SeReMAS: Self-Resilient Mobile Autonomous Systems Through Predictive Edge Computing

Davide Callegaro†, Marco Levorato† and Francesco Restuccia*
†Donald Bren School of Information and Computer Sciences, University of California at Irvine, United States
*Department of Electrical and Computer Engineering, Northeastern University, United States
e-mail: {dcallega, levorato}@uci.edu

Abstract*—Edge computing enables mobile autonomous systems (MASs) to execute continuous streams of heavy-duty mission-critical processing tasks, such as real-time obstacle detection and navigation. However, in practical applications, erratic patterns in channel quality, network load, and edge server load can interrupt the task flow's execution, which necessarily leads to severe disruption of the system's key operations. Existing work has mostly tackled the problem with *reactive* approaches, which cannot guarantee task-level reliability. Conversely, in this paper we focus on learning-based *predictive* edge computing to achieve *self-resilient* task offloading. By conducting a preliminary experimental evaluation, we show that there is no dominant feature that can predict the edge-MAS system reliability, which calls for an ensemble and selection of *weaker* features. To tackle the complexity of the problem, we propose SeReMAS, a data-driven optimization framework. We first mathematically formulate a redundant task offloading problem (RTOP), where a MAS may connect to multiple edge servers for redundancy, and needs to select which server(s) to transmit its computing tasks in order to maximize the probability of task execution while minimizing channel and edge resource utilization. We then create a predictor based on deep reinforcement learning (DRL), which produces the optimum task assignment based on application-, network- and telemetry-based features. We prototype SeReMAS on a testbed composed by a Tarot650 quadcopter drone, mounting a PixHawk flight controller, a Jetson Nano board, and three 802.11n WiFi interfaces. We extensively evaluate SeReMAS by considering an application where one drone offloads high-resolution images for real-time analysis to three edge servers on the ground. Experimental results show that SeReMAS improves the task execution probability by 17% with respect to existing reactive-based approaches. To allow full reproducibility of results, we share the dataset and code with the research community.

I. Introduction

Mobile autonomous systems (MASs) such as self-driving cars and drones are disrupting the wireless, embedded and computing manufacturing industries, with unprecedented repercussions on agriculture, film making, surveillance and urban mobility, among others. According to a study by PwC, the current global market value for drones is estimated to be over $127 billion [1], while it is expected that North America's self-driving car market will expand at a CAGR of 50.8% with a global revenue of $49.79 billions by 2024 [2]. Thus, it is no wonder that MASs have captured the interest of academia and industry, now rushing to research and develop MASs-related devices and technologies across many different facets [3].

Characterized by the abundance of rich sensors (*e.g.*, cameras, radars, and GPS), coupled with fifth generation (5G) wireless networking and advanced mobility, MASs are unique devices that can travel between destinations with little to no human control. To achieve this complex endeavor, MASs necessarily require the continuous, real-time execution of *streams* of computation-expensive tasks. For example, self-driving cars have to continuously build detailed 3D maps of the surrounding areas, and use them to categorize different navigation features such as blockages, intersections, driveways, or fire hydrants [4]. Moreover, autonomous drones are always at risk of sudden and significant drift due to adverse weather conditions, loss of power and/or GPS connectivity. Therefore, the seamless fusion of multimedia sensor data for real-time path planning is quintessential for the drone's survival.

Motivation and Problem Setting. Offloading the stream of tasks generated by the MASs to edge servers can extend battery lifetime and reduce task round-trip time delay [5]. However, strong assumptions such as perennial stability of high-capacity communication links do not apply in the highly-dynamic context of MASs, where wireless links are bound to exhibit erratic behavior even in very simple scenarios. This key problem is further exacerbated in larger MASs and urban deployments, where parameters such as server and network load may induce more system instability.

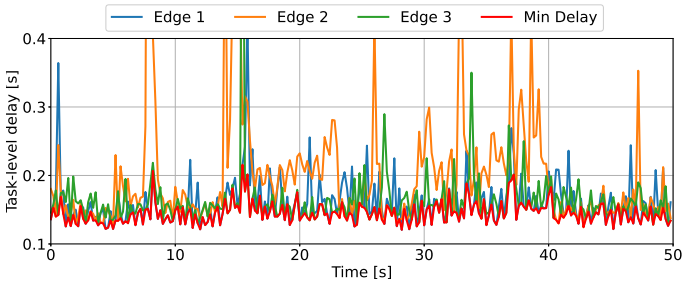


Fig. 1: Example of task level delay from a flying drone to 3 edge servers, transmitted over WiFi 802.11n in a 50s interval.

In this paper, we tackle the challenging problem of providing *task-level* performance guarantees to a stream of computing tasks generated by an airborne MAS. Specifically, we impose a bound on the maximum time between data acquisition and the completion of the corresponding analysis task. We remark how a task-level perspective is necessary in this class of systems, where temporally local degradation of task delay can severely harm control loops in MASs. Figure 1 shows the temporal pattern of the end-to-end application delay (the different curves are different edge servers) obtained through

our experimental drone testbed described in Section V-A. We can observe that the delay exhibits significantly time-varying patterns, with a standard deviation 0.14 and a peak-to-peak difference reaching 0.43, which is 241% of the average value of 0.178s. We note that the experimental setting is in line of sight (LOS), and that more convoluted propagation environments would just aggravate this problem. A bound on the average delay would not guarantee that the task-level delay will be below a certain threshold for *each* of the tasks belonging to the task stream, which is key to guarantee correct functionality of stream-oriented edge-based MASs.

Our vision is simple: *the seamless usage of edge resources by MASs necessarily requires techniques able to mitigate the impairments and erratic temporal patterns induced by the surrounding communication and computing ecosystems and the physics of the system itself*. Existing work – discussed in detail in Section VI – has tackled the issue of MASs reliability in a piecemeal and often highly abstract fashion, by focusing on static optimization of either mobile device's trajectory [6–9] or communication resources [10–12]. In Sec. V-A, we show that edge selection methodologies based on channel quality would fail, and we conclude that new task offloading strategies are needed to stabilize task completion delay in MASs.

To address this challenging problem, we developed SeReMAS – Self-Resilient Mobile Autonomous Systems – a framework whose core is a dynamic task replication mechanism, where individual tasks are replicated and sent over multiple channel/edge server resources. The key intuition is that the task delay experienced by the MAS will be the minimum delay of each replica. Thus, the larger the number of channel/edge couples, the greater the probability that one task will satisfy the delay requirement, which however also implies increased resource usage. The objective of SeReMAS is to minimize resource usage under the constraint that the probability that the task-level delay bound will be met.

Our Approach. To drive our design, we implemented a testbed composed by an airborne MAS and multiple ground servers (Section II). Specifically, we extracted a rich dataset from the system (Section V-A), whose analysis demonstrates a lack of variables strongly correlated with the delay (Section II-A). We show that the received signal strength indicator (RSSI), one of the key variables used to control connectivity and offloading, has limited influence on the delay. The dataset illustrates how in real-world MAS systems the delay pattern is the result of a wide variety of complex cross-variable interactions at various temporal scales. Importantly, influential variables are outside the network layers, and include *physical* variables such as orientation, acceleration and tilt.

Based on this considerations, SeReMAS embeds a *predictive* core based on Deep Reinforcement Learning (DRL) to determine a compact set of computing pipelines dynamically assigned task-by-task based on the perceived state of the system. Fig. 2 depicts the high-level schematics of SeReMAS. The key intuition is that the selected set of channel/computing resources will influence future decision making, which DRL is able to

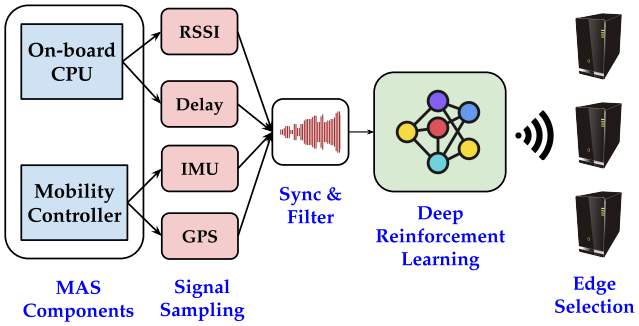


Fig. 2: Our Architecture for Task Offloading in MASs.

capture. Some of the features – e.g., application and most network-related features – become available only if a resource is used. For instance, if a channel/edge server pipeline is not selected for a task, then the corresponding delay is not observed, which motivates the adoption of a DRL-based approach. By including future rewards in action selection and taking as input unprocessed features such as RSSI, end-to-end delay, inertial measurement unit (IMU) and global positioning system (GPS) coordinates, *the DRL algorithm will implicitly embed the impact of current computing pipelines selection on the efficacy of future decisions, as well as real-world phenomena that can be hardly modeled through explicit mathematical terms*.

Novel Contributions

• We design SeReMAS, a framework for the dynamic control of task offloading in MASs with extreme temporal variations (Section III). SeReMAS is based on a preliminary experimental analysis (Section II-A), which indicates that there is no dominant feature, including obvious features such as channel quality, and that prediction necessitates an ensemble of *weaker* features. We first mathematically formulate (Section III-B) a redundant task offloading problem (RTOP). Then, we create predictors that can help managing the resource usage/performance trade-off. Specifically, we propose a myopic predictor as baseline (Section III-C) and a DRL-based approach, which operates on a set of features from application, network and device-level components (Section III-D). **To the best of our knowledge, SeReMAS is the first framework addressing the problem of redundant task offloading in MAS with a data-driven approach which efficacy is verified in a real-world testbed and with replicable dataset-based experiments.**

• We prototype SeReMAS on a drone-based experimental testbed (Section IV). The platform embeds a module for the real-time analysis of features, including the flight controller, tied to internal data routing control. As part of our prototype, we design a strategy to make the state representation compact (Section IV-B), and thus lower the complexity of the DRL agent, using an iterative feature selection procedure. We consider a real-time image analysis application through state-of-the-art edge-assisted object detection algorithms where a drone periodically acquires from onboard sensors data whose analysis is offloaded to edge servers on the ground (Section V-A). We let the drone perform task offloading through multiple WiFi

interfaces, and collect a total of 140 minutes of flying. We pledge to release the dataset to the community.

• Through experiments, we show how different subsets of features appear dominant at different time-scales (Section V-B). We also show in Section V-C how the DRL approach improves by 17% the task execution probability with respect to a reactive approach [13], thanks to the ability to manage state uncertainty in the action selection problem, measured in terms of probability of meeting a delay requirement per amount of resource used, with respect to a myopic controller based on a one-shot selection of the next set of edge servers to be used.

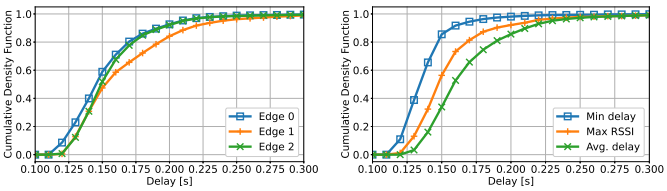
II. Preliminary Experiments

In our setting, a MAS is connected to N edge servers $\mathrm{es}_1, \mathrm{es}_2, \ldots, \mathrm{es}_N$ through separate wireless channels. The device generates a sequence of tasks $t_1, t_2, t_3, \ldots$ with fixed inter-arrival time equal to T seconds. A task is described as a chunk of data to be processed with a predetermined analysis algorithm to produce an output. We assume that tasks are homogeneous, meaning that the amount of data associated with any task and the analysis algorithm are fixed. Let us define $\delta_n(t_i)$ as the capture-to-output delay of task t_i executed as edge server es_n, defined as the time from the generation of the task to the availability of its output at the edge server. The delay $\delta_n(t_i)$ is the composition of two delays: the transmission delay $\delta_n^{\mathrm{comm}}(t_i)$ and the computing delay $\delta_n^{\mathrm{comp}}(t_i)$. In real-world settings, both components are highly stochastic, and depend on a number of latent variable, parameters as well as states of protocols at various layers of the stack.

A. Preliminary Analysis

We motivate our study by analyzing the data obtained from real-world experiments. We consider an experimental setting, described in significant detail in Section V-A, where a drone is offloading image processing tasks to three edge servers. Fig. 1 shows a section of the temporal pattern of the task-level delay δ_t at the three edge servers. We observe that the delay signals alternate low-delay ($150 - 175$ms) sections with spikes and higher delay sections. While some mild correlation between the delay signals is present, the minimum of the three signals provides the needed stability to the delay. Fig. 3.a shows the Cumulative Density Function (CDF) of the task-level delay δ_t for the three edge servers in our experiments. Note that in our scenario the task execution delay δ^{comp} is nearly deterministic. We remark that all the edge servers are within coverage, and that all the links are in Line of Sight (LoS). Most delays are in the range 120ms to 250ms, with about 40% of the delays below 135-145ms.

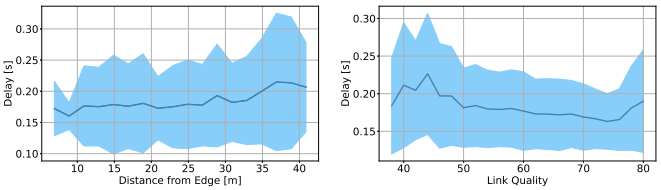
Fig. 3.b shows the distribution of the minimum delay $\delta_{\min}$ with respect to the cdf of the average delay and the delay associated with the edge server with the maximum channel quality index (RSSI). We observe that there is a noticeable difference between the minimum delay and the delay offered by the edge server with the best channel quality. Therefore, even a perfect SNR-based handover would fail to provide optimal performance in this context. This effect is the result of the convoluted interdependencies between protocol variables at the various layers and the physical and hardware properties of the system at multiple time scales.



(a) (b)

Fig. 3: Cumulative density function of delay (a) for each edge server and (b) selecting the minimum delay, or the one with maximum RSSI, or the average of the available delays.

We remark this important aspect by plotting in Fig. 4 the (delay, RSSI) and (delay, distance) mean and one standard deviation of the delay as a function of the two other variables. We can see the lack of a strong correlation between the delay and both RSSI and distance and emphasize again how experimental results emphasize effects and interactions that are rarely captured in simulations and models.



(a) (b)

Fig. 4: Distribution of task level delay as a function of distance from each of the edges and the RSSI.

III. The SeReMAS Framework

The results illustrated in the previous section emphasize the need for new techniques boosting the reliability of edge offloading for extreme real-time applications. In this section, we present SeReMAS, a data-driven framework addressing the reliability of task offloading in MAS. We first present an overview of the main system blocks and functionalities in Section III-A. Then, we formalize the learning-based redundant task offloading control problem in Section III-B.

A. SeReMAS: A Walkthrough

SeReMAS [14][1] enables the data-driven control of task offloading from the MAS to the edge servers. The architecture of SeReMAS is shown in Fig. 5, where we emphasize the modules performing mobility control of the MAS (yellow) and control of task offloading (blue), and the modules – multiplexing and filter – handling the communication between the section of the platform at the MAS to the section at the various edge servers.

[1] Code is left hidden in accordance with the double blind review process.

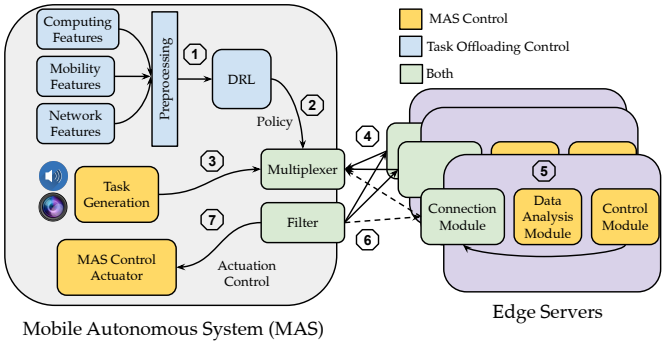


Fig. 5: SeReMAS system architecture: two different control cycles intersect at the communication modules, where the DRL agent's policy is applied by means of task replication.

We now provide a walk-through of the main operations performed by SeReMAS, following the steps indicated in Fig. 5. First, the framework takes computing features (e.g., CPU, GPU, and RAM utilization), mobility features (e.g., accelerometer, gyroscope, GPS coordinates, etc), and network features (e.g., TCP state, RSSI) and applies pre-processing (step 1) to construct the input to a DRL model (see Section III-B for details). The extracted features and the composition of the state space are described in Section IV-B. Then, the DRL state is given as input to the DRL algorithm, which outputs ϕ, the set of edge servers to be used as task executors (step 2).

Tasks are generated (step 3) according to the current MAS needs (e.g., multimedia classification), and handled by module called *multiplexer* (step 4) which handles task replication across multiple edge servers. Specifically, the multiplexer is responsible for replicating and forwarding the tasks to the edge servers, and is directly controlled by the output ϕ of the DRL algorithm. The tasks are sent to the edges specified by ϕ, which are then executed (step 5). The knowledge produced by the task execution can be used to drive control decisions on the MAS. For example, in our prototype we use the task result to control the mobility of the MAS, as explained in Section IV. The related control messages generated by the edge server(s) are sent back to the MAS, and processed by the filter module (step 6), which eliminates replicated messages when more than one edge server is selected to avoid the re-execution of flight commands. Finally, the control messages are fed to the control actuator (step 7), which takes care of implementing the control action, if needed (e.g., flight control).

B. Redundant Task Offloading Problem (RTOP)

As part of the SeReMAS framework, we investigate the problem of redundant task offloading to replicate tasks and send them over multiple channel-edge server pipelines for increased reliability, which we call RTOP. This problem will drive our DRL design. We define the capture-to-output delay as the minimum of the delays associated with the task replicas:

$$\delta_{t_i}(\phi_i) = \min\{\delta_n(t_i) : n{\in}\phi_i\}, \tag{1}$$

where $\phi_i{\subseteq}\{1,\dots,N\}$ is the subset of edge servers to which a replica of task t_i is sent. Then, we define a controller whose objective is to determine the sequence of edge servers $\boldsymbol{\phi}^*{=}[\phi^*_{t_1},\phi^*_{t_2},\dots]$ solving the following optimization problem:

$$\arg\min_{\boldsymbol{\phi}} \quad E_i\left[|\phi_i|\right] \tag{2}$$

$$\text{s.t.} \quad E_i\left[I\left(\delta_{\min}(t_i) > \delta^*\right)|\phi_i\right] < \Delta, \tag{3}$$

where $I(\cdot)$ is the indicator function and expectation is computed over the task sequence. This formulation is different than imposing a constraint on the average delay, i.e., $E_i\left[\delta_{\min}(t_i)|\phi_i\right] < \delta^*$. The latter formulation would allow a possibly large number of delays above δ^*, while our formulation is equivalent to

$$\arg\min_{\boldsymbol{\phi}} \quad E\left[|\phi_t|\right] \tag{4}$$

$$\text{s.t.} \quad P\left(\delta_{\min}(t) > \delta^*|\phi_t\right) < \Delta. \tag{5}$$

Thus, we impose a constraint on the probability that the task completion time is above a threshold δ^* while striving to minimize resource usage.

Intuitively, the larger the number of edge servers selected, the larger the probability that the minimum of the delays is below the threshold. However, the inevitable limitations on channel access and maximum edge server load leads to a *task-level selection* problem, where the number and members of the chosen set is informed by the uncertainty regarding future delays and their expected values. In real-world settings, the resolution of the RTOP defined above necessitates the consideration of complex inter-variable and temporal interdependencies. For this reason, we resort to data-driven solutions methodologies decomposing the problem into sequences of local problems.

C. Myopic-based Baseline for RTOP

First, we formulate a myopic predictive solution to address the RTOP. We introduce the notion of state of the system $s_i = \{s_{i,n}\}_{n=1,\dots,N}$, where $s_{i,n}$ is the feature matrix

$$s_{i,n} = \begin{bmatrix} \psi_{1,i-L+1,n} & \cdots & \psi_{1,i-1,n} & \psi_{1,i,n} \\ \psi_{2,i-L+1,n} & \cdots & \psi_{2,i-1,n} & \psi_{2,i,n} \\ \vdots & \vdots & \cdots & \vdots \\ \psi_{F,i-L+1,n} & \cdots & \psi_{F,i-1,n} & \psi_{F,i,n} \end{bmatrix}, \tag{6}$$

of $F{\times}L$ features, and $\psi_{f,j,n}$ is $f-th$ feature referring to task j and computing pipeline n. We describe the specific features and dataset in the Section IV-B. We train a probabilistic predictor as the function $p_{i+1,n} = \sigma(s_{i,n})$, where

$$p_{i+1,n} = P\left(\delta_n(t_{i+1}) > \delta^*\right). \tag{7}$$

We find the set ϕ_{i+1} with minimum cardinality such that

$$P\left(\delta_n(t_{i+1}) > \delta^*|\phi_{i+1}\right) < \Delta, \tag{8}$$

where the left-hand term is computed as

$$1{-}\prod_{n\in\phi_{i+1}} (1{-}p_{i+1,n}). \tag{9}$$

When more than one set with the same cardinality satisfies the constraint, then the one with the smaller probability is chosen. We note that stronger predictors $\sigma(\cdot)$ may lead to a reduced resource usage, as they would lead to reduced uncertainty in the class of the next delay (above and below threshold), and thus would allow the controller to bet on fewer remote computing pipelines. For example, let us assume that at least one of the pipelines has a next delay below threshold: an accurate and confident predictor returning probability 1 would allow the selection of only one edge server.

We extend the predictor to larger temporal windows to evaluate the predictive power of features blocks. We define

$$p^{W,y}_{i+1,n} = \sigma(s_{i,n}) \tag{10}$$

where

$$p^{W,y}_{i+1,n} = P\left(\sum_{\ell=0}^{W-1} I(\delta_n(t_{i+\ell}) > \delta^*) \geqslant y\right), \tag{11}$$

that is $p^{W,y}_{i+1,n}$ is the probability that at least y tasks will be completed with delay larger than δ^* in a window of W future tasks. We build a binary classifier from $\sigma(\cdot)$ by setting

$$p^{W,y}_{i+1,n} \underset{C_0}{\overset{C_1}{\gtrless}} \frac{1}{2} \tag{12}$$

D. Deep Q-Learning Approach for RTOP

The formulation above produces suboptimal control sequences. Thus, we adopt a Deep Q-Learning formulation to resolve the optimization problem. This formulation implicitly accounts for the impact of current decisions on the distribution of future states (and thus on the accuracy of control). In this case, the predictive function is defined to return the Q-values based on the state, that is,

$$Q(s_{i+1}, \phi_{i+1}) = \sigma_{DRL}(s_i), \tag{13}$$

where

$$\begin{aligned} Q(s_i, \phi_i) = E_{s_{i+1}|s_i,\phi_i}\left[E_{r_{i+1}|s_{i+1},\phi_i,s_i}\left[r_{i+1}|s_{i+1}, \phi_i, s_i|\right]\right] \\ + \gamma \max_{\phi'} E_{s_{i+1}|s_i,\phi_i}\left[Q(s_{i+1}, \phi')\right]. \end{aligned} \tag{14}$$

The cost variable c_i includes weighted penalties for the delay being above threshold and the cardinality of the selected set, that is

$$c_i = \lambda\ c_i^{\text{delay}} + (1-\lambda)\ c_i^{set}, \tag{15}$$

with

$$c_i^{\text{delay}} = I(\delta_{\min}(t_i) > \delta^*) S(\alpha^{\text{delay}} \delta_{\min} - \kappa^{\text{delay}}) \tag{16}$$

and

$$c_i^{\text{set}} = \alpha^{\text{set}} |\phi_i| - \kappa^{\text{set}}, \tag{17}$$

where α^{delay}, α^{set}, κ^{delay} and κ^{set} are normalization and offset parameters. $S(x)=1/(1+e^{-x})$ is the sigmoid function, here used to generate a smooth delay cost function which is 0 until δ^* and then progressively penalizing higher delay without overpenalizing tasks with poor channel conditions. Figure 6 shows the training procedure for our DRL-based approach.

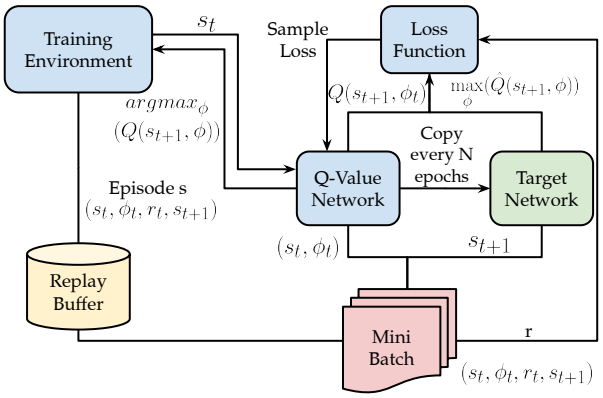


Fig. 6: Training architecture using Double Deep Q-Learning.

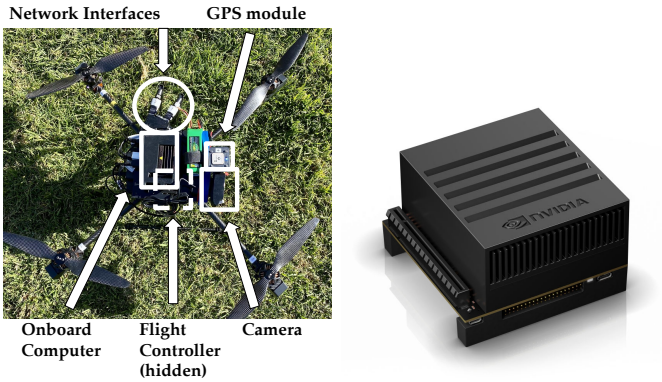
We remark that the recursive formulation of the Q-values embeds the distribution of future states and costs given the current policy. The Q-values guide the selection of the actions according to the rule:

$$\phi_{i+1} = \begin{cases} \text{argmin}_{\phi_i} Q(s_i, \phi_i) & \text{with prob. } 1-\epsilon_t \\ \mathcal{U}(\mathcal{P}(\phi) \backslash \varnothing) & \text{with prob. } \epsilon_t \end{cases} \tag{18}$$

where the best action (that is, subset of servers) is selected as the one maximizing the future reward with probability $1-\epsilon$, and selected uniformly at random with probability ϵ. This is commonly known as a *ϵ-greedy* strategy, it is often used in practical problems to balance exploration/exploitation in DRL problems.

IV. SeReMAS Prototype

We first describe the platform experimental components in Section IV-A, and then describe our feature selection process in Section IV-B. Finally, we explain how we implemented the SeReMAS predictors for the RTOP, both myopic and DRL, in Section IV-C.



(a) (b)

Fig. 7: (a) Drone prototype; (b) NVidia Jetson Xavier acting as edge server.

A. Platform Components

Figure 7 shows our experimental setup. Specifically, we use a Tarot650 quadcopter mounting a PixHawk flight controller.

We connect Telem2 port on the PixHawk to a serial interface on a NVidia Jetson Nano board with 4GB of RAM. We use three NVidia Jetson Xavier development boards, operating in performance mode with 8 core ARM 64-bit processor, 32GB of main memory, 512-core Volta GPU. We use three IEEE 802.11n WiFi cards to interconnect the drone to the edge servers. These boards act as access points on different channels in the 2.4GHz WiFi spectrum.

B. DRL State Space and Feature Selection

We discuss how we create the input state for the DRL algorithm. We consider features at the application, network stack and device level as follows:

- **Application and Onboard Computer:** We track relevant application variables such as past capture-to-control delays, number of samples in the intermediate buffers, and selected actions. These will include real-time statistics relative to power consumption and resource allocation of CPU, GPU, and RAM.
- **Telemetry and Position:** We use MAVLink [15] protocol messages to register a listener to the flight controller. The onboard computer receives monitoring statistics from the Inertial Measurement Unit (IMU), Global Positioning System (GPS) and the power consumption of the vehicle. Specifically, we include the edge servers' position, by including the distance on the three axes, using the (North, East, Down) reference frame, as well as a polar representation (Distance, Azimuth, Elevation) centered in the reference edge server and oriented with its antennas' direction. Moreover, we add the relative heading by computing the orientation of the drone with respect to the position of the edge server, as well as the distance from each server using the Harvesine formula. Furthermore, we consider the $L2$-norm of multi-dimensional vectors (such as accelerometer and gyroscope data) and compute speed with respect to absolute reference frame and edge servers. All the features are synchronized at 5Hz.
- **Network:** We select relevant parameters such as TCP window and retransmissions, RSSI, and modulation/coding scheme (MCS) of the IEEE 802.11n protocol. We do so separately for each network interface available, so to isolate features relative to each edge server.

The details of the features are available in [14].[2]

Feature Selection. We use feature importance methods such as Logistic Regression, Support Vector Machines and Random Forest as implemented in [16] and selected Logistic Regression with L1 regularizer due to the bias that Random Trees have towards features with high support's cardinality and the hybrid nature of the features, which include continuous and categorical variables. We then used a recursive algorithm, where at each iteration we train a predictor and discard the least influential features. We reduce the initial pool of 360 features to 73, maximizing accuracy on the validation set. Table I shows the normalized feature relevance predicting the number of high-delay tasks in a $1\ s$ window.

Feature	Normalized Correlation
Round Trip Time average	1
Transmission timeout	-0.83
Packets Received	-0.80
Channel Level	-0.48
Inclination (magnitude)	-0.17
Position w.r.t Edge	0.16
Altitude	0.16
Last Sent	-0.15
Heading	0.13
Speed	0.08
Congestion Window	0.08
Gyroscope	0.07

TABLE I: Normalized feature relevance to a linear model predicting the number of high-delay tasks in a $1\ s$ window.

Interestingly, while all available past delays are selected in the prediction (with $L = 3$ in Eq. 6), acceleration and inclination features are selected with a lag of 0.6s indicating a longer range dependency with the delay. Other relevant features include gyroscope and the increment of TCP fast retransmissions, failures, RSSI, and retries. The complete trend within the window is selected for these features. The selection shows how both vehicle and network parameters are relevant to characterize the state of the system and its future behavior, but their influence is expressed at different time scales.

C. Myopic Predictor and DRL Implementation

We provide the details of the myopic and DRL controllers.

Myopic Predictor - To implement the predictor $p^{W,y}_{i+1,n} = \sigma(s_{i,n})$ we train a series of dense DNNs (with two hidden layers at $[150, 50]$ nodes) using the Adam optimizer and trained for 100 epochs, with softmax output), which returns the probability that the next delay will belong to the predicted class.

Deep Q-Learning Agent - Naive implementations of Deep Q-Learning use one DNN function. However as demonstrated in [17], this approach may cause instability during training if the Q-values presents sudden changes. Due to the erratic behavior of the system, we consider, we then take a Double Deep Q-Learning (DDQL) approach to build our DRL agent. In DDQL, two separate Deep Neural Networks (DNN) are used. Referring to Eq. (14), one network approximates is trained to approximate $Q(\cdot) = Q(s_i, \phi_i)$, and the other one to approximate the future Q-value term in the expectation, that is $\hat{Q}(\cdot) = Q(s_{i+1}, \phi')$

Fig. 6 illustrates the DDQN architecture and the training procedure. We use a fully connected DNN, with $[200, 100, 50]$ hidden nodes, ReLu activation, Huber Loss. During training, we apply backpropagation to $Q(\cdot)$ over the epochs $e = 1, ..., N$. We periodically copy DNN's parameters so that $\hat{Q} \leftarrow Q$, as a means to reduce noise in during training. Note that we still choose the best action to learn on ϕ using the most updated $Q(\cdot)$, and in fact the decoupling between action selection and q-value function evaluation further stabilizes learning. We use a replay buffer during training, where the experience in the form of $(s_i, \phi_i, r_i, s_{i+1})$ are stored and sampled randomly to avoid forgetting, which may occur if only the most recent experiences are used [18].

[2] Omitted to comply with double blind submission policy.

V. Experimental Results

We first present the experimental setting in Section V-A, then the prediction performance in Section V-B, and the task offloading results in Section V-C.

A. Experimental setting

We consider a testbed illustrated in Fig. 8, which is composed of an airborne drone and N=3 ground edge servers in LOS. We consider an object tracking application where the MAS uses a camera to follow a predefined object at a certain distance. Specifically, the MAS captures images that are analyzed to extract the bounding box of the closest object of a certain class (*e.g.*, a person).

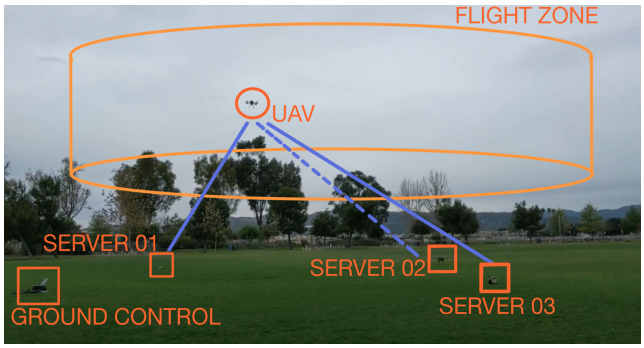


Fig. 8: Schematic representation of the system setting: three ground edge servers, connected to the drone. Not all connections are continuously actively used (unused is dashed).

The controller then steers the vehicle in the appropriate direction to (*i*) center the bounding box in the field of vision and (*ii*) obtain a bounding box of a predefined size by controlling the distance with respect to the object. In our testbed, the drone generates a regular stream of images to be analyzed using object detection. Specifically, the drone emits 15 images of size 19.5 kB per second. SSD-MobileNet-v2 model is used to analyze the images. In our measurements, the NVidia Jetson Xavier board takes 10 ms to execute the algorithm. Note that the onboard NVIDIA Jetson Nano takes 87 ms to complete the model, however, power expenditure shifts from 1.6 W to 4.2 W when the GPU is processing the images, that is, 11% of the power needed to fly.

To acquire a dataset for a wide-spectrum of flight parameters, we set the drone on a semi-random flight pattern around the edge server. The pattern is defined by assigning uniformly distributed GPS way-points to the drone in a cylinder of radius equal to 30m centered on the edge server constellation and confining the altitude in the $[5, 15]m$ range. The maximum speed is randomly chosen for every new GPS waypoint between $[1, 4]m/s$. A new waypoint is set as soon as the drone reaches 3 meters from the current one, to obtain a smooth flight as similar as possible to a real application. In drone applications, the outcome of the object detection analysis is promptly needed to take control action and adjust the trajectory. While the action taken after the image analysis is beyond the scope of the current manuscript, we mention target tracking [19], object avoidance [20] as possible applications.

B. Prediction Performance

All results are based on an experimental dataset[3] collected using the randomized flight patterns described in Section V-A. We first evaluate the prediction performance of the myopic predictors $p^{W,y}_{i+1,n} = \sigma(s_i)$ and associated binary classifier. In other words, the predictor determines whether at least half of the delay in the future window is below a given threshold, which we set to δ^*=175ms. We use the Area Under the Curve (AUC), integral of the ROC with respect to false positives, as performance metric, commonly used to evaluate algorithms predicting an imbalanced target.

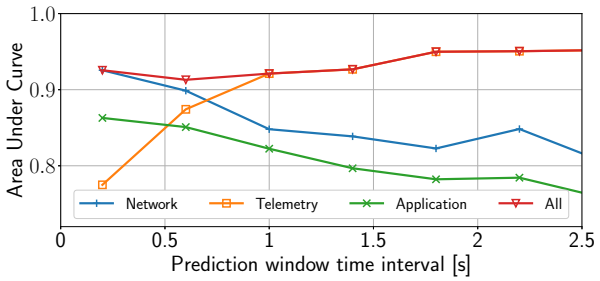


Fig. 9: Performance of future delay classification for different sets of features. Length of the prediction window is expressed in seconds.

Fig. 9 shows the performance of the predictor trained on different feature blocks as a function of the window W (where we set $y = W/2$). The results highlight how semantic differences across subsets of features influence their predictive power in the short and long term. When the prediction window is small, most of the predictive power lies in networking features, which capture short-term correlations between high delay events. However, network variables struggle to capture longer-term trends, which are, instead predicted by telemetry variables. Indeed, the latter directly influence the distribution of fine-grain network events.

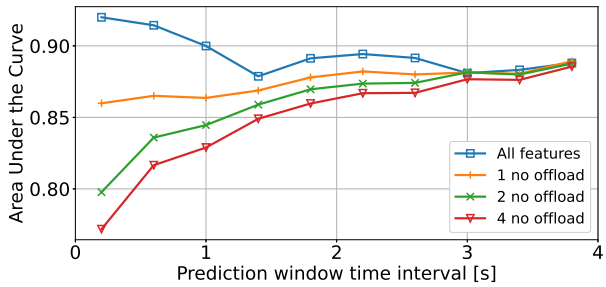


Fig. 10: Performance in future delay classification in presence of partial information for recent time slots. Length of the prediction window is expressed in seconds.

As noticed earlier, part of the network information is available only when offloading to a particular edge server. We now analyze how prediction performance is affected when

[3] Omitted to comply with the double blind policy.

several recent samples lack such information for one server. Fig. 10 shows how the lack of full state information (which is available only if the edge server is used) in recent samples (last one, last two, etc.) affects the ability of the myopic classifier to accurately predict future pipeline performance as a function of the prediction window W expressed in seconds. Missing information in one or few recent input samples, has a noticeable effect on classification in the short term, as the AUC reduces by 5% for one sample and 10% for just two samples. On the other hand, as expected, the influence of recent samples fades out when predicting further points in the future. As the decisions of the DRL agent embed the future performance beyond the next delay sample, they also consider the availability of information in future decision instances.

C. Redundant Offloading

Fig. 11 shows the performance of the myopic and DRL selectors in terms of delay (percentage below threshold) and resource usage (average number of edge servers used). The different points for the myopic approach are obtained by varying the parameter Δ, i.e., the bound on the probability that the delay is below threshold.

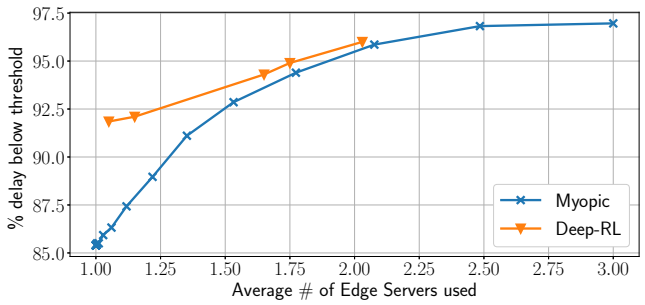


Fig. 11: Delay performance and resource utilization trend of the myopic and DRL-based selector.

The DRL approach, as described in Section III-B, generates different points in the plot for different values of the weight λ in the cost function, where a larger λ favors low delay over resource usage. For comparison, we include a selector which uses all the available edge servers for all the tasks, and a selector which uses the edge server with the best channel quality index. When using all the three edge servers all the time, the myopic selector achieves maximum performance ($\sim 97\%$), whereas when using only one edge server, it achieves $\sim 85.5\%$. We note that a selector that chooses the edge server with the best channel quality achieves 75% of tasks with delays below threshold. Thus, *predictive control greatly improves performance compared to traditional options, even when idealized to task-level granularity without connection delay*. As we make the bound on Δ more tight, the myopic approach uses more and more resources.

We observe that using two edge servers, the myopic controller already achieves a performance roughly 2% worse than the three edge server option, demonstrating that prediction can reduce resource usage. However, when using a small amount of resources, the myopic controller's effectiveness sharply decreases. Conversely, the DRL is capable of effectively select small sets of computing pipelines while preserving delay performance. Using 1.1 edge servers on average, the DRL approach achieves $\sim 92\%$, that is, 7% more than the myopic approach. We explain this trend by observing that the DRL agent optimizes the information available to make future decisions, thus maximizing the overall prediction accuracy when resources are scarce and selection needs to be precise.

To further illustrate the behavior of the proposed approach, we show in Figure 12 a time series of delays and decisions (selected edge servers) of the DRL-based approach for two different λ (0.1 and 0.2) used in Eq. 15. We can see that the DRL agent can stabilize delay, where a larger use of resources leads to the avoidance of more delay peaks. We note how the DRL agent rotates the edge servers periodically to harvest information for more informed future decisions.

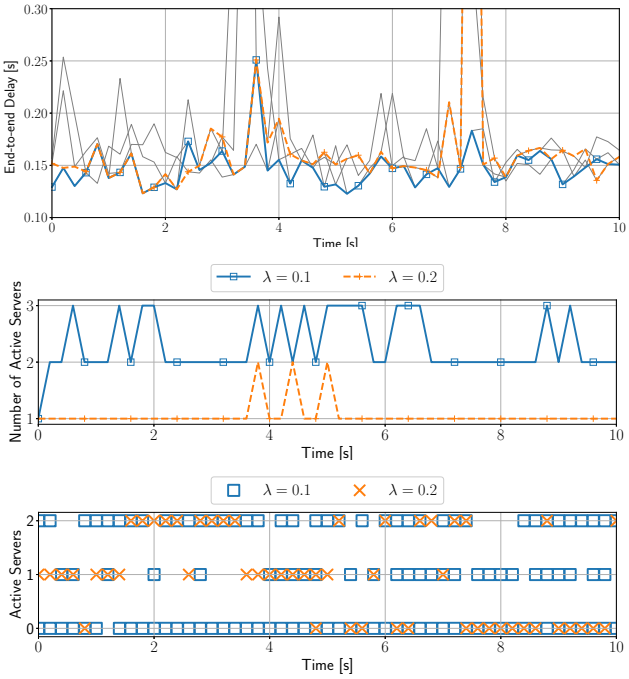


Fig. 12: DRL agent improving delay by using task replication. We plot in grey the traces of the non-selected delays.

VI. Related Work

Edge computing can significantly improve reliability and performance in mobile applications [5]. Different frameworks perform a multi-layer optimization to exploit the full potential of edge computing [6, 21]. To fully exploit the edge servers, the user equipment needs to periodically make a decision on whether to process tasks locally, or to offload. In the latter case, there might be multiple technologies or networks available, e.g. [22], and a link must be chosen for each transmission. Convex optimization has been proven to be ineffective due to the presence of complex factors such as user's mobility [3]. Classic approaches are shown to perform better with coarser

granularity settings and when considerable prior knowledge is available. For example, in [23] the authors develop an online multi-decision making scheme, solving a task offloading problem while jointly optimizing caching, communication and computation resources in the Internet of Vehicles, exploiting the proximity of users to roadside units.

Fast-changing mobile networks usually emply data-driven approaches, using Markov Decision Processes (MDP), Q-Learning or DRL. MDPs have shown a great compromise between the flexibility of learning, and the data efficiency of a model-based solution [24, 25]. However, MDP-based solutions present state space that is too large, and require vast amounts of data to find the correct transitions for each state-action pair during training. Finally, they are very memory intensive both in training and running time. For these reasons, DRL approaches have been proposed. Cao et. al [3] present a general framework for intelligent offloading in multi-access edge computing composed by observation tier, analysis tier, prediction tier and policy tier. In this paper, we consider a much more complicated problem where the trade-off is beyond power efficiency and link performance. Recently, researchers have worked towards simulation environments for drones, for example, OpenUAV [26] and FlyNetSim [27]. However, neither of the two environments can capture the interactions between mobility and application delay that are key in this paper. This underlines the importance of sharing our dataset with the community to further allow research that can interpret, explain and exploit further these interactions.

VII. Conclusions

This paper has proposed SeReMAS, a data-driven optimization framework for predictive task offloading in edge-assisted mobile autonomous systems (MASs). We have formulated a redundant task offloading problem (RTOP) and created a predictor based on deep reinforcement learning (DRL), which produces the optimum task assignment based on application-, network- and telemetry-based features. We have prototyped SeReMAS on a real-world testbed, and extensively evaluated SeReMAS by considering an application where one drone offloads high-resolution images for real-time analysis to three edge servers on the ground. Experimental results show that SeReMAS improves the task execution probability by 17% with respect to existing reactive-based approaches.

Acknowledgment

This work was supported by the NSF grant IIS-1724331 and MLWiNS-2003237.

References

[1] M. Mazur, A. Wisniewski, and J. McMillan, "Clarity from above: PwC global report on the commercial applications of drone technology," *Warsaw: Drone Powered Solutions, PriceWater house Coopers*, 2016.

[2] Market Watch, "North America Self-driving Car Market - Global Industry Analysis, Size, Share, Growth, Trends, and Forecast," https://tinyurl.com/w64u9jwn, 2020.

[3] B. Cao, L. Zhang, Y. Li, D. Feng, and W. Cao, "Intelligent Offloading in Multi-access Edge Computing: A State-of-the-art Review and Framework," *IEEE Communications Magazine*, vol. 57, no. 3, pp. 56–62, 2019.

[4] R. Dass, "5 Key Challenges faced by Self-driving cars," https://medium.com/@ritidass29/5-key-challenges-faced-by-self-driving-cars-ed04e969301e, 2018.

[5] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. ACM, 2012, pp. 13–16.

[6] F. Zhou, Y. Wu, R. Q. Hu, and Y. Qian, "Computation Rate Maximization in UAV-Enabled Wireless-Powered Mobile-Edge Computing Systems," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 9, pp. 1927–1941, 2018.

[7] F. Cheng, S. Zhang, Z. Li, Y. Chen, N. Zhao, F. R. Yu, and V. C. Leung, "UAV Trajectory Optimization for Data Offloading at the Edge of Multiple Cells," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 7, pp. 6732–6736, 2018.

[8] J. Zhang, L. Zhou, Q. Tang, E. C.-H. Ngai, X. Hu, H. Zhao, and J. Wei, "Stochastic Computation Offloading and Trajectory Scheduling for UAV-assisted Mobile Edge Computing," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 3688–3699, 2018.

[9] Q. Hu, Y. Cai, G. Yu, Z. Qin, M. Zhao, and G. Y. Li, "Joint Offloading and Trajectory Design for UAV-enabled Mobile Edge Computing Systems," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 1879–1892, 2018.

[10] Z. Yang, C. Pan, K. Wang, and M. Shikh-Bahaei, "Energy Efficient Resource Allocation in UAV-enabled Mobile Edge Computing Networks," *IEEE Transactions on Wireless Communications*, vol. 18, no. 9, pp. 4576–4589, 2019.

[11] T. Zhang, Y. Xu, J. Loo, D. Yang, and L. Xiao, "Joint Computation and Communication Design for UAV-Assisted Mobile Edge Computing in IoT," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 8, pp. 5505–5516, 2020.

[12] L. Bertizzolo, S. D'Oro, L. Ferranti, L. Bonati, E. Demirors, Z. Guan, T. Melodia, and S. Pudlewski, "SwarmControl: An Automated Distributed Control Framework for Self-Optimizing Drone Networks," in *Proc. of IEEE Conference on Computer Communications (INFOCOM)*, 2020, pp. 1768–1777.

[13] A. Awang, S. Agarwal, and M. Drieberg, "Data aggregation using rssi for multihop wireless sensor networks: Energy and delay performance," in *2013 IEEE 11th Malaysia International Conference on Communications (MICC)*, 2013, pp. 422–426.

[14] "Entry hidden to preserve anonymity and it will be replaced in camera ready version." 2021.

[15] A. Koubâa, A. Allouch, M. Alajlan, Y. Javed, A. Belghith, and M. Khalgui, "Micro air vehicle link (mavlink) in a nutshell: A survey," *IEEE Access*, vol. 7, pp. 87 658–87 680, 2019.

[16] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in python," *the Journal of machine Learning research*, vol. 12, pp. 2825–2830, 2011.

[17] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," *CoRR*, vol. abs/1509.06461, 2015. [Online]. Available: http://arxiv.org/abs/1509.06461

[18] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska *et al.*, "Overcoming catastrophic forgetting in neural networks," *Proceedings of the national academy of sciences*, vol. 114, no. 13, pp. 3521–3526, 2017.

[19] M. Mueller, N. Smith, and B. Ghanem, "A benchmark and simulator for uav tracking," in *European conference on computer vision*. Springer, 2016, pp. 445–461.

[20] D. Wang, W. Li, X. Liu, N. Li, and C. Zhang, "Uav environmental perception and autonomous obstacle avoidance: A deep learning and depth camera combined solution," *Computers and Electronics in Agriculture*, vol. 175, p. 105523, 2020.

[21] N. Cheng, F. Lyu, W. Quan, C. Zhou, H. He, W. Shi, and X. Shen, "Space/Aerial-Assisted Computing Offloading for IoT Applications: A Learning-Based Approach," *IEEE J. on Sel. Areas in Communications*, vol. 37, no. 5, pp. 1117–1129, 2019.

[22] B. Li, Z. Fei, and Y. Zhang, "UAV Communications for 5G and Beyond: Recent Advances and Future Trends," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 2241–2263, 2018.

[23] Z. Ning, K. Zhang, X. Wang, L. Guo, X. Hu, J. Huang, B. Hu, and R. Y. K. Kwok, "Intelligent edge computing in internet of vehicles: A joint computation offloading and caching solution," *IEEE Transactions on Intelligent Transportation Systems*, pp. 1–14, 2020.

[24] D. Callegaro and M. Levorato, "Optimal Computation Offloading in

Edge-Assisted UAV Systems," in *2018 IEEE Global Communications. Conference (GLOBECOM)*, 2018, pp. 1–6.

[25] B. Zhang, G. Zhang, W. Sun, and K. Yang, "Task offloading with power control for mobile edge computing using reinforcement learning-based markov decision process," *Mobile Information Systems*, vol. 2020, 2020.

[26] M. Schmittle, A. Lukina, L. Vacek, J. Das, C. P. Buskirk, S. Rees, J. Sztipanovits, R. Grosu, and V. Kumar, "Openuav: A uav testbed for the cps and robotics community," in *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*. IEEE, 2018, pp. 130–139.

[27] S. Baidya, Z. Shaikh, and M. Levorato, "Flynetsim: An open source synchronized uav network simulator based on ns-3 and ardupilot," in *Proceedings of the 21st ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, 2018, pp. 37–45.