A Two-Stage Cooperative Reinforcement Learning Scheme for Energy-Aware Computational Offloading

Marios Avgeris†, Meriem Mechennef*, Aris Leivadeas*, Ioannis Lambadaris†
*Department of Software and IT Engineering, École de technologie supérieure (ÉTS), Montréal, Canada
†Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada
mariosavgeris@cunet.carleton.ca, meriem.mechennef.1@ens.etsmtl.ca, aris.leivadeas@etsmtl.ca, ioannis@sce.carleton.ca

***Abstract*—In the 5G/6G era of networking, computational offloading, i.e., the act of transferring resource-intensive computational tasks to separate external devices in the network proximity, constitutes a paradigm shift for mobile task execution on Edge Computing infrastructures. However, in order to provide firm Quality of Service (QoS) assurances for all the involved users, meticulous planning of the offloading decisions should be made, which potentially involves inter-site task transferring. In this paper, we consider a multi-user, multi-site Multi-Access Edge Computing (MEC) infrastructure, where mobile devices (MDs) can offload their tasks to the available edge sites (ESs). Our goal is to minimize end-to-end delay and energy consumption, which constitute the sum cost of the considered system, and comply with the MDs' application requirements. To this end, we introduce a two-stage Reinforcement Learning (RL)-based mechanism, where the MDs-to-ES task offloading and the ES-to-ES task transferring decisions are iteratively optimized. The proper operation, effectiveness and efficiency of our proposed offloading mechanism is assessed under various evaluation scenarios.**

***Index Terms*—Computational Offloading, Edge Computing, Reinforcement Learning, Deep Q-Networks, Latency minimization, Energy consumption minimization.**

I. Introduction

The rapid advancement of Internet of Things (IoT) and 5G/6G networks has led to the emergence of a plethora of diverse time-sensitive end energy-consuming mobile applications, such as Augmented Reality (AR) and Virtual Reality (VR). However, the limited computational capabilities and battery lifetime of mobile devices (MDs) comes forth as a challenge in executing such applications. To mitigate this, during the last few years Multi-Access Edge Computing (MEC) and its variations (e.g., Fog Computing) has been proposed as a viable solution which has attracted significant attention. The key idea behind this concept is to liberate MDs by offloading the computationally intensive workloads to powerful servers in the network proximity, thus reducing the energy consumption of the devices [1]. Compared with cloud-based task execution, MEC can overcome the drawbacks of high transmission delay, therefore substantially achieving the millisecond-scale latency required in said applications [2].

However, computational offloading without proper planning can potentially result in various disturbing phenomena during the transmission and processing of the tasks, including unexpected increases in propagation and processing delay, as well as downtime [3]. In that case, the trade-off between on-device execution and offloading at the local Edge Site (ES) should be investigated. At the same time, potentially transferring offloaded tasks for processing from local to remote ESs, to alleviate the overloaded sites by utilizing the underloaded ones, should be evaluated. Overall, carefully determining the offloading strategy in a multi-site edge infrastructure, for achieving load balancing while minimizing processing delay and energy consumption is an imminent challenge.

A. Related Work & Motivation

The problem of computational offloading falls into the knapsack resource allocation category, which is NP-hard [1], thus approximate solutions based on Machine Learning, Game Theory and Heuristics have been proposed in the literature. Authors in [4], consider a MEC-enabled cellular system, in which multiple mobile users offload their computational tasks via wireless channels to one MEC server. A single-stage Q-learning and a Deep Q-Network (DQN)-based scheme are used to jointly optimize the offloading decisions and the computational resource allocation, with the objective of minimizing the sum of delay and energy consumption. In [5], the mutual Device to Device (D2D) cooperation problem among users with heterogeneous demands is investigated in a MEC environment. Here, a game theoretic, multi-round cooperation matching algorithm is proposed, where users with different demands can share their idle local resources in order to minimize their processing delay. On the other hand, Yin et al. in [6] benchmark different greedy heuristics for task offloading from multiple devices to multi-core edge servers, with the objective of minimizing the energy consumption under time constraints.

To further retain low energy consumption and end-to-end delays, intra-infrastructure load balancing after task offloading is being proposed. Thus, recently, many works in the pertinent literature started exploring the cooperation between the different edge sites. In [2], the authors introduce a Markov Random Field based mechanism for the distribution of the excess

workload between the different edge sites, as a subsequent stage to the device-to-edge task offloading. This technique is shown to achieve energy optimization at the infrastructure, while respecting QoS requirements. Xu et al. [7] include additionally the cloud tier in their proposed system; here they devise a two-stage game theoretic scheme to reach the goal of optimal offloading on a cooperative user device-edge-cloud environment. The existence of a Nash equilibrium is proven, that minimizes energy consumption under delay constraints. A similar three-tier cooperative computational offloading scheme is proposed in [8] as well, this time utilizing a joint iterative algorithm based on the Lagrangian dual decomposition to minimize latency under energy consumption constraints.

B. Contributions & Outline

In this work, our goal is to further investigate the energy consumption and processing delay minimization process during computational offloading in a cooperative MEC environment, through a novel, two-stage Reinforcement Learning (RL) mechanism. The contribution of this paper is threefold:

1) We consider a MEC infrastructure consisting of multiple interconnected nodes ("multi-site"), where MDs are able to offload computationally intensive tasks to reduce their energy consumption. To enable a distributed and autonomous decision making, an iterative RL-based mechanism is introduced, where the MDs act as Stochastic Learning Automata (SLA). In every iteration of this process, each MD independently selects whether to offload at the attached ES or execute the task on the device (MD-to-ES offloading), aiming at minimizing a cost which consists of the task processing time and the MD's energy consumption (first stage).
2) This decentralized offloading can potentially overload some ESs, thus resulting in higher than anticipated processing delays for the MDs that selected to offload their tasks to them. To resolve this, we integrate an ES-to-ES cooperative offloading mechanism which aims to balance the workload in the infrastructure, by proactively transferring these tasks to the underloaded sites. An offline-trained DQN is utilized to make these decisions at the end of each iteration of the first stage and the updated processing delays are fed back to the MDs for them to reconsider their offloading decisions (second stage).
3) We combine the two stages to a multi-round cooperative computational offloading mechanism, which iteratively optimizes the decisions of the MDs and the ESs and obtains the stable convergence of the optimization problem. Detailed numerical results, obtained via simulation, evaluate and demonstrate the effectiveness and efficiency of the proposed work in terms of processing delay and energy consumption minimization.

The rest of this paper is organized as follows. Section II presents the system model and introduces the concepts of RL in computational offloading, as they are adopted in this work. In Section III the task offloading optimization problem is formulated and solved using the SLA and the DQN algorithms in succession. Section IV presents the performance evaluation of our proposed framework and Section V concludes the paper.

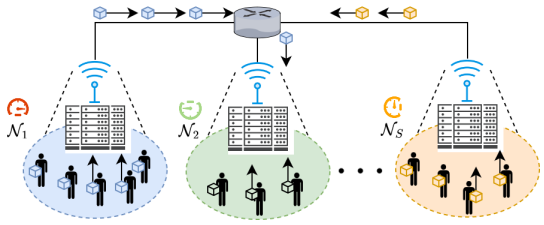


Fig. 1: System Overview.

II. System Model

We denote the set of the interconnected ESs as $\mathcal{S} = \{1, 2, ..., S\}$ where each ES consists of a small datacenter connected to a wireless Access Point (AP) to provide offloading services for the resource-constrained MDs of its coverage area, as illustrated in Figure 1. The available resources of each ES $s \in \mathcal{S}$ are given by the vector $\{F_s, W_s\}$, where F_s stands for the available computational resources and W_s for the available bandwidth respectively. The set of MDs covered by each ES s is denoted as $\mathcal{N}_s = \{1, 2, ..., N_s\}$ and they are assumed to be quasi-static for the examined offloading period, thus no MD mobility between different ESs is considered. An MD $n_s \in \mathcal{N}_s$ is characterized by the vector $\{f_{n_s}, p_{n_s}\}$ denoting its computing capabilities and uplink transmission power respectively. Additionally, each MD n_s has one application task for execution; depending on the type of application executed by n_s, each task is characterized by the vector $\{d_{n_s}, c_{n_s}, \tau_{n_s}, e_{n_s}\}$, where d_{n_s} specifies the the amount of input data to be processed, c_{n_s} represents the workload, i.e., the total number of CPU cycles required, τ_{n_s} denotes the maximum tolerable end-to-end delay and e_{n_s} the maximum tolerable energy consumption for the task. A task can either be executed on the MD ("on-device execution") or offloaded at the ES of coverage ("remote execution"). For the sake of simplicity, for one offloading period, we assume a one-to-one relationship between a device and its task, thus for the rest of the paper, these terms will be used interchangeably.

A. Computation Models

1) On-device Execution: Given that f_{n_s} denotes the computing capabilities (i.e., CPU cycles per second) of MD n_s, the on-device execution time is $t^{lc}_{n_s} = c_{n_s}/f_{n_s}$ and the corresponding energy consumption for the MD $E^{lc}_{n_s} = \kappa c_{n_s}$, where κ is the consumed energy per CPU cycle in joules.

2) Remote Execution: In the case where task n_s is offloaded to the attached ES s, its end-to-end delay, denoted by $t^{of}_{n_s}$, comprises of two parts: i) the uplink transmission time, $t^{tr}_{n_s}$ and ii) the execution time at the MEC server, $t^{ex}_{n_s}$. Following [4], we assume that the task's output size is much smaller than the input size and that the downlink rate from an ES to an MD is high enough, to allow for neglecting the transmission time

and energy consumption for delivering the computed results. Additionally, we assume that the wireless bandwidth of ES s, W_s, is equally allocated to the MDs that choose to offload to it. Then, with r_s^{of} being the number of offloaded tasks to s, the bandwidth assigned to MD n_s to upload its task input data to s is $w_{n_s} = W_s/r_s^{of}$ and based on that, the transmission time can be calculated by $t_{n_s}^{tr} = d_{n_s}/w_{n_s}$. The corresponding energy consumption for MD n during the transmission is then $E_{n_s}^{of} = p_{n_s} t_{n_s}^{tr}$. Regarding the offloading processing part, given again that the available resources of ES s, F_s, are equally allocated to the MDs that offload to it, the computing resources that site s allocates to task n_s can be calculated as F_s/r_s^{of} and subsequently the computation execution time is given by $t_{n_s}^{ex} = (c_{n_s} r_s^{of})/F_s$. According to the above, the total end-to-end delay experienced by MD n_s in the case of remote execution becomes $t_{n_s}^{of} = t_{n_s}^{tr} + t_{n_s}^{ex}$.

B. Problem Formulation and Analysis

The computing resources at the edge are limited to micro datacenters consisting of only few servers [2], thus overloading an ES with offloaded requests is not unusual. Balancing the offloaded workload by transferring tasks from the overloaded to the underloaded ESs for execution, in order to minimize QoS violations for the MDs, is a prominent way for dealing with this challenge. Effectively, in our setting we identify and explore two task-offloading opportunities: i) MD-to-ES, i.e., the typical computational offloading between the user device and the edge site and subsequently ii) ES-to-ES, which concerns the cooperation between the different sites of the infrastructure, to enhance the effectiveness of i).

To realize this synergy, we first introduce the binary variable $a_{n_s} \in \{0, 1\}$ which corresponds to the execution mode for task n_s; we have $a_{n_s} = 0$ for on-device execution and $a_{n_s} = 1$ for offloading to the attached ES. Consequently, we define an offloading decision vector $\mathcal{A}_s = \{a_1, a_2, ..., a_{N_s}\}$ to account for all the MD devices connected to ES s. Next, we introduce the binary variable $k_{n_s}^{s'} \in \{0, 1\}$ to signify the transferring of the offloaded task n_s from ES s to s' ($k_{n_s}^{s'} = 1$), with $s \neq s' \in \mathcal{S}$, as well as the transferring decision vector $K_{n_s} = \{k_{n_s}^1, k_{n_s}^2, ..., k_{n_s}^S\}$, $\forall n_s \in \mathcal{N}_s$. This allows for formulating the transferring decision matrix for the tasks of each ES s as $\mathcal{K}_s = \{K_1, K_2, ..., K_{N_s}\}^{S \times N_s}$, $\forall s \in \mathcal{S}$.

Before formulating the problem, we have to redefine the remote execution delay calculation $t_{n_s}^{of}$ to take into consideration the additional delay introduced by ES-to-ES task transferring. First, let us replace r_s^{of} with $\sum_{n_s=1}^{\mathcal{N}_s} a_{n_s}$ for w_{n_s} calculation in $t_{n_s}^{tr}$ and with $\sum_{n_s=1}^{\mathcal{N}_s} a_{n_s} + \sum_{s=1}^{\mathcal{S}} \sum_{n_{s'}=1}^{\mathcal{N}_{s'}} k_{n_{s'}}^{s}$ in $t_{n_s}^{ex}$ calculation respectively. Assuming that the propagation delay for the ES-to-ES communication, $t_{n_s,s'}$ is proportional to the distance (number of hops) between the edge sites s and s', the remote execution delay becomes $t_{n_s}^{of} = t_{n_s}^{tr} + \sum_{s' \in S}^{s' \neq s} [(1 - k_{n_s}^{s'}) t_{n_s}^{ex} + k_{n_s}^{s'} (t_{n_s,s'}^{pr} + t_{n_{s'}}^{ex})]$. We then formulate the joint task offloading problem for MEC infrastructures as an optimization problem. Our objective is to minimize the weighted sum cost in terms of the completion time and the energy consumption for all the MDs. Under the constraint of maximum tolerable delay and energy consumption, the problem can be formulated as follows:

$$\min_{\mathcal{A}_s, \mathcal{K}_s} \quad \sum_{s \in \mathcal{S}} \sum_{n_s \in \mathcal{N}_s} [(1 - a_{n_s})(\beta_1 t_{n_s}^{lc} + \beta_2 E_{n_s}^{lc}) + a_{n_s}(\beta_1 t_{n_s}^{of} + \beta_2 E_{n_s}^{of})] \tag{1a}$$

$$\text{s.t.} \quad a_{n_s} \in \{0, 1\}, \forall n_s \in \mathcal{N}_s, \forall s \in \mathcal{S}, \tag{1b}$$

$$k_{n_s}^{s'} \in \{0, 1\}, \forall n_s \in \mathcal{N}_s, \forall s, s' \in \mathcal{S}, \tag{1c}$$

$$\sum_{s' \in S}^{s' \neq s} k_{n_s}^{s'} \leq 1, \forall n_s \in \mathcal{N}_s, \forall s \in \mathcal{S}, \tag{1d}$$

$$(1 - a_{n_s}) t_{n_s}^{lc} + a_{n_s} t_{n_s}^{of} \leq \tau_{n_s}, \forall n_s \in \mathcal{N}_s, \tag{1e}$$

$$(1 - a_{n_s}) E_{n_s}^{lc} + a_{n_s} E_{n_s}^{of} \leq e_{n_s}, \forall n_s \in \mathcal{N}_s, \tag{1f}$$

where the weights $\beta_1, \beta_2 \geq 0, \beta_1 + \beta_2 = 1$, are selected based on the device configuration and the application needs. Constraint (1b) represents the computation offloading decision, while (1c) the task transferring between sites. Constraint (1d) ensures that a task is transferred to at most one server, while (1e) guarantees that the task completion time should not exceed the maximum tolerable delay τ_{n_s}, either when executed locally or remotely. Finally, constraint (1f) ensures that the task energy threshold is respected, with e_{n_s} being the maximum tolerable energy consumption.

Remark. *Since task offloading decision set $\mathcal{A}_s$ is composed of binary variables, both the feasible set and the objective function of Problem (1) are not convex, making it challenging to solve the problem. Fortunately, as demonstrated in the literature [4] this kind of NP hard problems can be easily solved effectively by applying reinforcement learning methods rather than conventional optimization methods.*

III. Energy-aware Computational Offloading Mechanism

Leveraging the power of Reinforcement Learning, a two-stage computational offloading mechanism is proposed to simultaneously minimize the delay and energy consumption at MD layer, while achieving a load balancing at the distributed edge infrastructure. In the first stage, the offloading decision problem is solved using a decentralized approach based on Stochastic Learning Automata (SLA). For the second stage, we initially propose a value-iteration based RL approach utilizing Q-Learning, to balance the offloaded workload at the edge and minimize the processing delay for the offloaded tasks. Later, to overcome the curse of dimensionality, we propose a DQN method which combines deep learning and RL to solve the same problem. These two stages are combined to a multi-round cooperative computational offloading mechanism which iteratively optimizes the offloading and transferring decisions until converging to a stable solution.

A. First Stage: MD-to-ES Task Offloading

Stochastic Learning Automata (SLA) is an adaptive, on-line decision-making unit that improves its performance by learning how to choose the optimal action from a finite set of allowed actions through repeated interactions with a random

environment [9]. We assume that each MD $n_s \in N_s$ acts as an agent and that each offloading decision $a_{n_s} \in \{0,1\}$ corresponds to the SLA action. Then, each agent's chosen action in iteration i is based on a probability distribution $p_{n_s,a_{n_s}} \in \mathbb{P}(\{0,1\})$ kept over the action-set and in each iteration; $p_{n_s,a_{n_s}}(i)$ is the probability that an agent n_s select action a_{n_s} in iteration i, and $\mathbb{P}(\{0,1\})$ is the set of probability distributions over the available action set. Initially, all action probabilities would be equal and hence the action is randomly chosen and the probabilities are updated in every iteration. Based on the problem formulation in (1a), we define the reward of each MD n_s in the i-th iteration as:

$$R_{n_s}^{(i)} = \beta_1 \cdot (1 - \frac{1}{1 + e^{-Z_1[(1-a_{n_s}^{(i)})t_{n_s}^{lc} + a_{n_s}^{(i)} t_{n_s}^{of} - \tau_{n_s}]}}) + \beta_2 \cdot (1 - \frac{1}{1 + e^{-Z_2[(1-a_{n_s}^{(i)})E_{n_s}^{lc} + a_{n_s}^{(i)} E_{n_s}^{of} - e_{n_s}]}}), \in [0,1], \quad (2)$$

where $Z_1, Z_2 \in \mathbb{R}^+$ are gain coefficients which help bring the respective terms of Eq. (2) close to 1 when the total delay is less than τ_{n_s} and the energy consumption less than e_{n_s}, and close to 0 otherwise. In this way, we embody constraints (1e) and (1f) into the reward function. The update rule of the SLA is based on the idea that if an action is selected by the agent n_s in iteration i, and the reward value $R_{n_s}^{(i)}$ received is high, then the probability of choosing this action in the next iteration of the learning procedure increases, with regards to the magnitude of the perceived reward [10]. The commonly used update rule in the research literature is the *linear reward-inaction* (LRI) defined as follows: $p_{n_s,a_{n_s}}^{(i+1)} = p_{n_s,a_{n_s}}^{(i)} + b \cdot R_{n_s}^{(i)} \cdot (1 - p_{n_s,a_{n_s}}^{(i)})$ when $a_{n_s}^{(i+1)} = a_{n_s}^{(i)}$ and $p_{n_s,a_{n_s}}^{(i+1)} = p_{n_s,a_{n_s}}^{(i)} - b \cdot R_{n_s}^{(i)} \cdot p_{n_s,a_{n_s}}^{(i)}$ otherwise. Furthermore, the learning rate parameter, $0 < b < 1$, controls the convergence of this stage. The system converges to a stable solution when at least one state probability is close to 1, for each n_s.

B. Second Stage: ES-to-ES Task Transferring

As the offloading decisions of the first stage are made in a distributed fashion, chances are that an ES might become overloaded with tasks, which can potentially hinder the processing times. In this direction, exploring a cooperative solution among the ESs, in the form of task transferring, is vital towards satisfying the QoS of the users. This step is performed right before calculating the reward $R_{n_s}^{(i)}$ in Eq. (2) and specifically produces the remote execution delay $t_{n_s}^{of}$. To solve this problem, we first employ a Q-learning (QL)-based algorithm (Algorithm 1), with the following elements:

- **State:** we define as state a vector that contains the available computational capacity in each ES, after considering the MDs' offloading decisions a_{n_s} of the first stage; $\sigma = \{F_s - \sum_{n_s=1}^{N_s} a_{n_s} c_{n_s} | s \in \mathcal{S}\}^{1 \times S}$. A state is terminal if it contains only non-negative values, $\sum_{s \in \mathcal{S}}^{\sigma_s < 0} \sigma_s = 0$.
- **Action:** as an action we use the task transferring decision matrix introduced in Section II, allowing, however, only one task transferring in the infrastructure per action; $\alpha = \{\mathcal{K}_s | s \in \mathcal{S}, \sum_{s \in S} \sum_{n_s \in \mathcal{N}_s} \sum_{s' \in S}^{s' \neq s} k_{n_s}^{s'} \leq 1\}^{S \times (S \times N_s)}$.
- **Reward:** as we opt for driving our edge infrastructure towards a balanced workload distribution, the first term of the reward in this stage is the difference between the current (σ) and the next (σ') state's sum of negative values, i.e., sites where the offloaded workload is greater than the available capacity. The second term, penalizes the transferring of tasks to remote ESs, an action which increases the additional propagation delay:

$$\mathcal{R}_{\sigma,\sigma',\alpha} = \delta_1 (\sum_{s \in \mathcal{S}}^{\sigma'_s < 0} \sigma'_s - \sum_{s \in \mathcal{S}}^{\sigma_s < 0} \sigma_s) + \delta_2 (\sum_{s \in S} \sum_{n_s \in \mathcal{N}_s} \sum_{s' \in S}^{s' \neq s} \alpha_{n_s,s'} t_{n_s,s'}^{pr})^{-2}, \quad (3)$$

where $\delta_1, \delta_2 \in \mathbb{R}^+$, are properly selected weights that balance the contribution of the two terms in the reward.

Algorithm 1: QL-based Task Transferring Training

```
1  Initialize with zeros: Q(σ, α).
2  for each episode do
3      Choose a random infrastructure state σ.
4      while Σ_{s∈S}^{σ_s<0} σ_s < 0 do
5          Select a task transferring action α.
6          Execute α, produce σ' and collect R_{σ,σ',α}.
7          Q(σ,α) ← Q(σ,α) + ζ(R_{σ,σ',α} +
             γ max_{α'} Q(σ',α') − Q(σ,α))
8          σ ← σ'
9      end
10 end
```

Algorithm 1, which is executed only once and offline, produces the matrix $Q(\sigma, \alpha)$ which contains the expectation of the long-term reward (calculated through the Bellman equation, line 7) for each infrastructure state and transferring decision, after being trained on numerous state-action pairs; ζ is the learning rate that satisfies $0 < \zeta < 1$, while $0 < \gamma < 1$ is the discount factor, used to model the uncertainty in the future actions.

As the number of ESs and MDs becomes larger, training Algorithm 1 on a sufficient number of episodes becomes a tedious and bulky task, as the possible infrastructure states grow exponentially in size. To overcome this, we propose the use of a Deep Neural Network to estimate $Q(\sigma, \alpha)$, which constitutes the basic idea behind Deep Q Networks (DQN) and is briefly presented in Algorithm 2. Notice that we make use of experience replay to improve the sample efficiency and stability of training, while we adopt the ϵ-greedy policy for the task transferring action selection, since it gives us a better balance between exploration and exploitation. To update the weights of the Q network, we use the Stochastic Gradient Descent algorithm (SGD, line 20).

C. Two Stage Cooperative MEC Computational Offloading

To sum up, the proposed cooperative MEC offloading mechanism operates in two steps per iteration, as illustrated

Algorithm 2: DQN-based Task Transferring Training

Initialize network Q and target network $\hat{Q}$ randomly.
Initialize experience replay memory $\mathcal{D}$.
while *not converged* **do**
  // Sampling Phase
  $\epsilon \leftarrow$ set new epsilon with ϵ-decay.
  Select a task transferring α using ϵ-greedy policy.
  Execute α, produce σ' and collect $\mathcal{R}_{\sigma,\sigma',\alpha}$.
  $done \leftarrow \sum_{s\in\mathcal{S}}^{\sigma_s<0} \sigma_s == 0$
  Store transition $(\sigma, \sigma', \alpha, \mathcal{R}, done)$ in $\mathcal{D}$.
  if *enough experiences in* $\mathcal{D}$ **then**
    // Learning Phase
    Sample minibatch of M transitions from $\mathcal{D}$.
    for *each* $(\sigma_m, \sigma'_m, \alpha_m, \mathcal{R}_m, done_m) \in M$ **do**
      if $done_m$ **then**
        $y_m \leftarrow \mathcal{R}_m$
      end
      else
        $y_m \leftarrow \mathcal{R}_m + \gamma max_{\alpha'} \hat{Q}(\sigma'_m, \alpha')$
      end
    end
    Loss $\mathcal{L} = \frac{1}{M}\sum_{m=0}^{M-1}(Q(\sigma_m, \alpha_m) - y_m)^2$
    Update Q using SGD by minimizing $\mathcal{L}$.
    Every $\mathcal{C}$ steps, copy weights from Q to $\hat{Q}$.
  end
end

Algorithm 3: Two-stage Cooperative MEC Computational Offloading

// Offline
Train the Q network using Algorithm 2.
// First Stage (Online)
Initialize the offloading decision probabilities.
$i \leftarrow 0$
while *not converged* **do**
  for *each ES* $s \in \mathcal{S}$ ***and*** *MD* $n_s \in \mathcal{N}_s$ **do**
    Select offloading action $a_{n_s}^{(i)}$ based on $p_{n_s,a_{n_s}}^{(i)}$.
  end
  // Second Stage (Online)
  Calculate infrastructure state σ.
  while $\sum_{s\in\mathcal{S}}^{\sigma_s<0} \sigma_s < 0$ **do**
    Select transferring action α with the highest Q value. Execute α and produce σ'.
    $\sigma \leftarrow \sigma'$
  end
  for *each ES* $s \in \mathcal{S}$ ***and*** *MD* $n_s \in \mathcal{N}_s$ **do**
    Calculate reward $R_{n_s}^{(i)}$ using Eq. (2).
    Update decision probabilities $p_{n_s,a_{n_s}}^{(i)}$.
  end
  $i \leftarrow i + 1$
end

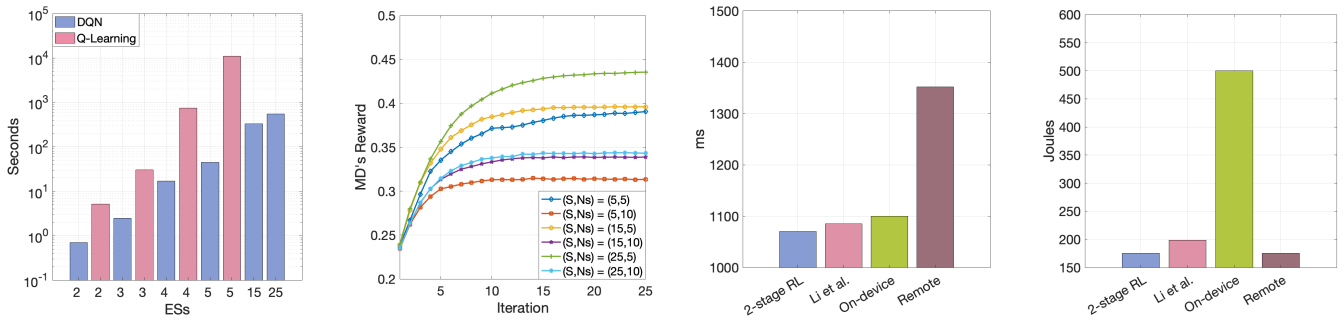
in Algorithm 3: (i) each MD of every ES temporarily selects whether to offload their current task or execute it locally, based on the probabilities of the SLA algorithm (Section III-A) and (ii) a load balancing is performed between the ESs to improve the response time of the offloaded tasks, based on an offline-trained neural network (Section III-B). The outcomes of this decision are fed back to the first stage to update the rewards and probabilities for the next iteration, until convergence to a stable solution that satisfies the delay and energy consumption constraints is achieved.

IV. Numerical Results

For the evaluation we consider a MEC infrastructure which consists of $2-25$ ESs, while $5-10$ MDs are connected to each ES. The available resources of each ES range between $\{8GHz, 48Mbps\}$ and $\{10GHz, 50Mbps\}$, while the MD capabilities between $\{1GHz, 498mW\}$ and $\{2GHz, 502mW\}$. Regarding the tasks we assume a single application executed by each one of the MDs with the following characteristics: $\{1000kbits, 2000Mcycles, 1080ms, 4J\}$. For the energy consumption per CPU cycle, following [4], we set $\kappa = 10^{-27}(f_{n_s})^2$. With regards to the First Stage of the algorithm, we opt for a balanced reward between energy consumption and delay (Eq. (2)) by setting $\beta_1 = \beta_2 = 0.5$ and $Z_1 = Z_2 = 1$; the learning parameter here is set to $b = 0.6$ and we assume that convergence is achieved when one action probability for every MD is ≥ 0.9. For the Second Stage, the reward weights are set as $\delta_1 = 0.0005$ and $\delta_2 = 50$, to balance the contribution of the load balancing and propagation delay minimization factors. The learning rate is set as $\zeta = 0.1$ and the discount factor as $\gamma = 0.9$. We utilize a neural network with 2 hidden layers with 700 and 600 neurons each. The size of the experience replay memory is set to 10^5 and the minibatch size $M = 256$. The weights of the target network $\hat{Q}$ are updated every 4 steps. We assume that training convergence is achieved when the average improvement in the reward (Eq. (3)) over the last 100 episodes is less than 1%.

Fig. 2 showcases the results of the evaluation. We start with comparing the training time between the DQN and the plain Q-learning algorithms, used for the Second Stage (Fig. 2a). Both trainings were performed on a MacBook Air with a 16-core Neural Engine M2 chip and 16GB of RAM. As seen there, the Q-learning algorithm training duration grows exponentially with the number of ESs, which makes it an unrealistic alternative when this number is greater than 5 (> 7 days of real time). On the other hand, DQN provides a far more tractable training process, with its duration showcasing a linear behavior to the number of ESs. Although the training is performed only once and offline, scalability is still important for applying our framework to larger infrastructures, which makes DQN a far more preferable choice.

Next, we examine the online convergence behavior of our algorithm; Fig. 2b illustrates the average collected reward (Eq. (2)), for various infrastructure settings over 25 iterations, for 100 experiment repeats. We observe that when fixing the



(a) (b) (c) (d)

Fig. 2: Evaluation: (a) Training Time, (b) Convergence Behavior, (c) Benchmarking: Delay and (d) Benchmarking: Energy

average number of MDs per site, increasing the number of ESs tends to yield a higher average reward for each MD, as more computational resources become available in the infrastructure. On the other hand, when fixing the number of ESs, increasing the number of average MDs per ES naturally results in lower rewards, as the competition for the available computational resources becomes stiffer. In any case, convergence to a stable solution is reached after 25 iterations which translates to less than $1sec$ of execution time, making our framework effectively a real-time decision-making tool.

Finally, we perform a comparative evaluation against a well-known work from the literature, Li et al. [4], and two baseline algorithms, one where all the tasks are executed on the MD ("On-device") and one where all the requests are offloaded ("Remote"). In this scenario, 25 ESs were considered, each one having an average of 7 MDs connected to it, which made some ESs overloaded when fully offloading. That is why the Remote's average achieved delay is the highest. We observe that our proposed solution manages to overcome this issue by transferring requests from the overloaded to the underloaded ESs. On the other hand, the algorithm in [4] selects the on-device execution for some MDs connected to the overloaded ESs, resulting in a slightly higher average delay. Regarding energy consumption, the On-device execution performs the worst, as expected, and the proposed solution being as energy efficient for the MDs as the Remote algorithm. The algorithm in [4] again scores slightly worse in this metric, as instead of transferring some tasks to underloaded sites, it selects the on-device execution. All in all, the presence of the load balancing mechanism in our framework, makes it capable of exploring the execution alternatives in the infrastructure, for MDs connected to overloaded ESs, achieving a better delay and energy consumption compared to typical on-device and/or remote execution solutions.

V. Conclusion

This paper presented a two-stage cooperative, reinforcement learning-based scheme for energy-aware computational offloading at the edge of the network. With this work, we aimed to minimize the application end-to-end delay as well as the energy consumption of the mobile devices during computationally intensive task execution. In the first stage, distributed and autonomous task offloading decision making in the mobile devices is enabled, based on Stochastic Learning Automata. The second stage devises a Q-learning approach to allow for task transferring between the edge sites and alleviate potential overloading phenomena. The results showed that the proposed framework outperforms the baseline solutions as well as a well-known similar work in the literature both in terms of delay and energy. Our future work will focus on incorporating the users' mobility in the decision making and eventually exploring alternative learning techniques for dealing with the increased dimensionality of the emerging state space.

References

[1] F. Saeik *et al.*, "Task offloading in Edge and Cloud Computing: A survey on mathematical, artificial intelligence and control theory solutions," *Computer Networks*, vol. 195, p. 108177, 2021.

[2] M. Avgeris *et al.*, "ENERDGE: Distributed energy-aware resource allocation at the edge," *Sensors*, vol. 22, no. 2, p. 660, 2022.

[3] S. Bouhoula, M. Avgeris, A. Leivadeas, and I. Lambadaris, "Computational offloading for the industrial internet of things: A performance analysis," in *2022 IEEE International Mediterranean Conference on Communications and Networking (MeditCom)*. IEEE, 2022, pp. 1–6.

[4] J. Li, H. Gao, T. Lv, and Y. Lu, "Deep reinforcement learning based computation offloading and resource allocation for MEC," in *2018 IEEE Wireless communications and networking conference (WCNC)*. IEEE, 2018, pp. 1–6.

[5] T. Fang, D. Wu, J. Chen, and D. Liu, "Cooperative Task Offloading and Content Delivery for Heterogeneous Demands: A Matching Game-Theoretic Approach," *IEEE Transactions on Cognitive Communications and Networking*, 2022.

[6] G. Yin, R. Chen, and Y. Zhang, "Effective task offloading heuristics for minimizing energy consumption in edge computing," in *2022 IEEE International Conferences on Internet of Things (iThings) and IEEE Green Computing & Communications (GreenCom) and IEEE Cyber, Physical & Social Computing (CPSCom) and IEEE Smart Data (SmartData) and IEEE Congress on Cybermatics (Cybermatics)*, 2022, pp. 243–249.

[7] F. Xu, Y. Xie, Y. Sun, Z. Qin, G. Li, and Z. Zhang, "Two-stage computing offloading algorithm in cloud-edge collaborative scenarios based on game theory," *Computers & Electrical Engineering*, vol. 97, p. 107624, 2022.

[8] Z. Kuang, Z. Ma, Z. Li, and X. Deng, "Cooperative computation offloading and resource allocation for delay minimization in mobile edge computing," *Journal of Systems Architecture*, vol. 118, p. 102167, 2021.

[9] N. Rasouli, R. Razavi, and H. R. Faragardi, "EPBLA: energy-efficient consolidation of virtual machines using learning automata in cloud data centers," *Cluster Computing*, vol. 23, no. 4, pp. 3013–3027, 2020.

[10] M. Diamanti, E. E. Tsiropoulou, and S. Papavassiliou, "Resource orchestration in uav-assisted noma wireless networks: A labor economics perspective," in *ICC 2021-IEEE International Conference on Communications*. IEEE, 2021, pp. 1–6.