV swarm consisting of a set of agents $\mathcal{N}_t = \{A_1, \ldots, A_N\}$, each of which has a task to be completed. We consider that the set $\mathcal{N}_t$ can change over time

since the agents may suffer failures or running out of power. However, for simplicity, we often refer to this set as $\mathcal{N}$ in the following, without any ambiguity.

The overall system is compound of M tasks, denoted by a set of tasks $\mathcal{M} = \{T_1, \ldots, T_M\}$. The mobile node can either compute the task locally or offload the computation to the edge cloud in two ways, i.e., through a mobile network (LTE) or through Wi-Fi access points. In this paper, we consider an application where tasks are independent.

1) Communication Model: As mentioned earlier, the access point for wireless communication can be either a Wi-Fi access point, or a base-station in cellular networks. The channel from mobile node i to access point s follows quasi-static block fading.

Let $o^1_{i,m}$ denote the computation offloading decision of task m of mobile device n. Specifically, $o^1_{i,m} = 1$ means that the node offloads the task via the wireless channel, while $o^1_{i,m} = 0$ means that the node performs the task locally on its own device. When task is set to be performed at the edge cloud, the communication can occur over cellular (e.g., LTE) network if $o^2_{i,m} = 1$ or Wi-Fi network for $o^2_{i,m} = 0$. Given the global action profile A for any node i and task m, we can compute the uplink data rate for computation offloading over cellular technology of task m of mobile device i as:

$$Rt^c_{i,m}(A) = W^c \cdot \cdot \log_2 \left(1 + \frac{P^c_{i,m} H^c_{i,m}}{(\sigma^c_{i,m})^2 + \sum_{j \neq i, k \neq m, o^1_{j,k}=1, o^2_{j,k}=1} P^c_{j,k} H^c_{j,k}} \right), \tag{10}$$

where $P^c_{i,m}$ is the transmission power of node i offloading task m to the edge cloud via cellular connectivity; $H^c_{i,m}$ denotes the channel gain from node i to access point s when transmitting task m due to the path loss and shadowing attenuation; $(\sigma^c_{i,m})^2$ indicates the thermal noise power associated with the link between the node i and the access point s, and W^c is cellular channel bandwidth. From (10) we can observe that when many mobile devices offload their tasks via cellular access simultaneously, they may lead to severe interference and low data rates.

Likewise, we define the uplink rate of Wi-Fi network similar to the cellular transmission as follows:

$$Rt^w_{i,m}(A) = W^w \cdot \cdot \log_2 \left(1 + \frac{P^w_{i,m} H^w_{i,m}}{(\sigma^w_{i,m})^2 + \sum_{j \neq i, k \neq m, o^1_{j,k}=1, o^2_{j,k}=0} P^w_{j,k} H^w_{j,k}} \right), \tag{11}$$

where the involved variables have the same meaning of those in (10).

2) Computation Model: Let $D_{i,m}$ denote the size of computation data (e.g., the recorded audio in UAVs swarm) related to computation task m of node i. $L_{i,m}$ denotes the computing workload, i.e., the total number of CPU cycles needed to accomplish task m of node i. In the following, we consider the computation overhead in terms of energy consumption and application completion time for local and edge cloud computing. Further, we differentiate the edge offloading into two cases, that represent the two possible communication options: cellular and Wi-Fi networks.

Local Computing Mode. We denote the computation capability, i.e., the clock frequency of the CPU chip, of node i, on task m, as $f_{i,m}$. Our model allow different mobile devices to have different computation capability with different clock frequency per task. The local execution time of task m on node i is hence given by:

$$T^{l,exec}_{i,m} = \frac{L_{i,m}}{f_{i,m}}, \tag{12}$$

while the energy consumption of the device is given by:

$$E^l_{i,m} = k L_{i,m} f^2_{i,m}, \tag{13}$$

where k denotes the effective switched capacitance for the specific chip architecture. In line with previous studies, e.g., [15], [57], we set $k = 10^{-11}$. Clearly, the clock frequency of the CPU chip can be adjusted by using the DVFS technique to achieve the optimum computation time and energy consumption on a device.

Aside the execution time, the time to complete task m is also affected by the *waiting time* $T^{wt}_{i,m}$. The waiting time of a task is defined as the time that task m spends on board of i before its execution.

Consequently, the completion time for a local execution of task m on node i is the sum of the local computation execution time and the waiting time in local computing,

$$T^l_{i,m} = T^{l,exec}_{i,m} + T^{wt}_{i,m}. \tag{14}$$

We are now ready to introduce the computational cost of a task, which dictates our energy-efficient strategy.

Definition IV.1: *Computational Cost. The computational cost is defined as the weighted sum of energy consumption and completion time related to the execution of a task m belonging to node i.*

In the case of local execution, it is given by:

$$Z^l_{i,m} = \alpha^l_{i,m} T^l_{i,m} + \beta^l_{i,m} E^l_{i,m}, \tag{15}$$

where $\alpha^l_{i,m}$ and $\beta^l_{i,m}$ are the weights for the energy consumption and the computation completion time respectively.

This form of computational cost enables to meet different user demands by adjusting the weights, and, for example, save more energy rather than shortening the delay. For delay-sensitive applications, such as rapid disaster response set-up, a larger $\beta^l_{i,m}$ is recommended to meet the strict user requirements. In this regard, the weights control the importance of the perceived latency and energy consumption respectively.

Edge Computing Mode. In case the mobile node i offloads the computation task m to the edge cloud, the latter executes the computation task and returns the results to the device. When the task is offloaded to the edge cloud, the execution entrails three

phases: *(i)* the transmission phase, *(ii)* the edge computation phase, *(iii)* the outcome receiving phase.

Starting with the first phase, we consider the time and energy consumed during transmission. In line with the computation and communication model, we can define the transmission time and energy consumption for task offloading over cellular network as:

$$T_{i,m}^{c,tra}(A) = \frac{D_{i,m}}{Rt_{i,m}^{c}(A)}, \tag{16}$$

$$E_{i,m}^{c,tra}(A) = P_{i,m}^{c} T_{i,m}^{c,tra}(A), \tag{17}$$

respectively. The transmission over Wi-Fi technology entails different transmission time and energy consumption, as follows:

$$T_{i,m}^{w,tra}(A) = \frac{D_{i,m}}{Rt_{i,m}^{w}(A)}, \tag{18}$$

$$E_{i,m}^{w,tra}(A) = P_{i,m}^{w} T_{i,m}^{w,tra}(A). \tag{19}$$

Besides, for edge cloud execution we can derive the computation execution time for task m of node i as:

$$T_{i,m}^{e,exec} = \frac{L_{i,m}}{f_e}, \tag{20}$$

where f_e refers to the clock frequency of the edge cloud. In this case, the assumption is that the frequency does not change during the computation and is constant over time. Moreover, we assume the energy consumption in the edge cloud is negligible since the cloud is in general powered by alternating current and has enough energy to execute the offloaded tasks. Although the offloaded task needs to wait before it is assigned to the proper resource in the cloud for the execution, we omit this waiting time for simplicity, as it is negligible with respect to the other quantities involved. Finally, as it is done in several other studies, e.g., [58], [59], we ignore the time for receiving the outcome of task m, since the received data is typically small. As such, the completion time for the edge offloading is the sum of the execution time and the transmission time over the wireless channel. For the cellular case we have:

$$T_{i,m}^{c} = T_{i,m}^{c,tra}(A) + T_{i,m}^{e,exec}. \tag{21}$$

On the other hand, if the offloading is performed over the Wi-Fi network, the completion time is computed as:

$$T_{i,m}^{w} = T_{i,m}^{w,tra}(A) + T_{i,m}^{e,exec}. \tag{22}$$

Consequently, the *computational cost* of task m of node i on the edge cloud through the cellular network is:

$$Z_{i,m}^{c} = \alpha_{i,m}^{c} T_{i,m}^{c} + \beta_{i,m}^{c} E_{i,m}^{c,tra}(A), \tag{23}$$

where a small data transmission rate $Rt_{i,m}(A)^c$ of the device i would result in high energy consumption in the wireless communication and long transmission time for offloading data to the closest edge cloud.

Similarly, we define the *computational cost* for the Wi-Fi offloading as:

$$Z_{i,m}^{w} = \alpha_{i,m}^{w} T_{i,m}^{w} + \beta_{i,m}^{w} E_{i,m}^{w,tra}(A), \tag{24}$$

where the weights may differ from the ones utilized in (23).

TABLE I
THE CONTEXTUAL METRICS GATHERED FOR BUILDING THE STATE SPACE

	Features	Description
1	d_i	Distance between agent i and base station [m]
2	q_i^c	Cellular Reference Signal Received Quality (RSRQ) [dB]
3	i_i^w	Wi-Fi Received Signal Strength Indicator (RSSI) [dB]
4	t_i^c	Cellular throughput [kbps]
5	t_i^w	Wi-Fi throughput [kbps]
6	r_i^c	Cellular RTT [ms]
7	$r_i^w i$	Wi-Fi RTT [ms]
8	l_i^c	Cellular lossrate [%]
9	$l_i^w i$	Wi-Fi lossrate [%]

Notably, to enable diversity among the three cases in the importance of latency with respect to the energy, the computational costs have different weights, depending on where the task is performed. That is, $\alpha_{i,m}^{w}$ is not necessarily equal to $\alpha_{i,m}^{c}$ and $\alpha_{i,m}^{l}$. Likewise for $\beta_{i,m}^{w}$, $\beta_{i,m}^{c}$ and $\beta_{i,m}^{l}$.

C. MARL Framework Formulation

Following the standard notation for reinforcement learning algorithms, we define the *state space* as the set of metrics used to select the best action among all the actions defined in the *action space*. The action selection occurs with the aim of maximizing a *reward function*, which represents the objective (utility) to optimize.

State Space. We report in Table I the features adopted to build our model state space. For each agent i in the network, we save the shown metrics for cellular and Wi-Fi communications. The first information esteems the distance between the agent and the base station, and is the same for both Wi-Fi and cellular transmissions. The subsequent features consider the quality of the signal, the throughput, the round-trip-time (RTT), and the loss rate, for the cellular and Wi-Fi channels separately. These quantities change over time as effect of the single and combined actions of the system, so we define the state space at time t as s_t.

The choice of such features is dictated by a design goal of balancing the overhead introduced by the metrics collection and the precision in grasping the system conditions. Empirically, we found that this state set produces the optimal trade-off, as also outlined by the goodness of our results (Section VI). It can be noted, indeed, as part of these quantities are already captured by the TCP protocol and form its state. Thus, our solution can easily leverage these quantities, reducing the overhead.

Action Space. The main decision that the agent is supposed to take, is whether or not to offload the task to the edge cloud. Formally, the first decision for each agent i is the binary offloading decision $o_{i,m}^{1}$:

$$o_{i,m}^{1} = \begin{cases} 1, & \text{if tasks are to be offloaded} \\ 0, & \text{otherwise.} \end{cases}$$

If $o_{i,m}^{1} = 0$, task is computed locally, whereas for $o_{i,m}^{1} = 1$ the incoming task is offloaded to the closest station. In the latter case, the subsequent decision regards the technology on which the transmission occurs. In fact, as the offloading occurs, the protocol and technology for transmitting bytes are extremely

relevant for shortening the latency. With this respect, we define a second binary decision $o^2_{i,m}$:

$$o^2_{i,m} = \begin{cases} 1, & \text{if cellular technology is preferred} \\ 0, & \text{if Wi-Fi technology is preferred.} \end{cases}$$

Such a decision takes place only for an $o^1_{i,m} = 1$, and we can observe how the total number of actions for each agent i is three, for an action set as follows: $A_i = [a^1_i, a^2_i, a^3_i]$, where a^1_i denotes $o^1_{i,m} = 0$, a^2_i is $o^1_{i,m} = 1, o^2_{i,m} = 0$, and a^3_i is $o^1_{i,m} = 1, o^2_{i,m} = 1$.

Utility function (RL reward). Based on reinforcement learning, the agent selects the action with the highest global reward. This choice relies upon the utility function, that specifies the objective of our algorithm. While RL can take a variety of different objectives, we define a function as follows to minimize the total latency and the usage of resources:

$$\begin{aligned} &U_{i,m} \\ &= -o^1_{i,m}o^2_{i,m}Z^c_{i,m} - o^1_{i,m}(1-o^2_{i,m})Z^w_{i,m} - (1-o^1_{i,m})Z^l_{i,m}, \end{aligned} \tag{25}$$

where a high cost in terms of computational time and energy consumption leads to small utility value. Further, we can easily define the utility per agent for all tasks as follows

$$U_i = \sum_{m=1}^{M} U_{i,m}. \tag{26}$$

The utility function is the real objective that each agent attempts to optimize; still, its value cannot be used to specify the desirability of the action taken in a particular state and hence cannot be directly used as a *reward* for the learning process [60]. The ambiguity in action evaluation comes from the unique dynamic network environment the learning agent is interacting with, and it means we cannot merely take the utility value to define the reward. For this reason, we consider the difference between consecutive utility values as the reward. This is because an increase in the utility value denotes an improvement and hence, the corresponding action should be encouraged, regardless of the original value of the utility. Consequently, we define the reward value as follows:

$$r^i_t = \begin{cases} a & \text{if } U^i_t - U^i_{t-1} > \epsilon \\ b & \text{if } U^i_t - U^i_{t-1} < -\epsilon \\ 0 & \text{otherwise,} \end{cases} \tag{27}$$

where U^i_t refers to the cumulative utility at time t, a is a positive value, and b is a negative value. Both indicate the reward (a reinforcement signal) given the direction of changes between two newly observed consecutive utility values, while ϵ is a tunable parameter that sets the sensitivity of the learning agent to changes in the utility values (i.e., it sets a tolerance in the value change).

It is worth noticing that each agent can potentially utilize a different reward, and the system can be easily extended towards this scenario. However, for the sake of simplicity, in the following we assume that all agents share the same utility.

D. Problem Formulation

Given the system model, we can formulate the optimization problem that our MARL algorithm aims to solve. First, let the computational cost of a sequence of tasks $\mathcal{M}$ for the mobile node i be:

$$\begin{aligned} Z_i = \sum_{m=1}^{M} Z_{i,m} = \sum_{m=1}^{M} \Big(& o^1_{i,m}o^2_{i,m}Z^c_{i,m} + \\ & + o^1_{i,m}(1-o^2_{i,m})Z^w_{i,m} + (1-o^1_{i,m})Z^l_{i,m} \Big), \end{aligned} \tag{28}$$

where M is the size of the set $\mathcal{M}$.

Formally, we have the following optimization problem:

$$\min_{A} \sum_{i} Z_i \tag{29}$$

$$\begin{aligned} \text{s.t.} \quad & o^1_{i,m}o^2_{i,m}T^c_{i,m} + o^1_{i,m}(1-o^2_{i,m})T^w_{i,m} + \\ & o^1_{i,m}(1-o^2_{i,m})T^w_{i,m} \leq T^{max}_m \quad \forall m = 1, \ldots, M \end{aligned} \tag{30}$$

where $A = \{o^1_{i,m}, o^2_{i,m} | i \in \mathcal{N}, m \in \mathcal{M}\}$. The constraint stated by (30) imposes that the total completion time of all the tasks is bounded by the required maximum completion time, T^{max}_m. This time deadline is application-specific, and can vary based on user needs.

The key challenge in solving the optimization problem is that the integer constraint of the device actions, i.e., $o^1_{i,m}, o^2_{i,m}$, makes the problem a mixed integer programming problem, which is generally non-convex and NP-hard. Thus, solving the problem by using a multi-agent reinforcement learning approach reduces complexity and allows reaching a feasible solution in polynomial time.

V. Our Algorithm

Based on the previous formulations, we design an algorithm to establish the offloading decision. An Actor-Critic (AC) algorithm comes with multiple flavours, e.g., Q Actor-Critic, Advantage Actor-Critic, TD-error Actor-Critic. Among them, we follow the TD-error variant for the computation of the Critic.

In the following we first show the formulation of the policy gradient in a multi-agent setting. Then, we present the proposed MARL algorithm for our decentralized multi-agent system.

A. MARL System Optimization

We recall that $\pi_\theta : S \times A \to [0, 1]$ is the derived joint policy for the packed weights of the neural networks $\theta \in \Theta$, the globally long-term averaged return is $J(\theta)$, and Q_θ and A_θ are the action-value function and advantage function, respectively. Then, for any $i \in \mathcal{N}$, we define the local advantage function $A^i_\theta : S \times A \to \mathbb{R}$ as:

$$A^i_\theta(s, a) = Q_\theta(s, a) - \widetilde{V^i_\theta}(s, a^{-i}), \tag{31}$$

where a^{-i} denotes actions of all agents except for agent i, and $\widetilde{V^i_\theta}(s, a^{-i}) = \sum_{a^i \in A^i} \pi^i_{\theta^i}(s, a^i) \cdot Q_\theta(s, a^i, a^-i)$. Given the

outcome of the Policy Gradient Theorem for MARL systems [39], we can compute the gradient of $J(\theta)$ as follows:

$$\nabla_{\theta^i} J(\theta) = \mathbb{E}_{s\sim d_\theta, a\sim\pi_\theta}[\nabla_{\theta^i} \log \pi^i_{\theta^i}(s,a^i)\cdot A^i_\theta(s,a)] \quad (32)$$

This gradient is applied to $J(\theta)$, previously defined in (8).

This result is precious as it shows that the policy gradient with respect to each θ^i can also be computed locally using the corresponding score function $\nabla_{\theta^i}\log\pi^i_{\theta^i}(s,a^i)$. However, local information is insufficient to estimate the global action-value and the advantage functions. These functions are necessary to compute the gradient and they require the reward values $\{r^i_t\}_{i\in\mathcal{N}}$ of all agents. For this reason, our proposed algorithm fosters collaboration among the agents and includes a consensus-based phase to diffuse the local information among them.

B. Local Updates and Consensus-Based Phase

The AC algorithm consists of two steps that occur at different time scales. In the critic step, the update is similar to the action-value TD-learning in (6), followed by a linear combination of its neighbor's parameter estimates. This parameter sharing step is also known as the consensus update, and involves a weight matrix $C_t = [c_t(i,j)]_{N\times N}$, where $c_t(i,j)$ denotes the weight on the message transmitted from i to j at time t. In the following process, each agent only uses the transition at time t, i.e., sample (s_t, a_t, s_{t+1}) for updating the parameters. First, we estimate $J(\theta)$ and V_θ with, respectively, a scalar μ and a parameterized function $V(\cdot,v): S\to\mathbb{R}$, where parameter $v\in\mathbb{R}^L$ with $L\ll|S|$. Each agent i shares local parameters μ^i and v^i, and updates its information as follows:

$$\begin{aligned}
\widetilde{\mu}^i_t &= (1-\xi_{v,t})\cdot\mu^i_t + \xi_{v,t}\cdot r^i_{t+1},\\
\mu^i_{t+1} &= \sum_{j\in\mathbb{N}} c_t(i,j)\cdot\widetilde{\mu}^j_t,\\
\delta^i_t &= r^i_{t+1} - \mu^i_t + V_{t+1}(v^i_t) - V_t(v^i_t),\\
\widetilde{v}^i_t &= v^i_t + \xi_{v,t}\cdot d^i_t\cdot\nabla_v V_t(v^i_t),\\
v^i_{t+1} &= \sum_{j\in\mathbb{N}} c_t(i,j)\cdot\widetilde{v}^j_t,
\end{aligned} \quad (33)$$

where, for the sake of simplicity, $V_t(v) = V(s_t;v)\forall v\in\mathbb{R}^L$, and $\xi_{v,t}>0$ is the stepsize. In this context, differently from the single AC case, δ^i_t denotes the state-value TD-error of agent i.

Given the globally averaged reward $\overline{R}(s,a) = N^{-1}\cdot\sum R(s,a)$, the agent estimates the value $\overline{R}(s,a)$ in the critic step. Formally, let $\overline{R}(\cdot,\cdot;\lambda): S\times A\to\mathbb{R}$ be the class of parameterized functions and $\lambda\in\mathbb{R}^M$ be the parameter with $M\ll|S|\cdot|A|$. Motivated by the distributed optimization literature [61], [62], in order to obtain the estimate of $\overline{R}(\cdot,\cdot;\lambda)$, we minimize the following weighted mean-square error at the faster time scale:

$$\min_{\lambda}\sum_{i\in\mathcal{N}}\sum_{s\in S,a\in A}\delta_\theta(s)\cdot\pi_\theta(s,a)\cdot\left[\overline{R}(s,a;\lambda) - R^i(s,a)\right], \quad (34)$$

where δ_θ refers to the stationary distribution of the Markov chain $\{s_t\}_{t\geq 0}$ under policy π_θ. To solve this minimization problem, the updates to λ^i_t are as follows:

$$\begin{aligned}
\widetilde{\lambda}^i_t &= \lambda^i_t + \xi_{v,t}\cdot\left[r^i_{t+1} - \overline{R}_t(\lambda^i_t)\cdot\nabla_\lambda\overline{R}_t(\lambda^i_t)\right],\\
\lambda^i_{t+1} &= \sum_{j\in\mathbb{N}} c_t(i,j)\cdot\widetilde{\lambda}^j_t,
\end{aligned} \quad (35)$$

where $\overline{R}_t(\lambda)$ is a compact notation for $\overline{R}_t(s,a;\lambda)$. It is worth noticing that this procedure preserves the privacy of agents on their rewards and policies, since the rewards of other agents are not transmitted and the estimate $\overline{R}(\cdot,\cdot;\lambda)$ cannot be used to reconstruct original reward of other agents.

The updates in (35), (33) forms the critic step. On the other hand, the actor step uses the estimate $\overline{R}_t(\lambda^i)$ to evaluate the globally averaged TD-error $\widetilde{\delta}^i_t$ and performs the updates:

$$\begin{aligned}
\widetilde{\delta}^i_t &= \overline{R}_t(\lambda^i_t) - \mu^i_t + V_{t+1}(v^i_t) - V_t(v^i_t),\\
\theta^i_{t+1} &= \theta^i_t + \xi_{\theta,t}\cdot\widetilde{\delta}^i_t\cdot\psi^i_t,
\end{aligned} \quad (36)$$

where ψ^i_t is defined as $\psi^i_t = \nabla_{\theta^i}\log\pi^i_{\theta^i_t}(s_t,a^i_t)$ and $\xi_{\theta,t}>0$ is the stepsize.

We summarize the steps of the presented algorithm in Algorithm 1. After a first initialization phase, the agents start the individual actor and critic steps. These steps occur with a period of Δt in order to not overload the agent itself, where the optimal Δt is selected via a sensitivity analysis (Section VI). The elaborated values are then sent to the neighbors, and upon receiving such values, each agent updates its parameters to embrace a global view of the action performed.

The actor is a neural network working as a function approximator and its task is to produce the best action for a given state. The network shape is optimized empirically and motivated in the evaluation (Section VI). The critic is another function approximator, i.e., a neural network, which, receiving as input the environment and the action by the actor, outputs the action value (Q-value) for the given pair.

Given the values to be stored for critic and actor steps, online implementing this algorithm requires a memory complexity of $\mathcal{O}(N+L+M+R_i)$ for each agent i. This complexity results in a great benefit compared to the regular reinforcement learning algorithm, where a huge Q-table need to be stored in each agent for a large N.

VI. Evaluation Results

A. Experimental Setup

To evaluate the proposed solution, we run extensive experiments on an emulated cloud edge system scenario where several agents (the UAVs) can offload tasks to the edge by means of either cellular or Wi-Fi communications. Each agent is represented by a process running in the system, while the edge cloud is replicated by means of a further process emulating the execution of offloaded tasks. Channel parameters regarding the cellular and Wi-Fi connections are obtained from a real dataset publicly available [63]. The LTE technology is considered as a reference for the cellular case. To represent these channel conditions, we use the Mahimahi emulator [64], a recent network emulator that allows testing with real traces. First, we adapt the information

Algorithm 1: MARL Actor-Critic.

Initialize $\mu_0^i, \widetilde{\mu_0^i}, v_0^i, \widetilde{v_0^i}, \lambda_0^i, \widetilde{\lambda_0^i}, \theta_0^i, \forall i \in \mathcal{N}$
Initialize $s_0, \{\xi_{v,t}\}_{t\geq 0}, \{\xi_{\theta,t}\}_{t\geq 0}$
Each agent i implements $a_0^i \sim \pi_{\theta_0^i}(s_0;\cdot)$
Step counter $t \leftarrow 0$
for all $i \in \mathcal{N}$ **do**
  if queued tasks **then**
    for All tasks **do**
      Take action $a_t^i \sim \pi_{\theta_0^i}(s;\cdot)$
    if $a_t^i = a_i^1$ or $a_t^i = a_i^2$ **then**
      Offload task m to the edge
    else
      Compute task m locally
for every interval Δt **do**
  Collect metrics that form state s_t
  Update $\widetilde{\mu_t^i}, \delta_t^i$ according to (33)
  Update $\widetilde{\lambda_t^i}$ according to (35)
  Critic Step: $\widetilde{v_t^i} \leftarrow v_t^i + \xi_{v,t} \cdot d_t^i \cdot \nabla_v V_t(v_t^i)$
  Update $\widetilde{\delta_t^i}$ according to (36)
  Update $\psi_t^i \leftarrow \nabla_{\theta^i} \log \pi_{\theta_t^i}^i(s_t, a_t^i)$
  Actor Step: $\theta_{t+1}^i \leftarrow \theta_t^i + \xi_{\theta,t} \cdot \widetilde{\delta_t^i} \cdot \psi_t^i$
  Send $\widetilde{\mu_t^i}, \widetilde{\lambda_t^i}, \widetilde{v_t^i}$ to the neighbors
  Consensus Step:
    $\mu_{t+1}^i \leftarrow \sum_{j\in\mathbb{N}} c_t(i,j) \cdot \widetilde{\mu_t^j}$
    $v_{t+1}^i \leftarrow \sum_{j\in\mathbb{N}} c_t(i,j) \cdot \widetilde{v_t^j}$
    $\lambda_{t+1}^i \leftarrow \sum_{j\in\mathbb{N}} c_t(i,j) \cdot \widetilde{\lambda_t^j}$
  $t \leftarrow t+1$
close;

of the dataset to the format accepted in Mahimahi, and then, we create two interfaces for each agent, one with LTE traces and one with Wi-Fi traces. The task arrival rate at the agent follows a uniform distribution, and in the case of edge offloading, each task transmission is performed running *TCP iperf3* over the emulated link for the size of transmitted data, $D_{i,m}$, of 7 MB.

The channel bandwidth is set to the default value available in Mahimahi (i.e., $W^w = 5$ MHz for the Wi-Fi access, and $W^c = 4$ MHz for LTE). The thermal noise power is set equal for the two technologies, as $(\sigma_{i,m}^c)^2 = (\sigma_{i,m}^w)^2 = 50$ dBm. For the channel gain we have $H_{i,m} = d_{i,s}^{\nu}$, where $d_{i,s}$ denotes the distance between mobile node i and access point s, and $\nu = 4$ is the path loss factor. We then simply set the default values of the weights defined in (15), (23), and (24), so that energy consumption and task completion time have an equivalent importance in the computational cost evaluation, i.e., $\alpha_{i,m}^l = \beta_{i,m}^l = \alpha_{i,m}^c = \beta_{i,m}^c = \alpha_{i,m}^w = \beta_{i,m}^w = 0.5$. For the sake of simplicity we also set $f_{i,m} = 2.3$ GHz for all nodes, $f_e = 3.4$ GHz, and if not otherwise specified, $L_{i,m} = 25 \times 10^9$. The other metrics change over time and are collected when needed. In the following evaluation, the average values are computed after 35 experiments.

Each agent maintains two neural networks for actor and critic, respectively, and both of them have one hidden layer, containing 64 neural units (this number is motivated in the following), and use ReLU as the activation function. While the output layer for the actor network is softmax, that for the critic network is linear. Considering the graph G_t of the N agents, in which, at first, all agents can communicate with the others, we create the consensus weight matrix C_t by normalizing the absolute Laplacian matrix of G_t to be doubly stochastic. The stepsizes for the actor and critic step are set as constants, respectively $\xi_{\theta,t} = 0.001$ and $\xi_{v,t} = 0.01$.

B. Trace-Driven Emulation Results

In the following experiments we compare our solution against other currently deployed algorithms. Among the related studies described in Section II, we select as benchmarks the most similar algorithms using some variants of machine learning-based methods for the offloading process. Specifically, we compare our approach against the DROO framework [18], which implements a deep neural network that learns the binary offloading decisions, and a hotbooting Q-learning based computation offloading scheme [52], that for simplicity we refer to as hotbooting DQN, as it uses a fast deep Q-network (DQN) model to further improve the offloading performance.

Fig. 2a and Fig. 2b show the impact of the UAV swarm size (i.e., the number of agents) on the task completion time and on the utility function defined in (26), which also contemplates the power consumption. Decisions of each agent about whether to offload the task or not, as well as which technology to use for the offloading, are based on the information received from other nodes, according to the cooperative algorithm at the basis of our solution. We can notice how this approach can take full advantage of a rising number of computing nodes, shortening the task completion time and increasing the overall utility. Conversely, for hotbooting DQN occurs the opposite: if a large number of agents are present in the system, the task completion time increases. Besides, with an increasing number of computing nodes, power consumption increases as well. In the DROO case, the two quantities remain almost constant when the number of nodes increases, in any case leading this solution to perform worse than ours. These results show how a proper algorithm for task offloading decisions plays a crucial part in the system performance, and a multi-agent approach to optimize actions more efficiently is a valuable solution.

Besides the dependence on the number of agents, we further examine how the distance between nodes and the antenna affects the performance in Fig. 2 c. We perform experiments for a fleet of 50 nodes, and we can observe how, clearly, the distance degrades the performance of the system because of the higher delays in the communication with the edge cloud. However, in the case of our solution, the curve is flattened, thus further proving its effectiveness in taking the offloading decision. In fact, our state space also includes the distance to the antenna, which is then considered in the decision process.

In the same setting we then consider the energy consumption in Fig. 3 a. The advantages of our solution regarding the energy spent for the computation and the transmission are even more prominent. The system can properly manage the diversity in the locations, as this metric is part of the state variables. This results

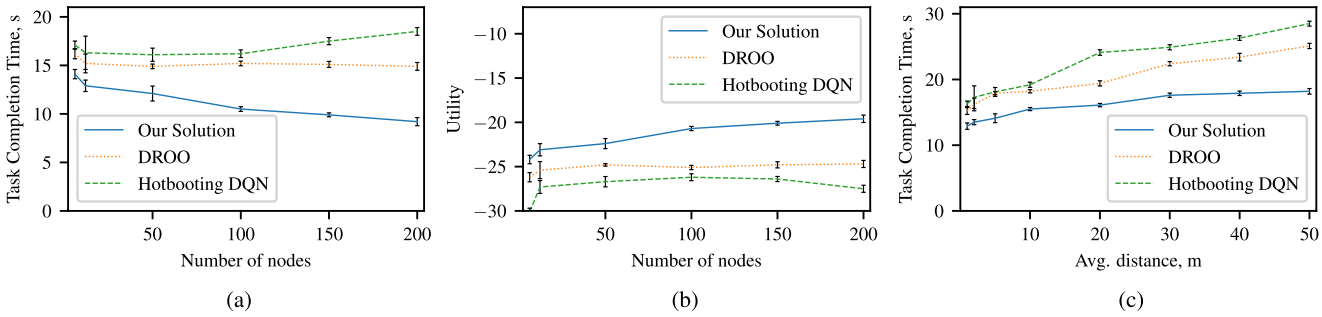


Fig. 2. System performance in terms of (a) task completion time and (b) utility for a varying number of agents and (c) node-antenna distance. The results are compared with similar solutions whose aim is analogous to ours.

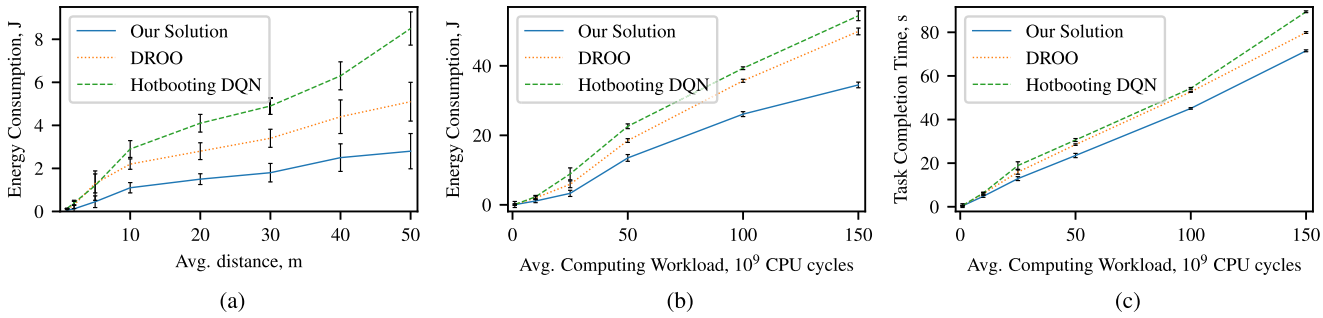


Fig. 3. System performance evaluation. (a) Energy spent for the task computation at varying the node-antenna distance. (b) Energy consumption and (c) application completion time for increasing average computing workload.

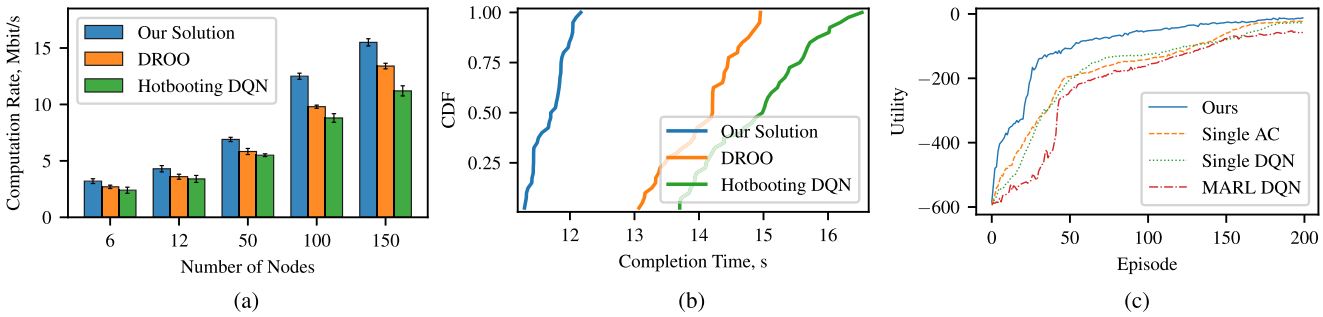


Fig. 4. (a) Comparison in terms of computation rate for various offloading solutions. (b) CDF of the task completion time for the compared solutions. (c) Utility evolution for different RL-based algorithms. Our model can shorten convergence time compared to the alternatives.

in an optimized usage of resources and a decision to offload only when really beneficial. Moreover, we compare the performance of the three solutions with respect to the average computing workload, i.e., the average amount of CPU cycles required to complete the tasks submitted to the system. Fig. 3 b and Fig. 3 c depict the energy consumption and the task completion time, respectively, for the three considered solutions. We can observe how both the energy consumption and the task completion time increases with the average computing workload, for all the considered solutions. However, for DROO and hotbooting DQN, the increment in the energy consumption is notably larger. This is because they do not have the adaptive and control mechanism of energy consumption we have in our model, which adaptively takes the offloading decision in a distributed manner. Fig. 3 a leads to similar conclusions for the task completion time. Although in a less pronounced way with respect to the energy consumption, also the task completion time achieved by our solution increases slowly with the increase of the computing workload.

In light of the previous findings, we can conclude that *the knowledge not only of the states, but also of some model parameters of the other agents (see Section V), improves the decisions of the single agent. In fact, in this way the action can also consider the likely actions of other agents, thus possibly anticipating their future behavior.*

Moreover, we evaluate the computation rate of all the agents, i.e., the number of processed bits within a unit time from the system. In Fig. 4 a we report the computation rate for different algorithms at varying sizes of agents fleet. It is straightforward to observe how our algorithm outperforms the analogous

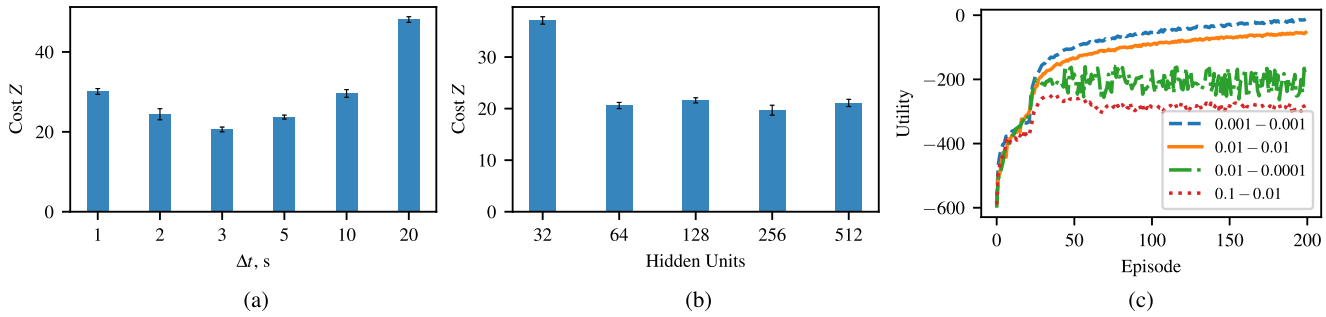


Fig. 5. Sensitivity analysis of the average cost Z and the utility in terms of (a) time interval for model updates, (b) hidden units in the neural networks of actor and critic agents, (c) stepsize for the actor and critic updates. This analysis motivates the choice of our default algorithm parameters.

approaches, and the more agents, the larger the rate improvements compared to the other methods. Although this metric is only implicitly covered by the utility function, our solution offers a high computation rate due to the optimized resource management and distributed approach. In fact, minimizing the computational time for tasks results in better computation rate performance too.

To analyze the variability of performance among nodes, we also evaluate the cumulative distribution function (CDF) of the task completion time for the three considered solutions. Results are reported in Fig. 4 b and refer to a case when the number of nodes is 15. Not only does our approach provide a lower completion time on average, but most of the nodes complete the task at a time close to the average. This small variance is extremely important in UAV systems, especially for real-time applications requiring low and constant task completion times.

For the sake of completeness, we finally compare the convergence performance of our MARL-based method against other possible RL-based algorithms when applied in our solution. Specifically, we consider the following three alternative possibilities. Firstly, *Single AC*, an approach still based on the Actor-Critic (AC) framework, but where each agent takes independent decisions. Secondly, *Single DQN*, a similar approach where the RL algorithm belongs to the class of Value-based methods that exploit Q-values to determine the probabilities of actions and any other parameter of the algorithm. In this class of algorithms, deep Q-network (DQN) is one of the most common methods that integrate deep neural networks into RL, originating the deep reinforcement learning. It has been shown how deep neural networks can empower RL to directly deal with high dimensional states thanks to techniques used in DQN [65]. Finally, *MARL DQN*, which implements the DQN algorithm in a multi-agent context, where the Q-values are transmitted among the agents for a collaborative approach. Fig. 4 c shows the result of this comparison. It is possible to observe how the utility function increases as the number of episodes increases, until it attains a relatively stable value, in all the methods. However, we can notice that our approach provides a higher value for the utility function and that the convergence is faster. *MARL DQN*, for example, despite the cooperation among agents, is unable to properly handle the information of other nodes, whose learning process hardly fits this context. On the other hand, both *Single AC* and *Single DQN* have comparable yet better results with respect to *MARL DQN* due to the simplicity of their approach, which is able to achieve quite fast convergence. However, with local reward and action, classical reinforcement learning algorithms, i.e., *Single AC* and *Single DQN*, fail to maximize the system-wide average reward, whose value is determined by the joint actions of all agents. In conclusion, our algorithm can distribute the information in an efficient way, thus resulting in an appropriate solution for our context.

C. Sensitivity Analysis

We further conduct a sensitivity analysis of the average cost (i.e., $Z = 1/N \sum Z_i$) with respect to the key design parameters (Fig. 5). Firstly, we analyze the impact of the update interval, Δt. This value specifies the rate on which the agents share the information and performs both actor and critic steps. In Fig. 5 a we plot how Δt affects the performance in terms of cost Z. Too frequent updates lead to an improvement in the model but a burden in the system, while a large Δt may neglect state values and undermine the overall model. We can observe how a value of $\Delta t = 3$s is a valuable trade-off, which guarantees adaptability without incurring in too frequent changes.

Secondly, we study how various neural network settings may affect performance. Actor and critic agents utilize two separate neural networks differing in the input and output layers but using the same amount of hidden units for design simplicity. Fig. 5 b shows the cost Z for increasing number of hidden units. These results suggest that the more neurons, the more efficient is the model. However, it may also be considered that a larger neural network requires a larger overhead, e.g., memory footprint, which is not justifiable since the effect in the cost is minimal when the number of units is greater than 64. For this reason, we set the number of hidden units to 64.

Lastly, we investigate the importance of stepsizes $\xi_{\theta,t}$ and $\xi_{v,t}$. The utility for multiple combinations of stepsizes is examined for each episode and reported in Fig. 5 c, which illustrates the utility during the training phase. It is shown that only for the two combinations $0.001 - 0.01$ and $0.01 - 0.1$ the proposed algorithm successfully converges. However, the former one is able to achieve higher utility at a slightly faster speed. Conversely, the other two combinations $0.01 - 0.001$ and $0.1 - 0.01$ have a turbulent evolution and leads to a lower utility compared to the

other values. These results motivate our choice to set the default stepsize values of actor step to 0.001 and critic step to 0.01.

VII. Conclusion

This paper presents a distributed algorithm for the offloading task decision whose aim is to speed up the task completion time and, at the same time, limit the overall energy consumption. To this end, we propose a multi-agent reinforcement learning algorithm to decide whether or not to offload a task to the edge cloud. The overall state of the system is appropriately shared between the nodes and used when each agent has to decide where to perform an assigned task: locally or in the edge cloud by means of an offloading procedure. Each node, in case of task offloading, can further decide the transmission technology to use, Wi-Fi or LTE, according to the current utilization.

Results validate our algorithm, demonstrating the good performance of our system. Our evaluation also shows how the developed algorithm can manage the large quantity of information coming from the environment in an efficient way, thus making our distributed solution a truly viable approach for task offloading decision problems.

References

[1] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Commun. Surveys Tut.*, vol. 19, no. 4, pp. 2322–2358, Fourthquarter 2017.

[2] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, "Mobile edge computing: A survey," *IEEE Internet Things J.*, vol. 5, no. 1, pp. 450–465, Feb. 2018.

[3] T. K. Rodrigues, K. Suto, H. Nishiyama, J. Liu, and N. Kato, "Machine learning meets computation and communication control in evolving edge and cloud: Challenges and future perspective," *IEEE Commun. Surveys Tut.*, vol. 22, no. 1, pp. 38–67, 2019.

[4] A. Sacco, F. Esposito, and G. Marchetto, "Resource inference for task migration in challenged edge networks with RITMO," in *Proc. 9th IEEE Int. Conf. Cloud Netw. (CloudNet)*., 2020, pp. 1–7.

[5] A. Sacco, M. Flocco, F. Esposito, and G. Marchetto, "An architecture for adaptive task planning in support of IoT-based machine learning applications for disaster scenarios," *Comput. Commun.*, vol. 160, pp. 769–778, 2020.

[6] M. Chiang and T. Zhang, "Fog and IoT: An overview of research opportunities," *IEEE Internet Things J.*, vol. 3, no. 6, pp. 854–864, 2016.

[7] Y. Mao, J. Zhang, and K. B. Letaief, "Dynamic computation offloading for mobile-edge computing with energy harvesting devices," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 12, pp. 3590–3605, 2016.

[8] J. Wang, L. Zhao, J. Liu, and N. Kato, "Smart resource allocation for mobile edge computing: A deep reinforcement learning approach," *IEEE Trans. Emerging Top. Computing*, 2019.

[9] A. V. Ventrella *et al.*, "Apron: An architecture for adaptive task planning of internet of things in challenged edge networks," in *Proc. 8th IEEE Int. Conf. Cloud Netw. (CloudNet)*., 2019, pp. 1–6.

[10] C. You, K. Huang, H. Chae, and B. Kim, "Energy-efficient resource allocation for mobile-edge computation offloading," *IEEE Trans. Wireless Commun.*, vol. 16, no. 3, pp. 1397–1411, 2017.

[11] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Trans. Netw.*, vol. 24, no. 5, pp. 2795–2808, 2016.

[12] H. Guo, J. Liu, J. Zhang, W. Sun, and N. Kato, "Mobile-edge computation offloading for ultradense IoT networks," *IEEE Internet Things J.*, vol. 5, no. 6, pp. 4977–4988, 2018.

[13] S. Bi and Y. J. Zhang, "Computation rate maximization for wireless powered mobile-edge computing with binary computation offloading," *IEEE Trans. Wireless Commun.*, vol. 17, no. 6, pp. 4177–4190, 2018.

[14] T. X. Tran and D. Pompili, "Joint task offloading and resource allocation for multi-server mobile-edge computing networks," *IEEE Trans. Veh. Technol.*, vol. 68, no. 1, pp. 856–868, 2018.

[15] S. Guo, B. Xiao, Y. Yang, and Y. Yang, "Energy-efficient dynamic offloading and resource scheduling in mobile cloud computing," in *Proc. IEEE INFOCOM 2016-The 35th Annu. IEEE Int. Conf. Comput. Commun.*, 2016, pp. 1–9.

[16] T. Q. Dinh, J. Tang, Q. D. La, and T. Q. Quek, "Offloading in mobile edge computing: Task allocation and computational frequency scaling," *IEEE Trans. Commun.*, vol. 65, no. 8, pp. 3571–3584, 2017.

[17] B. Liu, W. Zhang, W. Chen, H. Huang, and S. Guo, "Online computation offloading and traffic routing for uav swarms in edge-cloud computing," *IEEE Trans. Vehicular Technol.*, 2020.

[18] L. Huang, S. Bi, and Y. J. Zhang, "Deep reinforcement learning for online computation offloading in wireless powered mobile-edge computing networks," *IEEE Trans. Mobile Comput.*, vol. 19, no. 11, pp. 2581–2591, 2019.

[19] 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 Trans. Veh. Technol.*, vol. 68, no. 8, pp. 8050–8062, 2019.

[20] A. Sacco, F. Esposito, and G. Marchetto, "A distributed reinforcement learning approach for energy and congestion-aware edge networks," in *Proc. 16th Int. Conf. emerging Netw. EXperiments Technol. (CoNEXT)*., 2020, pp. 546–547.

[21] Y. Liu, H. Yu, S. Xie, and Y. Zhang, "Deep reinforcement learning for offloading and resource allocation in vehicle edge computing and networks," *IEEE Trans. Veh. Technol.*, vol. 68, no. 11, pp. 11 158–11 168, 2019.

[22] J. Baxter and P. L. Bartlett, "Direct gradient-based reinforcement learning," in *Proc. IEEE Int. Symp. Circuits and Syst. Emerging Technologies for the 21st Century. Proceedings (IEEE Cat No 00CH36353)*, vol. 3, 2000, pp. 271–274.

[23] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Proc. Adv. Neural Inf. Process. Syst.*, 2000, pp. 1057–1063.

[24] J. Peters and S. Schaal, "Natural actor-critic," *Neurocomputing*, vol. 71, no. 7-9, pp. 1180–1190, 2008.

[25] S. Bhatnagar, M. Ghavamzadeh, M. Lee, and R. S. Sutton, "Incremental natural actor-critic algorithms," in *Proc. Adv. Neural Inf. Process. Syst.*, 2008, pp. 105–112.

[26] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[27] R. Munos, T. Stepleton, A. Harutyunyan, and M. Bellemare, "Safe and efficient off-policy reinforcement learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 1054–1062.

[28] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *Proc. 31st Int. Conf. Int. Conf. Mach. Learn.*, *ser. ICML'14. JMLR.org*, 2014, vol. 32, pp. 387–395.

[29] T. P. Lillicrap *et al.*, "Continuous control with deep reinforcement learning," 2015, *arXiv:1509.02971*.

[30] V. Mnih *et al.*, "Asynchronous methods for deep reinforcement learning," in *Proc. 33rd Int. Conf. Mach. Learn.*, PMLR, 2016, pp. 1928–1937.

[31] M. L. Littman, "Value-function reinforcement learning in Markov games," *Cogn. Syst. Res.*, vol. 2, no. 1, pp. 55–66, 2001.

[32] M. Lauer and M. Riedmiller, "An algorithm for distributed reinforcement learning in cooperative multi-agent systems," in *Proc. Seventeenth Int. Conf. Mach. Learn.*, Morgan Kaufmann Publishers Inc., 2000, pp. 535–542.

[33] J. Hu and M. P. Wellman, "Nash q-learning for general-sum stochastic games," *J. Mach. Learn. Res.*, vol. 4, pp. 1039–1069, Nov. 2003.

[34] J. Foerster, I. A. Assael, N. De Freitas, and S. Whiteson, "Learning to communicate with deep multi-agent reinforcement learning," in *Proc. Adv. Neural Inf. Process. Syst.*, Curran Associates, Inc., 2016, pp. 2137–2145.

[35] J. K. Gupta, M. Egorov, and M. Kochenderfer, "Cooperative multi-agent control using deep reinforcement learning," in *Proc. Int. Conf. Auton. Agents Multiagent Syst.*, Springer, 2017, pp. 66–83.

[36] R. Lowe, Y. I. Wu, A. Tamar, J. Harb, O. P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," in *Proc. Adv. Neural Inf. Process. Syst.*, Curran Associates, Inc., 2017, pp. 6379–6390.

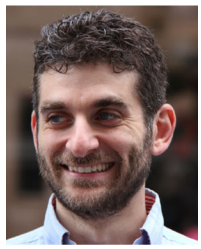
[37] S. Omidshafiei, J. Pazis, C. Amato, J. P. How, and J. Vian, "Deep decentralized multi-task multi-agent reinforcement learning under partial observability," 2017, *arXiv:1703.06182*.

[38] M. Lanctot *et al.*, "A unified game-theoretic approach to multiagent reinforcement learning," in *Proc. Adv. Neural Inf. Process. Syst.*, Curran Associates, Inc., 2017, pp. 4190–4203.

[39] K. Zhang, Z. Yang, H. Liu, T. Zhang, and T. Basar, "Fully decentralized multi-agent reinforcement learning with networked agents," in *Proc. 35th Int. Conf. Mach. Learn.*, vol. 80, 2018, pp. 5872–5881.

[40] Z. Zhang, Z. Hong, W. Chen, Z. Zheng, and X. Chen, "Joint computation offloading and coin loaning for blockchain-empowered mobile-edge computing," *IEEE Internet Things J.*, vol. 6, no. 6, pp. 9934–9950, 2019.
[41] M.-A. Messous, H. Sedjelmaci, N. Houari, and S.-M. Senouci, "Computation offloading game for an uav network in mobile edge computing," in *Proc. IEEE Int. Conf. Commun.*, 2017, pp. 1–6.
[42] J. Zhao, Q. Li, Y. Gong, and K. Zhang, "Computation offloading and resource allocation for cloud assisted mobile edge computing in vehicular networks," *IEEE Trans. Veh. Technol.*, vol. 68, no. 8, pp. 7944–7956, 2019.
[43] M. Gong and S. Ahn, "Computation offloading-based task scheduling in the vehicular communication environment for computation-intensive vehicular tasks," in *Proc. Int. Conf. Artif. Intell. Inf. Commun.*, 2020, pp. 534–537.
[44] E. Coronado, G. Cebrian-Marquez, and R. Riggio, "Enabling computation offloading for autonomous and assisted driving in 5G networks," in *Proc. IEEE Glob. Commun. Conf.*, 2019, pp. 1–6.
[45] S. Barbarossa, S. Sardellitti, and P. Di Lorenzo, "Communicating while computing: Distributed mobile cloud computing over 5G heterogeneous networks," *IEEE Signal Process. Mag.*, vol. 31, no. 6, pp. 45–55, 2014.
[46] O. Muñoz, A. Pascual-Iserte, and J. Vidal, "Optimization of radio and computational resources for energy efficiency in latency-constrained application offloading," *IEEE Trans. Veh. Technol.*, vol. 64, no. 10, pp. 4738–4755, 2015.
[47] N. Kalatzis, M. Avgeris, D. Dechouniotis, K. Papadakis-Vlachopapadopoulos, I. Roussaki, and S. Papavassiliou, "Edge computing in iot ecosystems for uav-enabled early fire detection," in *Proc. IEEE Int. Conf. Smart Comput.*, 2018, pp. 106–114.
[48] B. Zhou, A. V. Dastjerdi, R. N. Calheiros, and R. Buyya, "An online algorithm for task offloading in heterogeneous mobile clouds," *ACM Trans. Internet Technol.*, vol. 18, no. 2, pp. 1–25, 2018.
[49] H. Ke, J. Wang, L. Deng, Y. Ge, and H. Wang, "Deep reinforcement learning-based adaptive computation offloading for mec in heterogeneous vehicular networks," *IEEE Trans. Veh. Technol.*, vol. 69, no. 7, pp. 7916–7929, 2020.
[50] X. Zhu, Y. Luo, A. Liu, M. Z. A. Bhuiyan, and S. Zhang, "Multi-agent deep reinforcement learning for vehicular computation offloading in IoT," *IEEE Internet Things J.*, 2020.
[51] L. Huang, X. Feng, A. Feng, Y. Huang, and L. P. Qian, "Distributed deep learning-based offloading for mobile edge computing networks," *Mobile Netw. Appl.*, pp. 1–8, 2018.
[52] M. Min, L. Xiao, Y. Chen, P. Cheng, D. Wu, and W. Zhuang, "Learning-based computation offloading for iot devices with energy harvesting," *IEEE Trans. Veh. Technol.*, vol. 68, no. 2, pp. 1930–1941, 2019.
[53] J. Franz, T. Nagasuri, A. Wartman, A. V. Ventrella, and F. Esposito, "Reunifying families after a disaster via serverless computing and raspberry PIS," in *Proc. IEEE Int. Symp. Local Metrop. Area Netw.*, 2018, pp. 131–132.
[54] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Int. Things J.*, vol. 3, no. 5, pp. 637–646, 2016.
[55] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Hoboken, NJ, USA: John Wiley & Sons, 2014.
[56] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," 2015, *arXiv:1506.02438*.
[57] A. P. Miettinen and J. K. Nurminen, "Energy efficiency of mobile clients in cloud computing," in *Proc. 2nd USENIX Conf. Hot Top. Cloud Comput. (HotCloud). USENIX Assoc.*, 2010, pp. 1–7.
[58] D. Huang, P. Wang, and D. Niyato, "A dynamic offloading algorithm for mobile computing," *IEEE Trans. Wireless Commun.*, vol. 11, no. 6, pp. 1991-1995, 2012.
[59] A. Rudenko, P. Reiher, G. J. Popek, and G. H. Kuenning, "Saving portable computer battery power through remote process execution," *ACM SIGMOBILE Mobile Comput. Commun. Rev.*, vol. 2, no. 1, pp. 19–26, 1998.
[60] W. Li, F. Zhou, K. R. Chowdhury, and W. M. Meleis, "Qtcp: Adaptive congestion control with reinforcement learning," *IEEE Trans. Netw. Sci. Eng.*, vol. 6, no. 3, pp. 445–458, 2018.
[61] J. Tsitsiklis, D. Bertsekas, and M. Athans, "Distributed asynchronous deterministic and stochastic gradient optimization algorithms," *IEEE Trans. Autom. Control*, vol. 31, no. 9, pp. 803–812, 1986.
[62] S. Boyd, N. Parikh, and E. Chu, *Distributed Optimization and Statistical Learning Via the Alternating Direction Method of Multipliers*. Hanover, MA, USA: Now Publishers Inc, 2011.
[63] *Cell vs WiFi*. Accessed: Mar. 13, 2021. [Online]. Available: http://web.mit.edu/cell-vs-wifi/
[64] R. Netravali *et al.*, "Mahimahi: Accurate record-and-replay for http," in *Proc. USENIX Annu. Tech. Conf. (USENIX ATC 15)*, 2015, pp. 417–429.
[65] Y. Sun, M. Peng, and S. Mao, "Deep reinforcement learning-based mode selection and resource management for green fog radio access networks," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 1960–1971, 2018.

Alessio Sacco (Student Member, IEEE) received the M.Sc. degree in computer engineering from the Politecnico di Torino, Turin, Italy, where he is currently working toward the Ph.D. degree in computer engineering. His research interests include architecture and protocols for network management, implementation and design of cloud computing applications, algorithms and protocols for service-based architecture, which include software defined networks, used in conjunction with machine learning algorithms.



Flavio Esposito (Member, IEEE) received the M.Sc. degree in telecommunication engineering from the University of Florence, Florence, Italy and the Ph.D. degree in computer science from Boston University, Boston, MA, USA, in 2013. He is currently an Assistant Professor with the Department of Computer Science, Saint Louis University (SLU), St. Louis, MO, USA. He also has an affiliation with the Parks College of Engineering, SLU. He was with the industry for a few years, and his main research interests include network management, network virtualization, and distributed systems. He was the recipient of several awards, including four National Science Foundation awards and two Best Paper awards, one at IEEE NetSoft 2017 and one at IEEE NFV-SDN 2019.

Guido Marchetto (Senior Member, IEEE) received the Ph.D. degree in 2008 in computer engineering from the Politecnico di Torino, Turin, Italy, where he is currently an Associate Professor with the Department of Control and Computer Engineering. In 2009, he visited the Department of Computer Science, Boston University, Boston, MA, USA. His research interests include distributed systems, formal verification of systems and protocols, network protocols and network architectures. He is an Associate Editor for the IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY.

Paolo Montuschi (Fellow, IEEE) is currently a Full Professor with the Department of Control and Computer Engineering, Rector's Delegate for Information Systems, and a past Member of the Board of Governors Politecnico di Torino, Turin, Italy. His research interests include computer arithmetic, computer graphics, and intelligent systems. He is a Life Member of the International Academy of Sciences, Turin, Italy, and of HKN, the Honor Society of IEEE. He is the Editor-in-Chief of the IEEE TRANSACTIONS ON EMERGING TOPICS IN COMPUTING, the 2020–21 Chair of the IEEE TAB/ARC and the Co-Chair of the 2021 TAB/PSPB Ad Hoc Committee on Publications Strategy. From 2015 to 2018, he was in a number of positions, including the Editor-in-Chief of the IEEE TRANSACTIONS ON COMPUTERS, from 2017 to 2020, the IEEE Computer Society Awards Committee Chair, from 2018 to 2020, a Member-at-Large of the IEEE PSPB, and from 2019 to 2020, the Chair of its Strategic Planning Committee.