A Self-Learning Strategy for Task Offloading in UAV Networks

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

***Abstract*—The edge computing paradigm has opened new opportunities for IoT devices, which can be used in novel applications involving heavy processing of data. Typical and common examples of IoT devices are the Unmanned Aerial Vehicles (UAVs), which are deployed for surveillance and environmental monitoring and are attracting increasing attention because of their ease deployment. However, their limited capacity, e.g., battery, forces the design of an edge-assisted solution, where heavy tasks are offloaded to the edge cloud. To solve the problem of task offloading from UAV to the closest edge computation, many proposals have appeared, mainly based on a Reinforcement Learning (RL) formulation. While these solutions successfully learn how to reduce task completion time in the UAV context, some limitations appear when these models are applied in real scenarios, given the memory-hungry nature of RL. To this end, we propose a simple yet effective formalization that still enables a learning process, but reduces the required information and the training time. Our evaluation results confirm our hypothesis, showing a marked improvement when compared to other RL-based strategies and deep learning-based solutions.**

***Index Terms*—Machine learning, task offloading, time series, UAV.**

I. Introduction

THE past decade has witnessed an explosive growth in mobile internet applications consuming a significant amount of computational resources, e.g., face recognition, virtual/augmented reality, realtime media streaming, mainly favored by the development of the Internet of Things (IoT). A specific area of interest entails vehicles and, in particular, Unmanned Aerial Vehicle (UAV) systems, that have experienced a constantly increasing popularity in the last years, mainly thanks to their maneuverability, flexibility, and limited deployment costs. UAVs have been primarily used for military applications, but they are now expanding into business, science, agriculture, and civilian fields, where successful examples include supports of first responders, surveillance, aerial photography to cite a few [1]. Their constrained resources, however, open the problem of offloading part of their tasks to the close multi-access edge computing (MEC) in order to speed up the computation.

The problem of task offloading has been extensively studied in the literature [2]–[6], where recent solutions attempt to significantly reduce the processing time of mobile vehicle applications while greatly reducing data processing delays and energy consumption. With the advent of machine learning (ML) and, specifically, reinforcement learning (RL), this learning approach became dominant in solving the offloading decisions in vehicular scenarios. Compared to traditional approaches based on heuristics, these solutions have shown the ability to learn the best strategies adapting to the challenging and highly varying environments [7]–[10]. Despite the good results of newly computation offloading techniques, however, RL-based methods have a severe impact on the memory and processing usage of the mobile nodes. Moreover, it still remains challenging to develop a reliable system that can anticipate future demands and take advisable computation offloading decisions.

In this paper, we present a self-learning strategy that supports the UAV during the decision of offloading incoming tasks. This decision is taken on the basis of the predicted behavior of the agent, suggesting whether edge cloud is beneficial or not to the incoming tasks. Two alternative methods are designed to perform a prediction about future device load: a model belonging to time-series class, i.e., Vector Autoregressive Moving-Average (VARMA), and a model belonging to the class of ML regressors, i.e., Random Forest Regression (RFR). In such a way, not only the agent learns how to forecast future values, but it can also learn online what type of model is more accurate, following a paradigm known as Follow the Perturbed Leader (FPL). Having chosen two different ways in treating the input metrics, this approach also provides flexibility and adaptability, resulting in a learning agent that can select which predictor best fits a particular environment.

While other RL-based models can be computationally expensive to run on board of constrained resources devices, our formulation simplifies the decision process. The results illustrate clear advantages in the implementation of our approach, which can shorten the time required to accomplish a task. Besides, our solution can reduce the energy consumed and the resource usage, i.e., memory and CPU, compared to other benchmark algorithms. These benefits are originated by our predictor, which outperforms alternatives, leading to a small error and a very accurate decision.

Manuscript received September 8, 2021; revised November 30, 2021; accepted January 11, 2022. Date of publication January 25, 2022; date of current version May 2, 2022. This work was supported by NSF under Awards CNS-1836906 and CNS-1647084. The review of this article was coordinated by Dr. M. Fouda.

Alessio Sacco, Guido Marchetto, and Paolo Montuschi are with 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.2022.3144654

The remainder of this article is organized as follows. In Section II we discuss the existing literature on the task offloading problem. Section III presents the model of the system and formulates the specific problem we need to solve. In Section IV we describe the methods used to predict future behavior and in Section V we explicit how they are used in our algorithm. Results are presented in Section VI. Finally, we conclude our paper in Section VII.

II. Related Work

In the last years, edge computing has been proved to be an effective method in shortening task completion time for some latency-critical applications [11]–[13]. This paradigm can be particularly beneficial for unmanned aerial systems (UASs), e.g., self-driving vehicles and UAV swarms, to conduct a computation offloading scheme with edge computing. UASs are often used for collecting data and sending them to the close edge for data-intensive visual and acoustic computing. At network edges, indeed, there may be present more resources that are not available over the UAS, and that can thus speed up the processing. For example, seamless processing of imagery/video at the network edge is particularly critical in natural or human-made disaster scenarios, where bandwidth is limited and network conditions are highly variable. In such a case, the more powerful resources at the edge cloud are employed for elaborating imagery to recognize body temperatures or identify bodies under ruins or massive avalanches.

In general, the goal of performance-based offloading policies is to enhance the performance of mobile devices in terms of execution/completion time and throughput by utilizing cloud resources. Therefore, the resource-intensive computations are offloaded to the cloud or close edge. In [14], the authors propose a novel technique based on compiler code analysis that optimizes the overall execution time by dynamically offloading part of Android code running on a smartphone to the cloud. Similarly, [15] presents a framework providing run-time support for the dynamic computation partitioning and execution of the application. Such a framework not only allows the dynamic partitioning for a single user, but also supports the sharing of computation instances among multiple users in the cloud for an efficient utilization of the underlying cloud resources.

Other approaches can be found in [16]–[18]. In [16] the focus is on the mobile-edge computation offloading (MECO) problem, proposing a two-tier game-theoretical greedy approximation offloading scheme. Considering an ultradense IoT network, the authors formulate the optimal MECO problem as a constrained optimization problem, which aims to minimize the overall computation overhead while satisfying the given wireless channel constraints. An online algorithm is instead presented in [18], where task offloading decisions are modeled as a well-known sub-problem called *rent/buy* problem [19]. Similarly, [20] introduces a joint optimization scheme for the offloading decision and energy consumption, that is based on a greedy heuristic algorithm. After having modeled the problem aiming to minimize the energy, the proposed algorithm is based on the Reformulation-Linearization-Technique based Branch-and-Bound method (RLTBB). Via a greedy heuristic, the system can allocate the radio resource and computational resource among smart mobile devices of the searching set.

This offloading process can be further optimized via the application of ML/AI methods, as explained in recent studies [21]–[23]. One of the most profitable recent trends is the utilization of RL for a task offloading solution, given its ability to adapt to highly dynamic environments [22], [24]. For example, in [7], the authors 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. Focusing on the industrial scenario, [25] jointly takes both the network traffic and computation workload of industrial traffic into consideration, and finds a trade-off between energy consumption and service delay. To solve this offloading decision problem, they propose a dynamic RL scheduling algorithm combined with a deep dynamic scheduling. Similarly, [26] deployed a task offloading framework using the multi-armed bandit (MAB) theory, which enables vehicles to learn the potential task offloading performance of neighboring vehicles. Redesigning the utility function of the classic MAB algorithm, it can adapt to the volatile environment and minimize the average offloading delay.

Besides, although these solutions are sound, there is now an attempt to distribute the decision logic, in order to improve the performance while reducing the burden for a single UAV. To this end, DDLO [27] and a hotbooting Q-learning based schema [28], are valuable examples of distributed approaches in task offloading decisions, that use multiple parallel deep neural networks. In a similar way, but for collaborative offloading decisions, [10] presents a solution for multiple heterogeneous agents with potentially distinct policies and rewards, and further improvements on protocol decisions.

As an alternative or in conjunction with an RL problem, there is an increment of solutions attempting to predict future resources utilization in order to take better offloading decisions and allocate proper network and computation resources [6], [29], [30]. For example, [6] showed how time-series regressors can be used in predicting future agent's loads to efficiently balance the load over a UAV system. We share with these approaches the idea of proactively acting to mitigate possible performance degradation before its occurrence. However, although all these solutions are effective, they are also computationally and memory intensive, especially the approaches based on the use of RL and deep learning. Our proposed self-learning-based methodology can reduce the overall resource consumption and thus be more suitable for UAV execution.

III. System Model

The considered system consists of a UAV swarm including 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. The overall system is then composed of M tasks, denoted by a set of tasks $\mathcal{M} = \{T_1, \ldots, T_M\}$. We consider that

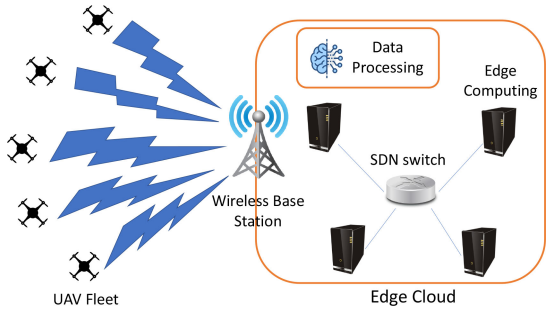

Fig. 1. System Overview. The mobile devices, e.g., UAVs, interacts with the edge cloud asking help for the processing of the collected data.

tasks are independent among them. Each task is assigned to a node, which can decide either to compute the task locally or to offload the computation to the edge cloud.

To capture real-world scenarios, we consider a limited capacity of nodes. We model this constraint in resources as a finite queue where to store waiting tasks. Thus, we denote the amount of tasks of the i-th node as s_i within $[0, s_i^{max}]$, $s_i^{max} \in \mathbb{R}^+$. We also assume that the agent can execute only one task at a time, and, to avoid burdening the notation, task deadlines have not been considered.

For each new task arriving to a node, it has to decide where to perform the computation of such a task. We denote the computation offloading decision of task m of mobile device i by $o_{i,m}$. Specifically, $o_{i,m} = 1$ means that the node offloads the task to the close cloud, while $o_{i,m} = 0$ means that the node executes the task locally.

We summarize the main components of the system in Fig. 1. The UAV fleet relies on the close edge cloud for shortening the task completion time via task offloading. In such a case, the task is sent to the edge, where the appropriate network and computational infrastructure resides. The sent data is thus used for extrapolating helpful information by means of AI/ML algorithms. For clarity, we report in Table I all the symbols used throughout the paper.

TABLE I
SYMBOLS AND NOTATIONS

Symbol	Description
$\mathcal{N}_t$	Set of agents at time t
$\mathcal{M}$	Set of tasks
$T^l_{i,m}$	Task completion time for local computation
$T^e_{i,m}$	Task completion time for offloading computation
s_i^{max}	Maximum amount of possible enqueued tasks in node i
s_i	Number of enqueued tasks in node i
x_t	Observation at timestamp t
y_t	Prediction for timestamp t
$o_{i,m}$	Offloading decision of node i
$T^{l,exec}_{i,m}$	Local execution time of task m on node i
$C_{i,m}$	Computing workload of task m on node i
$T^{wt}_{i,m}$	Local waiting time of task m on node i
$T^{e,tra}_{i,m}(A)$	Transmission time of task m
$T^{e,exec}_{m}$	Execution time of task m in the edge cloud
$T^{e,rec}_{i,m}(A)$	Reception time of task m
W^{up}	Wireless uplink channel bandwidth
W^{down}	Wireless downlink channel bandwidth
$r^{e,up}_{i,m}(A)$	Wireless uplink data rate for offloading of task m
$r^{e,down}_{i,m}(A)$	Wireless downlink data rate for offloading of task m
σ^{up}	Background noise power in the uplink channel
σ^{down}	Background noise power in the downlink channel
$G^{up}_{i,m}$	Channel gain from node i to the access point
$G^{down}_{i,m}$	Channel gain from the access point to the node i
$p^{up}_{i,m}$	Transmission power of node i to the edge cloud
$p^{down}_{i,m}$	Transmission power of the edge cloud to the node i
$d_{i,s}$	Distance between node i and access point s

TABLE II
PARAMETERS SETTING

Parameter	Values
Number of nodes	2, 3, **5**, 7, 10, 20, 50
Task arrival rate (Hz)	0.1, **0.2**, 0.3, 0.9
Nodes' Average Distance [m]	1, 2, 3, 5, **10**, 20, 30, 40
Computing workload, 10^9	0.5, **1**, 10, 25, 50, 100
Channel bandwidth (Mbps)	**5**
Noise power (dBm)	**50**
Number of trials	**35**
Confidence interval [%]	**90**

A. Local Execution

When the task is locally executed, 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 aboard the agent,

$$T^l_{i,m} = T^{l,exec}_{i,m} + T^{wt}_{i,m}, \tag{1}$$

where $T^{l,exec}_{i,m}$ and $T^{wt}_{i,m}$ are the execution time and the waiting time, respectively. Formally, the waiting time of a task is defined as the time that task m spends on board of i before its execution, and mainly depends on the enqueued tasks.

On the other hand, given $C_{i,m}$ the computing workload, i.e., the total number of CPU cycles needed to accomplish task m of node i, the local execution time of task m on node i is hence given by:

$$T^{l,exec}_{i,m} = \frac{C_{i,m}}{f_{i,m}}, \tag{2}$$

where $f_{i,m}$ is the computation capability, i.e., the clock frequency of the CPU chip, of node i, on task m. Our model allows different mobile devices to have different computational capacities with different clock frequencies per task.

B. Edge Cloud Offloading

In case the mobile node offloads the task to the edge cloud, the latter executes the computation task and returns the results to the device. This process entails three phases: *(i)* the transmission phase, *(ii)* the edge computation phase, *(iii)* the outcome receiving phase. Before defining the resulting completion time, it must be noted that this time is affected by the joint action space of all agents, A, also referred to as global action profile. Therefore, given the global action profile A, the completion time for the edge offloading is the sum of these three phases, as such:

$$T^e_{i,m} = T^{e,tra}_{i,m}(A) + T^{e,exec}_{m} + T^{e,rec}_{i,m}(A), \tag{3}$$

where $T_{i,m}^{e,tra}(A)$ refers to the transmission of task m to the edge e; $T_m^{e,exec}$ is the execution time in the edge, and $T_{i,m}^{e,rec}(A)$ is the reception time.

Analyzing these parts in order, we start defining the transmission time for task offloading as:

$$T_{i,m}^{e,tra}(A) = \frac{D_{i,m}^{in}}{r_{i,m}^{e,up}(A)}, \tag{4}$$

where $D_{i,m}^{in}$ denotes the size of computation data sent over the channel (e.g., the recorded audio in UAVs swarm) related to computation task m of node i, and $r_{i,m}^{e,up}(A)$ is the uplink data rate.

Then, we consider the data rate affected by both the background noise power and the channel gain, as in other studies [10], [16]. Thus, given the global action profile A for any node i and task m, we can obtain the wireless uplink data rate for computation offloading of task m of mobile device i as:

$$r_{i,m}^{e,up}(A) = W^{up} \cdot \log_2 \left(1 + \frac{p_{i,m}^{up} G_{i,m}^{up}}{\sigma^{up} + \sum\limits_{j \neq i, k \neq m, o_{j,k}=1} p_{j,k}^{c} G_{j,k}^{up}} \right), \tag{5}$$

where $p_{i,m}^{up}$ is the transmission power of node i offloading task m to the edge cloud; $G_{i,m}^{up}$ denotes the channel gain from node i to the access point when transmitting task m, mainly affected by the path loss and shadowing attenuation; σ^{up} indicates the background noise power, and W^{up} is the wireless uplink channel bandwidth. Clearly, we can observe from the formula that when many mobile devices offload their tasks to the edge simultaneously, the nodes can experience severe interference and low data rates.

Subsequently, the task arrives to the edge that proceeds with the execution. Although the offloaded task needs likely to wait before it is assigned to the proper resource in the cloud for the execution, in the following we omit this waiting time for simplicity, as it is negligible with respect to the other quantities involved. Thus, we can derive the computation execution time for task m in the edge cloud as:

$$T_m^{e,exec} = \frac{C_{i,m}}{f_e}, \tag{6}$$

where f_e denotes the clock frequency of the edge cloud, assuming that the frequency does not change during the computation and is constant over time.

Finally, the results of the computation is sent back to the mobile device, incurring in a reception time defined as:

$$T_{i,m}^{e,rec}(A) = \frac{D_{i,m}^{out}}{r_{i,m}^{e,down}(A)}, \tag{7}$$

where $D_{i,m}^{out}$ denotes the size of obtained output data sent over the channel and $r_{i,m}^{e,down}(A)$ is the downlink data rate. Such a wireless downlink data rate is given by:

$$r_{i,m}^{e,down}(A) = W^{down} \cdot \log_2 \left(1 + \frac{p_{i,m}^{down} G_{i,m}^{down}}{\sigma^{down} + \sum\limits_{j \neq i, k \neq m, o_{j,k}=1} p_{j,k}^{c} G_{j,k}^{down}} \right), \tag{8}$$

where $p_{i,m}^{down}$ is the transmission power of the edge cloud communicating the results of offloaded task m to the node i; $G_{i,m}^{down}$ refers to the channel gain from the access point to the node i when transmitting data of task m; σ^{down} denotes the background noise power, and W^{down} indicates the wireless downlink channel bandwidth.

C. Problem Formulation

We formulate the optimization problem that aims to minimize the total delay in finishing all devices' tasks, by optimizing each node offloading decisions $o_{i,m}$:

$$\min_{o_{i,m}} \sum_{i \in \mathcal{N}} \sum_{m \in \mathcal{M}} (1 - o_{i,m}) T_{i,m}^{l} + o_{i,m} T_{i,m}^{e} \tag{9}$$

$$\text{s.t.} \quad \sum_{i=1}^{N} o_{i,m} \leq 1 \quad \forall m \in \mathcal{M}, \tag{10}$$

$$\sum_{m=1}^{M} o_{i,m} \leq s_i^{max} \quad \forall i \in \mathcal{N}, \tag{11}$$

where the constraints (10) and (11) force the solution to *(i)* mutually choose if offloading task computation or executing the task locally, and *(ii)* not to exceed the resources of the mobile device, respectively.

The given optimization problem (9)–(11) can be solved to find results of offloading decision variables $o_{i,m}$. However, since the decision variables are binary, the formulated problem is not convex. Moreover, we would like to consider realistic scenarios where the interaction between devices, the communication channel conditions, and the nodes computation abilities are all dynamically changing. Given these considerations, in the following we propose a online learning method to solve this problem.

IV. Regression Prediction Methods

With the aim of improving offloading decisions, each device predicts future conditions in order to verify is beneficial circumstances hold or not. The node can listen to the advice coming from two different class of predictors and obtain the best from both of them. In particular, we select two algorithms belonging to the class of time-series and to ML supervised regressors. In the following, we describe how these two methods behave, explaining why and where they differ.

A. Time-Series Analysis With VARMA

To model the evolution of data over time, we employ a Vector Autoregressive Moving-Average (VARMA) model. VARMA models are the multivariate generalization of univariate autoregressive-moving average (ARIMA) models. However, while ARIMA is used to represent stationary time series in

almost all domains where a variable is measured at equidistant times, VARMA can contemplate multiple parallel time series, for a multivariate evolution. This class of models well fit problems in econometrics and financial markets, but boasts a wide exploration even in other fields since the 1970's [31]. Our solution, then, uses a VARMA model for "real time" model predictions (hindcasts) that are made within the independent dataset, using only data up to that date were used. The general form of VARMA(r, q) is given by the following equation:

$$y_t = A_1 y_{t-1} + \ldots + A_r y_{t-r} + B_0 \epsilon_t + \ldots + B_q \epsilon_{t-q}, \quad (12)$$

where y_t denotes an $n \times 1$ vector of observed variables, ϵ_t is an $n \times 1$ vector of unobserved disturbances $\sim IID(O_{n\times 1}, I_n)$, where I_n denotes the $n \times n$ identity matrix, r and q denote any assumed nonnegative integers, such that at least one of r or q is positive.

In our solution, we predict the future values of the series by means of a forecasting method named minimization of the Mean Squared Forecast Error (MSFE), which denote the goodness of the prediction using the cumulative error encountered so far. The current information from the dataset, which constitutes the current knowledge, contains the current and past values of the series. In detail, we are focused on the one-step-ahead prediction, which just considers the prediction at the next time step, i.e., y_{t+1} given the last observation at time t.

B. ML Regression With RFR

The Random Forest Regression (RFR) is a type of additive model that predicts by combining decisions from a sequence of base models. More formally, this class of algorithms can be written as:

$$g(x) = f_0(x) + f_1(x) + f_2(x) + \ldots, \quad (13)$$

where the final model g is the sum of simple base models f_i. Although each base model f_i can be any ML algorithm, the most common version of RFR considers as f_i a simple decision tree. In this paper, we also consider this setting. This broad technique of using multiple models to obtain better predictive performance is also known as model ensembling. Moreover, in RFR, all the tree base models are constructed and trained independently using a different subset of data.

Predictions are, then, made by averaging the predictions of each decision tree. In other words, to extend the analogy–much like a forest is a collection of trees, the random forest model is also a collection of decision tree models. This makes random forests a strong modeling technique that is much more powerful than a single decision tree. RFR is suitable for regression problems given its features: *(i)* it can capture non-linear or complex relationships between inputs and outputs, *(ii)* compared to a single decision tree, RFR is more robust, with a limited dependence to the noise in the training set, as it uses a set of uncorrelated decision trees, *(iii)* it is able to limit both the variance and the bias, better addressing the problem of overfitting.

C. State Variables in Our Solution

In our system, the current knowledge Y is modeled as a matrix of features, with a shape $N \times M$, where the column j represents the list of metrics gathered for task i, given i the index of the row. Such a list of features for task i are three: *(i)* the number of enqueued tasks when task i arrived to the node, *(ii)* time to complete task i, in seconds, *(iii)* a boolean stating if task i has been offloaded to the edge cloud.

Features selection is a key topic when dealing with Big Data, demanding for a trade-off between having a vast knowledge and time and resource constraints. This imposes to limit the complexity, with little or none effect on the performance. In fact, a smaller M yields simpler models, but it may be inadequate to represent the space of possible behaviors. On the other hand, a large M leads to a more complex model with more parameters, but may, in turn, lead to overfitting issues. While in time-series this choice for observed variables is much easier, as it usually entails the interested variable, in ML features selection is much more important. Our choice of $M = 3$ and metrics that are extremely easy to collect, moves towards this direction. Each node can train its model without the need to communicate with others, attaining null communication overhead and modest memory occupation. Results support this choice, achieving higher accuracy while reducing noise (Section VI).

However, given the differences in the two models, they also treat the input data differently. Regarding the VARMA model, its input $y_t, y_{t-1}, \ldots, y_1$, is modeled with a vector of metrics at timestamp t. This means that the input of the VARMA model consists of all the values in the matrix. When the number of rows of Y exceeds a threshold ($Z = 1000$), the considered temporal window is limited to a sub-matrix consisting of the last Z rows. Alternatively, regarding the RFR, since it is agnostic of the time order, it only considers the last line of the matrix Y, i.e., the input of the model is composed by all the columns for the last row of the matrix.

V. Agent's Decision Process

In the following, we first overview the procedure as in the Follow the Perturbed Leader (FPL) method. Then, we describe how the IoT agent implements our version of FPL in our system.

A. Follow the Perturbed Leader

Learning from a constant flow of data is considered one of the central challenges of machine learning. Online learning entails sequentially decide on actions given the changes in the environments. In past years, a variety of online learning algorithms have been devised [32], [33]. Among them, in our work we investigate Follow the Perturbed Leader algorithm, whose advantage is its simplicity and computational efficiency.

Such a prediction with expert advice proceeds as follows. At each time step t the system performs sequential predictions $y_t \in \mathcal{Y}$. At times $t = 1, 2, \ldots$, we have access to the predictions $(y_t^i)_{1\leq i\leq n}$ of n experts $\mathcal{E} = e_1, \ldots, e_n$. After having made a prediction, we receive observation $x_t \in \mathcal{X}$, and the system computes our suffered loss $l(x_t, y_t)$ and each expert's loss $l(x_t, y_t^i)$.

As our observations entail continuous values, i.e., lie in a regression problem, the loss is calculated as: $l(x_t, y_t) = (y_t - x_t)^2$.

Our goal can be summarized in achieving a total loss "not much worse" than the best expert, after T time steps. More formally, we denote the cumulative loss of expert i by $L_i^T = \sum_{t=1}^{T} l(x_t, y_t^i)$ and the cumulative loss of our system by $L^T = \sum_{t=1}^{T} l(x_t, y_t)$. Thus, the goal of the system is to minimize the regret, defined as the difference between the cumulative loss of the learner and the cumulative loss of the best prediction in hindsight. The regret over T rounds is defined as:

$$R_T = \sum_{t=1}^{T} l(x_t, y_t) - \min_{i \in 1..n} \sum_{t=1}^{T} l(x_t, y_t^i) = L^T - \min L_i^T. \tag{14}$$

The term $\min L_i^T$, is often defined as the loss of the *best expert in hindsight (BEH)*. Moreover, when this regret is sublinear, namely $R_s \leq o(T)$, the learning algorithm is said to be *Hannan-consistent*.

One algorithm for achieving Hannan-consistency is Follow the Perturbed Leader (FPL), as demonstrated by Hannan [34] and Kalai and Vempala [35]. Let γ be some n-dimensional random variable and $\eta_i > 0$. FPL involves picking the expert exp that minimizes the perturbed cumulative loss:

$$exp = \arg\min_i (L_i + \eta\gamma_i) \tag{15}$$

Intuitively, if η is small, then we expect exp to be "close" to a minimizer of the (non-perturbed) cumulative loss. On the other hand, when η is large, we expect $\mathcal{E}$ to be "close" to the uniform distribution. Namely, η controls how similar the algorithm is to Follow the Leader, the version of the algorithm that always picks the expert who has minimized the cumulative loss. However, this version, and any other deterministic learning algorithm, is not Hannan-consistent [36].

We summarize our version of FPL in Algorithm 1. As demonstrated by theorems in [36], [37], for all possible sequences of losses where the loss is bounded and the noise is spread out, i.e. the noise has a sufficiently high variance, FPL achieves an expected regret that is bounded by:

$$\mathbb{E}[R_t] \in \mathcal{O}(\sqrt{T}). \tag{16}$$

Consequently, this result is also valid for our Algorithm 1, where we consider the random value γ sampled from a Gaussian distribution $\mathcal{N}(0, I)$.

Furthermore, it can be noted that this problem is similar to the Multi-Armed Bandit (MAB), in the class of RL methods. FPL follows the "arm" that is assumed to have the best performance so far, adding exponential noise to it to provide exploration. However, while the MAB algorithm offers more strict bounds to the regret, FPL can drastically simplify the entire learning process making it suitable for constrained agents as the UAVs. For example, differently from MAB that may take a long time to converge, FPL requires a shorter time, and mostly, is not eager of computation and memory resources as is the MAB.

Algorithm 1: Follow the Perturbed Leader `fpl(m: task, s:state, t: time)`.

$\eta > 0$, $L_i \leftarrow 0$
for every expert i **do**
  Compute loss l of last prediction given evidence s
  Accumulate the loss $L_i^t \leftarrow L_i^{t-1} + l$
  Sample $\gamma_i \sim \mathcal{N}(0, I)$
$exp \leftarrow \arg\min_i (L_i^t + \eta\gamma_i)$
Predict y_t asking to the expert exp given state s and task m
return y_t

Algorithm 2: Overall Algorithm.

Initialize threshold T_i for all nodes
for all $i \in \mathcal{N}$ **do**
  wait new task m
    Monitor the queue and node state
    $p \leftarrow$ `fpl(m, s, t)`
    if $p > T_i$ **then**
      Offload task m to the edge cloud
    else
      Enqueue task m locally
    Store states for the future
  End Wait

B. Our Algorithm

We can now present our algorithm, built upon the Follow the Perturbed Leader (FPL) formulation, which dictates the offloading decision process. The overall algorithm running on each device is defined in Algorithm 2. Each learning agent is able to monitor and gather statistics required to perform the prediction, as mentioned in Section IV. Once a new task arrives at the node, it asks for help from the experts, as in FPL. The experts that our FPL algorithm can employ are the two regressor algorithms. This method, however, returns only one value, p, i.e., the expert's prediction for the next time step, and this predicted value is then used to determine whether offloading the incoming task as follows. If this value exceeds a determined threshold specific for the agent i, T_i, this implies a long local task execution, and the task is consequently offloaded. Otherwise, it is kept local and enqueued for future execution.

From the described algorithm, it appears as the value of the threshold T_i is a crucial parameter. Clearly, its setting depends on the environment and the nature of tasks, but, in general, it should be defined in order to balance the two actions, i.e., offloading and local execution. Offloading means diminishing the waiting time but increasing the transmission time. Keeping the task locally implies facing the waiting time but avoiding wireless transmission. Thus, offloading should be selected only when the expected waiting time is considered "too high". The threshold is a numerical definition of the "too high" concept.

In light of this self-learning procedure, not only can our agents progressively enhance the single predictors, but they can learn the more advisable algorithm to follow given the considered

environment. Ideally, the expected behavior of this framework is that the VARMA is selected in the first place, given its ability to require a short amount of data (see Section VI). Then, when more metrics become available, the RFR can outperform the statistical model and, consequently, becoming the preferred choice of FPL.

It is known, indeed, that Random Forests produce better results on large datasets and are able to work with missing data by using estimations of them [38]. However, they pose a major challenge as they cannot extrapolate outside unseen data. On the other hand, VARMA has the ability to work well with unseen data, interpolating the given data to obtain the prediction. In conclusion – and as confirmed by our results – we can enumerate the differences as follows: classical models are simpler and more interpretable, while ML methods are more complex but more flexible. The choice of VARMA to represent classical models and RFR for ML is then motivated by the accuracy obtained in our experimental campaign (see Section VI).

VI. Results

In this section, we report the results of experiments performed to assess the effectiveness of the proposed approach. First, we analyze the accuracy of the proposed prediction methods. Then, we consider the performance of our approach comparing it to state-of-the-art solutions.

A. Experimental Setup

To evaluate the performance of the proposed task offloading strategy, we developed a Python event-driven simulator, where a networked fleet of drones has to complete incoming tasks. The edge cloud is replicated by means of a further process emulating the execution of offloaded tasks. To adopt realistic parameters for our experimental campaign, we base the choice of their default values on recent studies addressing the considered scenario, e.g., [9], [17]. In particular, new tasks are generated according to a Poisson process with an arrival rate of 0.2 Hz if not otherwise specified. In terms of computing resources, we assume the CPU capability of each server in the edge cloud and each UAV to be $f_e = 20$ GHz and $f_{i,m} = 1$ GHz, respectively. The computing workload is set as default to $C_{i,m} = 1 \times 10^9$. The channel bandwidth is set to be $W^{down} = W^{up} = 5$ Mbps, the transmitted data $D^{out}_{i,m} = 7$ MB, while $D^{in}_{i,m} = 1$ MB. The background noise power is set equal for the two technologies, as $\sigma^{down} = \sigma^{up} =$ 50 dBm. For the channel gain we have $G^{down}_{i,m} = G^{up}_{i,m} = d^{\nu}_{i,s}$, where $d_{i,s}$ is the distance between mobile agent i and access point s, and $\nu = 4$ denotes the path loss factor. By default, the distance $d_{i,s}$ is set to 10 m. Finally, we simply set the default value of the weights defined in Algorithm 1 as $\eta = 1$.

The results reported are obtained after 35 trials. This value is in line with the common practice adopted in studies dealing with simulations of distributed systems based on similar machine learning algorithms — e.g., [29], [39], where even less runs are performed. This leads us to claim how 35 trials represents a good trade-off between simulation times and proper statistical accuracy. The resulting graph's bars refer to a confidence interval of 90%. We summarize in Table II the configuration parameters utilized during the following evaluation, where the default values are reported in bold.

B. Evaluation Metrics

Throughout this section, we make use of metrics and quantities defined in Section III, such as the task completion time. Besides them, in order to study the efficacy of predictors, we use the Mean Absolute Percentage Error (MAPE), which is a simple regression error metric. For every data point, the residual is computed by taking only its absolute value so that negative and positive residuals do not cancel out. The error is then converted into a percentage, providing a clear interpretation that makes the results easily understandable. The formal equation of MAPE is given by:

$$MAPE = \frac{1}{n}\sum_{t=1}^{n} 100 \times \left|\frac{x_t - y_t}{x_t}\right|, \tag{17}$$

where x_t and y_t are the real and the predicted observations, respectively. One key advantage of MAPE is its robustness to the effects of outliers thanks to the use of the absolute value. In summary, such a value describes how far the model's predictions are off from their corresponding outputs on average.

Similarly, we compute the Mean Absolute Deviation (MAD) for the predicted values as follows:

$$MAD = \frac{1}{n}\sum_{t=1}^{n} \left|y_t - \overline{X}\right|, \tag{18}$$

where $\overline{X}$ denotes the mean of the observed values. The MAD value, as explained in [40], is another key metric during the evaluation of regressors.

Moreover, even though it is not an explicit objective of the process, we also consider the energy consumed by agent i during the execution of task m. Since the node can either compute the task locally or offload its computation to the edge cloud, we define two types of energy consumption, $E^l_{i,m}$ and $E^e_{i,m}$, for the local execution and the edge execution, respectively. In the case of local computation, we use the widely adopted model of the energy consumption per computing cycle as $\mathcal{E} = kCf^2$ [41], [42], where k is the energy coefficient depending on the chip architecture, $f_{i,m}$ is the CPU frequency, and $C_{i,m}$ specifies the workload, i.e., the amount of computation to accomplish the task in terms of numbers of cycles. According to some realistic measurements available in [43], we set the energy coefficient k as 5×10^{-11}. Moreover, in the other event of task offloading, $E^{up} = \frac{p_{i,m}T^{tra}_{i,m}}{\xi_i}, \forall i, m$, where ξ_i is the power amplifier efficiency of node i. Without loss of generality, we assume that $\xi_i = 1\ \forall i$. We also assume the energy consumption in the edge cloud is negligible since the cloud typically has enough energy to execute the offloaded tasks. Then, similar to the energy spent in transmission, we define the energy consumed during the reception phase, $E^{down} = \frac{p_{i,m}T^{rec}_{i,m}}{\xi_i}, \forall i, m$. Thus, the energy spent in offloading the task is the sum of these two communications, $E^e_{i,m} = E^{up} + E^{down}$.

In conclusion, for simplicity, we refer to the energy consumption as E, and is computed as follows:

$$\begin{aligned} E &= (1 - o_{i,m})E^l_{i,m} + o_{i,m}E^e_{i,m}, \\ &= (1 - o_{i,m})\left(k(f^l_{i,m})^2 C_{i,m}\right) \\ &\quad + o_{i,m}\left(p_{i,m}T^{tra}_{i,m} + p_{i,m}T^{rec}_{i,m}\right). \end{aligned} \tag{19}$$

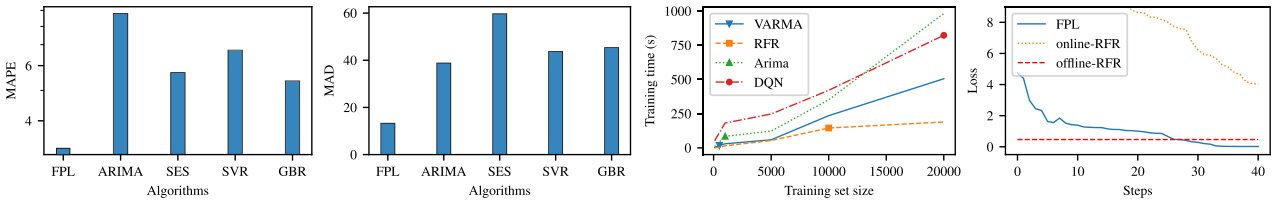

Fig. 2. (a) MAPE error and (b) mean absolute deviation (MAD) for different algorithms. (c) Training time of the two class of regressors at varying the training set sizes. Markers denote the amount of samples required for convergence. (d) Convergence time comparison, i.e., loss evolution, for FPL and RFR methods.

C. Predictor Accuracy

For this first part focusing on the accuracy of the predictors, we first offline train the considered models on a relatively small dataset consisting of 5036 samples. In particular, we apply a walk-forward validation. In such a technique, the dataset is split into train and test sets by selecting a cut point, and we select a point to split the dataset as 80% training set and 20% test set. Then, even over the test set these models are fitted for every new observation, and the training phase continues online to improve the accuracy and to fit the specific circumstances on the agent.

Fig. 2(a) shows MAPE for different predictors. Specifically, we compare against two other time series forecasting methods, autoregressive-moving average (ARIMA) and Simple Exponential Smoothing (SES), and two ML-based regressors, Support Vector Regression (SVR) and Gradient Boosting Regression (GBR). From the graph, we can observe how results validate our approach. In particular, by leveraging alternate techniques, our FPL model provides the lowest error in predicting. Notably, compared to the second-best regressor, i.e., SES algorithm, our method can halve the error.

We then compare the MAD error among the same set of predictors, and we report the results in Fig. 2(b). We can easily conclude that not only FPL can provide a smaller error, but the variance is reduced. This result is particularly important since it assures that our approach leads to fewer outliers in the prediction task. In fact, a method with high MAD suggests that when it is wrong, the error could be too high, leading to an inappropriate conclusion. On the other hand, our FPL is always close to the real value, so even though the value is not exact, the finding is likely more accurate.

We then reason about the number of samples required to the algorithms for the convergence. In this phase, we also consider an approach based on Reinforcement Learning (RL) [44], a class of machine learning known for its capability of solving sequential decision-making problems with unknown state-transition dynamics. The sequential decision-making problem is typically formulated as a Markov decision process (MDP). Hence, using a Markov process, we model the action of task offloading given the same state space we considered in our formulation. The action, then, is obtained via the DQN setting of RL, one of the most popular [45].

Fig. 2(c) shows the training time for different algorithms at increasing the number of samples in the training set. It also reports the number of samples required for these algorithms to converge. This value is reported in the graph as a marker, and is defined as the value that, when exceeded, the advantage in the error is marginal, i.e., less than 5%. It is worth noticing that RL models typically use the episode metric to define the convergence time. Thus, we convert this metric into the number of samples in order to use a uniform metric.

We can detect the different behavior of the methodologies. Time series models, i.e., VARMA and ARIMA, require a medium amount of training time but are not data hungry, and can converge even when not much data is present, around 500. On the other hand, a pure ML model, as RFR, is very fast in its training phase but requires a considerable amount of data to converge, around 10,000. As expected, while RL can lead to excellent results and can model more challenging scenarios, it takes a long time to converge and is needed for a very large training set. These results motivate our approach based on FPL, which can fit the application and network requirements by leveraging alternatively a method requiring fewer samples to converge or fast in on-online training. We can thus conclude that our model can adapt to multiple scenarios and network conditions/technologies, as well as can better face the dynamic evolution of the conditions, which evolve over time.

After having studied the diverse amount of samples required for training, as well as the time elapsed, we now study the time needed by our FPL-based algorithm to converge. To this end, we compared the convergence time of FPL to RFR, as it is the most accurate at regime. We consider two different versions for the latter: an already trained version (offline-RFR) and an RFR during its learning phase (online-RFR). Fig. 2(d) displays the loss (actual value - predicted value) of these alternatives, where the offline-RFR is constant over time since the model parameters were already fixed during training. We can notice how our approach outperforms the offline-RFR after approximately 30 steps, while the online-RFR in the first 20 steps has a too excessive loss to report in the figure. Such an online-RFR method achieves a reasonable still high loss at 30 steps, the number of steps required by our FPL to stably converge. This observation validates our hypothesis of using an FPL-based algorithm to online adapt the prediction and mix online and offline parameter settings to speed up the learning.

D. Solution Performance Analysis

To study the behavior of our solution at varying environmental conditions, we consider diverse indicators for increasing fleet sizes. In Fig. 3(a) we show the cumulative distribution function (CDF) of task completion time. We can easily observe how a more significant number of nodes leads to a reduced mean task completion time. Not only, this also reduces the number of outliers in the distribution, since the number of congested

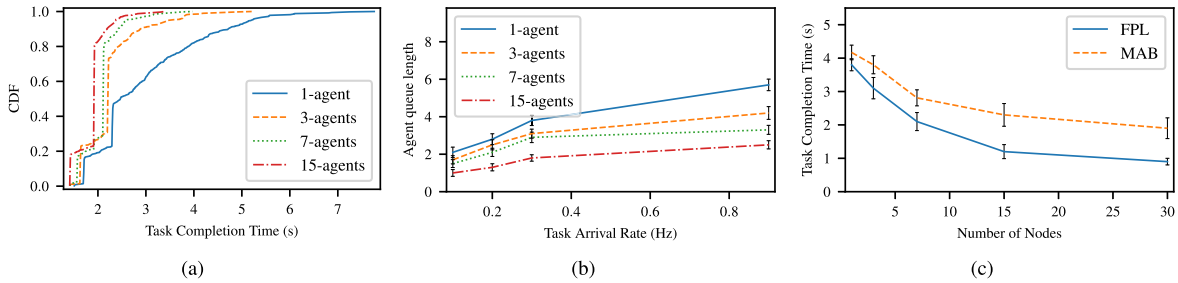

Fig. 3. (a) CDF of task completion time and (b) queue agents length at varying the task arrival rate. Both experiments consider an increasing fleet size. (c) Task completion time of our FPL-based approach compared to a more complex solution as MAB. Our FPL outperforms this alternative.

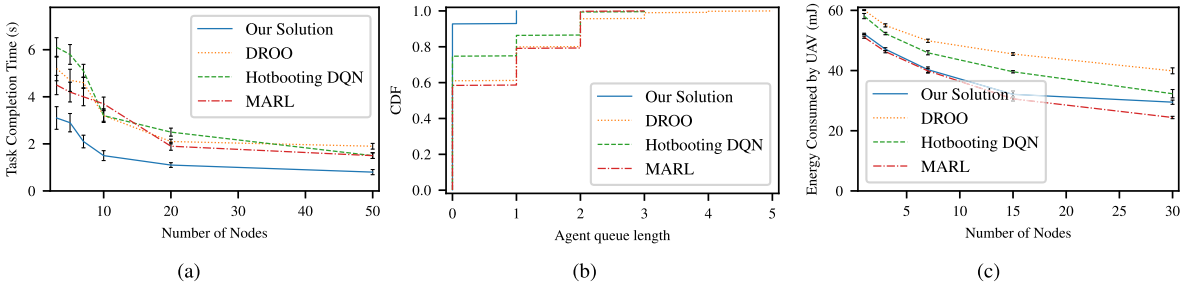

Fig. 4. (a) Task completion time, (b) CDF of queue agents length, and (c) energy consumption for various offloading solutions. Despite not as an objective of our algorithm, our solution can also limit the energy consumed by UAVs.

nodes is reduced as well. This result confirms the goodness of our model, which can be employed even for multiple IoT nodes.

Similar conclusions can be taken if examining the average queue length of the agents when increasing the task arrival rate. In particular, we can observe how, when tasks are introduced in the system at a higher rate, the growth in the queue size is logarithmic. This result suggests that our approach can efficiently handle the presence of many tasks in the system. Confirmation of this behavior is presented in Section VI-E, when our solution is compared against other methods.

We then evaluate how a more elaborated approach, as Multi-Armed Bandit (MAB), behaves when utilized in this scenario. In MAB algorithms, the set of possible actions is typically referred to as "arms". Unlike the more general class of reinforcement learning, in bandit problems, MAB only observes the outcome of a selected action for a given state. We model the arms of this approach as the two leaders of our FPL: one arm refers to the VARMA predictor, and the other arm is the SVR regressor. As shown in Fig. 3(c), our FPL-based algorithm can consistently provide a lower task completion time. Although FPL does not provide the same guarantees of MAB and RL, we can observe that the performance of our formulation resembles the results obtained using an enhanced model as in MAB.

E. Comparison With State-of-the-Art

To study the effectiveness of our solution, we compare it against three similar solutions: the DROO framework [7], which implements a deep neural network that learns the binary offloading decisions; a solution based on the multi-agent reinforcement learning framework [10], that is able to select the best radio access technology for the offloading process; a hotbooting Q-learning scheme for computation offloading [28], herein referred to as hotbooting DQN, as it uses a fast deep Q-network (DQN) model to further improve the offloading performance.

We can first consider Fig. 4(a), showing the impact of the number of nodes on the task completion time. Our solution is able to reduce such completion time compared to the other benchmark algorithms. Even when the number of nodes increase, our method outperforms the alternatives.

We then evaluate the effects of task offloading over the queue of agents, reporting in Fig. 4(b) the CDF of queue length. The queue length is considered a key metric in this scenario, as it clearly impacts the time to complete tasks, but also the computation rate and the energy spent by the node. It can be easily observed that, with our solution, we can shorten the amount of tasks waiting in the agent's queue, while other RL-based methods are more prone to overload the agent.

Additionally, we investigate the energy consumption that the solutions lead to, reporting the results in Fig. 4(c). While other benchmark algorithms consider the minimization of energy consumption in the problem formulation, our model is unaware of this aspect. Nevertheless, our solution is able to achieve comparable results with MARL that has been designed for energy efficiency purposes. Moreover, we can reduce consumption with respect to DROO and DQN. Thus we can conclude that, although our approach is blindfolded concerning power saving, it can lead to an energy-efficient method. These results confirm our hypothesis that modeling the device as a queue of tasks is a simplistic yet effective way of exploiting the edge while considering the application performance.

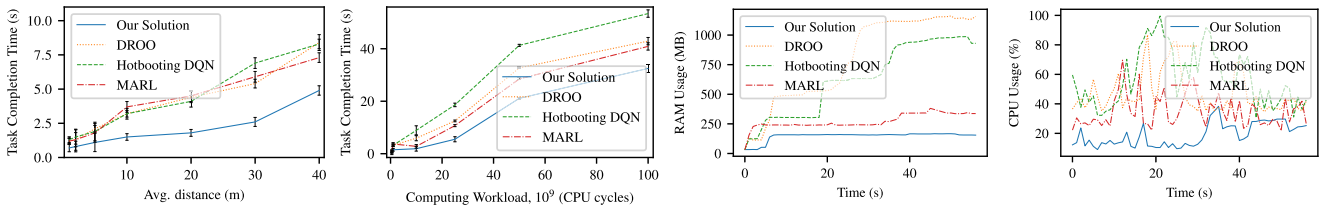

Fig. 5. (a) Time spent for the task computation at varying the node-antenna distance. (b) Task completion time for increasing average computing workload. (c) Memory resources and (d) CPU consumed during the execution of our considered algorithms.

Moreover, we consider the impact of the distance among the nodes of the system and the close edge cloud (Fig. 5(a)). While the time to complete the task is clearly increasing as the distance increases, we can also observe how this increment is attenuated if compared to other solutions. The ability of our solution to online learn the best offloading decision, as well as to improve the prediction, makes it more suitable to handle a variety of conditions, as in the case of diverse antenna locations. Other differences may arise when tasks to be performed require different execution times. After having conducted an experimental analysis to investigate this behavior, we report in Fig. 5(b) the task completion time for ours and the three considered solutions while the computing workload changes. It can be seen how our solution can adapt its prediction and significantly reduce the task completion time for each task size. In light of these and previous results, we can conclude that the adaptive property ensures that our model is able to face multiple circumstances that are challenging for other approaches.

Finally, we consider the amount of RAM and CPU required to train and execute these algorithms. As can be seen in Fig. 5(c), our implementation can drastically reduce the amount of memory consumed, leading to a significant improvement. While the deep learning approaches are hungry for RAM, the MARL model can optimize consumption. However, our regressors can further shorten the demand for memory. Considering then the CPU consumption in Fig. 5(d), similar conclusions hold. Simulating the execution of the learning processes over an Intel(R) Core(TM) i7-7500 U CPU @ 2.70 GHz, it is observable a reduction of CPU usage. In conclusion, we can consider our solution more lightweight than alternatives. This result is extremely important, especially in the UAV context, where a reduced memory footprint, along with less computation, is fundamental.

VII. Conclusion

This paper presents a learning-based solution to solve the dilemma of whether a task should be offloaded to the close edge cloud or not. Our solution lets the devices autonomously learn the offloading decisions on the basis of the current state. Such a decision exploits two classes of predictors, i.e., time series and ML regression, to predict future conditions. By doing so, the node can determine online the accuracy of these methods. Based on this value, then, the agent determines where incoming tasks should be executed. The results validate our model, evidencing how our implementation outperforms state-of-the-art solutions. In particular, despite the simplicity of our learning algorithm, its accuracy is comparable to other RL-based processes.

References

[1] Y. Zeng, R. Zhang, and T. J. Lim, "Wireless communications with unmanned aerial vehicles: Opportunities and challenges," *IEEE Commun. Mag.*, vol. 54, no. 5, pp. 36–42, May 2016.

[2] J. Xu, L. Chen, and P. Zhou, "Joint service caching and task offloading for mobile edge computing in dense networks," in *Proc. IEEE Conf. Comput. Commun.*, 2018, pp. 207–215.

[3] 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.

[4] A. V. Ventrella, F. Esposito, A. Sacco, M. Flocco, G. Marchetto, and S. Gururajan, "APRON: An architecture for adaptive task planning of Internet of Things in challenged edge networks," in *Proc. IEEE 8th Int. Conf. Cloud Netw.*, 2019, pp. 1–6.

[5] T. K. Rodrigues, J. Liu, and N. Kato, "Offloading decision for mobile multi-access edge computing in a multi-tiered 6G network," *IEEE Trans. Emerg. Topics Comput.*, to be published, doi: 10.1109/TETC.2021.3090061.

[6] A. Sacco, F. Esposito, and G. Marchetto, "Resource inference for sustainable and responsive task offloading in challenged edge networks," *IEEE Trans. Green Commun. Netw.*, vol. 5, no. 3, pp. 1114–1127, Sep. 2021.

[7] L. Huang, S. Bi, and Y.-J. A. Zhang, "Deep reinforcement learning for online computation offloading in wireless powered mobile-edge computing networks," *IEEE Trans. Mobile Comput.*, vol. 19, no. 11, pp. 2581–2593, Nov. 2020.

[8] 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, Aug. 2019.

[9] 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. 11158–11168, Nov. 2019.

[10] A. Sacco, F. Esposito, G. Marchetto, and P. Montuschi, "Sustainable task offloading in UAV networks via multi-agent reinforcement learning," *IEEE Trans. Veh. Technol.*, vol. 70, no. 5, pp. 5003–5015, May 2021.

[11] 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, Dec. 2019.

[12] 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. Veh. Technol.*, vol. 69, no. 8, pp. 8777–8791, Aug. 2020.

[13] T. K. Rodrigues, J. Liu, and N. Kato, "Application of cybertwin for offloading in mobile multi-access edge computing for 6G networks," *IEEE Internet Things J.*, vol. 8, no. 22, pp. 16231–16242, Nov. 2021.

[14] S. Yang, D. Kwon, H. Yi, Y. Cho, Y. Kwon, and Y. Paek, "Techniques to minimize state transfer costs for dynamic execution offloading in mobile cloud computing," *IEEE Trans. Mobile Comput.*, vol. 13, no. 11, pp. 2648–2660, Nov. 2014.

[15] L. Yang, J. Cao, Y. Yuan, T. Li, A. Han, and A. Chan, "A framework for partitioning and execution of data stream applications in mobile cloud computing," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 40, no. 4, pp. 23–32, 2013.

[16] 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, Dec. 2018.
[17] 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, Jan. 2019.
[18] 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.
[19] A. R. Karlin, M. S. Manasse, L. A. McGeoch, and S. Owicki, "Competitive randomized algorithms for nonuniform problems," *Algorithmica*, vol. 11, no. 6, pp. 542–571, 1994.
[20] P. Zhao, H. Tian, C. Qin, and G. Nie, "Energy-saving offloading by jointly allocating radio and computational resources for mobile edge computing," *IEEE Access*, vol. 5, pp. 11255–11268, 2017.
[21] 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, Aug. 2019.
[22] 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, Jul. 2020.
[23] 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.
[24] 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.*, vol. 8, no. 12, pp. 9763–9773, Jun. 2021.
[25] Y. Wang, K. Wang, H. Huang, T. Miyazaki, and S. Guo, "C learning in fog computing for industrial applications," *IEEE Trans. Ind. Informat.*, vol. 15, no. 2, pp. 976–986, Feb. 2019.
[26] Y. Sun, X. Guo, S. Zhou, Z. Jiang, X. Liu, and Z. Niu, "Learning-based task offloading for vehicular cloud computing systems," in *Proc. IEEE Int. Conf. Commun.*, 2018, pp. 1–7.
[27] 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.
[28] 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, Feb. 2019.
[29] D. Di Paola, M. Gaggero, A. Petitti, and L. Caviglione, "Optimal control of time instants for task replanning in robotic networks," in *Proc. Amer. Control Conf.*, 2016, pp. 1993–1998.
[30] N. K. Ure, G. Chowdhary, J. P. How, M. A. Vavrina, and J. Vian, "Health aware planning under uncertainty for UAV missions with heterogeneous teams," in *Proc. Eur. Control Conf.*, 2013, pp. 3312–3319.
[31] E. Hannan, "The identification of vector mixed autoregressive-moving average system," *Biometrika*, vol. 56, no. 1, pp. 223–225, 1969.
[32] N. Littlestone and M. K. Warmuth, "The weighted majority algorithm," *Inf. Comput.*, vol. 108, no. 2, pp. 212–261, 1994.
[33] M. Zinkevich, "Online convex programming and generalized infinitesimal gradient ascent," in *Proc. 20th Int. Conf. Mach. Learn.*, 2003, pp. 928–936.
[34] J. Hannan, "Approximation to bayes risk in repeated play," *Contributions to Theory Games*, vol. 3, pp. 97–139, 1957.
[35] A. Kalai and S. Vempala, "Efficient algorithms for online decision problems," *J. Comput. Syst. Sci.*, vol. 71, no. 3, pp. 291–307, 2005.
[36] N. Cesa-Bianchi and G. Lugosi, *Prediction, Learning, and Games*. Cambridge, U.K.:Cambridge Univ. Press, 2006.
[37] A. Cohen and T. Hazan, "Following the perturbed leader for online structured learning," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 1034–1042.
[38] A. Sacco, F. Esposito, and G. Marchetto, "RoPE: An architecture for adaptive data-driven routing prediction at the edge," *IEEE Trans. Netw. Service Manag.*, vol. 17, no. 2, pp. 986–999, Jun. 2020.
[39] S. Abbasloo, C.-Y. Yen, and H. J. Chao, "Classic meets modern: A pragmatic learning-based congestion control for the internet," in *Proc. Annu. Conf. ACM Special Int. Group Data Commun.*, 2020, pp. 632–647.
[40] G. P. Zhang, "Time series forecasting using a hybrid arima and neural network model," *Neurocomputing*, vol. 50, pp. 159–175, 2003.
[41] Y. Wen, W. Zhang, and H. Luo, "Energy-optimal mobile application execution: Taming resource-poor mobile devices with cloud clones," in *Proc. IEEE Int. Conf. Comput. Commun.*, 2012, pp. 2716–2720.
[42] X. Chen, "Decentralized computation offloading game for mobile cloud computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 4, pp. 974–983, Apr. 2015.
[43] A. P. Miettinen and J. K. Nurminen, "Energy efficiency of mobile clients in cloud computing," in *Proc. 2nd USENIX Conf. Hot Topics Cloud Comput.*, 2010, pp. 1–7.
[44] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA:MIT Press, 2018.
[45] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Process. Mag.*, vol. 34, no. 6, pp. 26–38, Nov. 2017.

Alessio Sacco (Graduate 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, such as 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 worked in the industry for a few years.

His research interests include network management, network virtualization, and distributed systems. He was the recipient of the 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 computer engineering from the Politecnico di Torino, Turin, Italy, in 2008. He is currently an Associate Professor with the Department of Control and Computer Engineering, Politecnico di Torino. 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 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 in Turin, and of HKN, the Honor Society of IEEE. He is the Editor-in-Chief of IEEE TRANSACTIONS ON EMERGING TOPICS IN COMPUTING, the 2020–2021 Chair of the IEEE TAB/ARC and the Co-Chair of the 2021 TAB/PSPB Ad Hoc Committee on Publications Strategy. Previously, he was in a number of positions, including the Editor-in-Chief of IEEE Transactions on Computers (2015–2018), the 2017–2020 IEEE Computer Society Awards Committee Chair, a Member-at-Large of IEEE PSPB (2018–2020), and as the Chair of its Strategic Planning Committee (2019–2020).