Energy-Efficient Deadline-Aware Edge Computing: Bandit Learning with Partial Observations in Multi-Channel Systems

Babak Badnava*, Keenan Roach†, Kenny Cheung†, Morteza Hashemi*, Ness B Shroff‡

*Department of Electrical Engineering and Computer Science, University of Kansas

†Universities Space Research Association (USRA)

‡Department of Electrical and Computer Engineering, The Ohio State University

Abstract—In this paper, we consider a task offloading probl[em] in a multi-access edge computing (MEC) network, in which e[dge] users can either use their local processing unit to compute th[eir] tasks or offload their tasks to a nearby edge server thro[ugh] multiple communication channels each with different charact[er]istics. The main objective is to maximize the energy efficien[cy] of the edge users while meeting computing tasks deadlin[es.] In the multi-user multi-channel offloading scenario, users [are] distributed with partial observations of the system states. [We] formulate this problem as a stochastic optimization problem a[nd] leverage *contextual neural multi-armed bandit* models to deve[lop] an energy-efficient deadline-aware solution, dubbed E2DA. T[he] proposed E2DA framework only relies on partial state informat[ion] (i.e., computation task features) to make offloading decisio[ns.] Through extensive numerical analysis, we demonstrate that [the] E2DA algorithm can efficiently learn an offloading policy and achieve close-to-optimal performance in comparison with several baseline policies that optimize energy consumption and/or response time. Furthermore, we provide a comprehensive set of results on the MEC system performance for various applications such as augmented reality (AR) and virtual reality (VR).

Index Terms—Multi-access edge computing, energy efficiency, latency-sensitive applications.

I. INTRODUCTION

It is envisioned that the next generation wireless networks (5G-and-Beyond) will enable an unprecedented proliferation of computationally-intensive applications, such as face recognition, location-based AR/VR, and online 3D gaming [1]. However, adoption of these resource-hungry applications will be negatively affected by limited on-board computing and energy resources in edge devices. In fact, there is an ever-increasing demand for mobile computational power, while on-board resources remain constrained. In order to bridge this gap, multi-access edge computing (MEC) [2] has been contemplated as a solution to supplement the computing capability of the end-users. In contrast to the traditional cloud computing architectures, MEC leverages the radio access networks (RANs) to boost the computing power in the proximity to the end-users, thereby enabling the users to offload their computations to the MEC servers and achieve low-latency computations.

However, practical MEC architectures face significant challenges including efficient resource management (computing, energy, communication), coordination among distributed users, and providing guaranteed quality of service (QoS) for latency-critical services. Moreover, it is critical that any MEC management framework fully incorporates the underlying 5G-and-Beyond RAN architecture, such as multi-connectivity and multi-
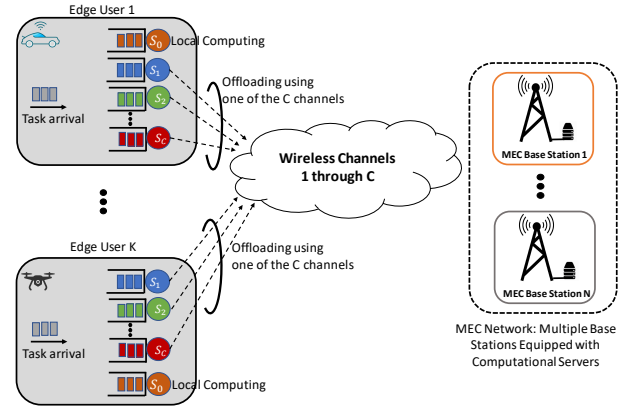

Fig. 1: Multi-User Multi-Channel MEC: Tasks arrive at the edge user randomly, and our E2DA agent decides, based on partial system information, to either compute each task locally or offload it to a nearby edge server through one of the C different wireless channels.

channel communication between edge users and base stations (edge servers) [3, 4]. Within this context, there are three distinct sets of requirements that must be met: (i) *network-imposed* constraints that determine the available communication links and their equality, as well as the number of edge servers, (ii) *user-imposed* constraints that could be expressed in terms of available energy resources and local computing power, and (iii) *task-imposed* constraints that dictate the required QoS (i.e., latency and deadline). To account for all these requirements, a decision-making agent is required to determine the optimal offloading policy. The offloading policy could be initiated by either edge users or edge servers. User-initiated task offloading provides better-personalized services tailored to each user's individual preference, especially when edge servers are managed by different operators [5]. However, distributed edge users only have access to *partial observations* of the stochastic and dynamic system states (e.g., link rates, computational power of the edge server, total number of users in the system, etc.). Hence, an efficient decision making policy should account for the stochastic nature of the system that affect latency and energy consumption, while being able to make offloading decisions in a partially observable environment.

A multitude of prior works have been devoted to investigate and improve various aspects of MEC systems. A group of work is dedicated to latency minimization in both partialy observable [5–7] and fully observable [8–11] environments. Several works investigated the energy efficiency [12–14] and the effect of

multi-connectivity technologies [8, 15, 16] on energy efficiency and latency in MEC systems. However, to the best of our knowledge, no prior research fully integrates all the previously mentioned constraints in a partially observable MEC setting. In this paper, we propose an energy efficient deadline-aware algorithm for task offloading in a decentralized multi-user multi-channel MEC setting, as depicted in Fig. 1. First, we develop a complete system model that incorporates the network model, edge users characteristics, and computation task features. Next, we formulate a stochastic optimization problem and cast the problem of task offloading as a contextual multi-user multi-armed bandit (MAB) framework, where offloading to an edge server, using one of multiple available communication channels, is regarded as playing an arm. We develop a sequential energy-efficient delay-aware (E2DA) offloading scheme to balance the offloading exploration-exploitation trade-off. E2DA can learn the optimal offloading decision for each user, while only has access to its own computation task information such as task size, computational intensity, and deadline. Our numerical results show that E2DA provides a promising service performance in comparison with three optimal baselines. In summary, the main contributions of this paper are as follows:

- We introduce a partially observable decentralized multi-user multi-channel task offloading schema, where users of a MEC network can offload their tasks to a nearby edge server using different communication channels. We provide a queuing model of the proposed schema and formulate a stochastic optimization problem to find the optimal offloading decision and the best wireless channel.
- We propose a partially observable decentralized contextual neural multi-armed bandit algorithm to find the optimal task offloading policy that maximizes the energy efficiency.
- We develop a simulator based on standardized LTE implementation and perform extensive simulations to analyze the behavior of our proposed method. Our analysis shows that task computational intensity has little impact on edge users' energy consumption, while task size has a significant impact on energy consumption. Moreover, we show that task size has a significant impact on increasing task response time due to increased transmission and computation time.

The rest of this paper is organized as follows. In Section II, we present the system model. In section III, we first formulate an energy-efficient deadline-aware task offloading optimization problem, and then presents our solution. In Section IV, we provide simulation results, and Section V concludes the paper.

II. SYSTEM MODEL

We consider a MEC network consisting of K edge users and N base stations, each of which is equipped with an edge server. As shown in Fig. 1, tasks arrive at the kth edge user according to an unknown random process (*e.g.,* Poison process). Then the scheduler (*i.e.,* E2DA agent) decides if the tasks should be computed locally or offloaded to the edge server. Edge users can communicate with the edge servers through C different channels (*i.e.,* using carrier aggregation or multiple radio access

technology (RAT)) to offload their tasks. We assume that each edge user is associated with one base station at a time.

On the MEC side, edge servers allocate a virtual machine (VM) to each edge user. We assume that edge servers allocate the same amount of computational resources (*i.e.,* one CPU core with frequency f_{mec}) to all the VMs, and tasks get service in a first-in-first-out (FIFO) manner. After the task computation is done, the edge server sends back the task result to the corresponding edge user. In this section, we elucidate how we model tasks, their arrival at the edge users, the communication channel used to send tasks to edge servers, the computation model on both edge users and edge servers, and the energy consumption model for both communication and computation.

A. Computation Task Model and Response Time

Task Model: Tasks arrive at the edge users according to an unknown random process. Each of these tasks is characterized by four different features: (i) task arrival time (in time), denoted by T_a; (ii) task size/length (in bits), denoted by S_{task}; (iii) task computational intensity (in CPU cycles per bit), denoted by I_{task}; and (iv) task deadline (in seconds), denoted by T_d.

As shown in Fig. 1, upon arrival, each task follows one of the two routes: it either gets computed locally or offloaded to the edge server for computation using one of the C communication channels. For example, in the case of an autonomous car as an edge user, object detection is a computationally intensive task that must be completed within some stringent deadline. In this scenario, the E2DA agent decides whether the object detection task should be computed locally or offloaded to an edge server.

In local computation, the scheduler adds the task to the processing queue at the edge user, and the task waits in the queue until the CPU picks it up for computation. Here, we assume that the CPU processes tasks in a FIFO manner. After the CPU finishes computing the task, the result of computation is ready, and the edge user can use that result, which is the point that the task exits the system. In the example of autonomous cars, the generated result is the image in which all cars and objects are detected. On the other hand, when the scheduler decides to offload the task, it adds the task to the transmission queue that corresponds to the c'th communication channel. Then, on the MEC side, tasks are computed by the edge processing unit in a FIFO manner. After computing a task, a result with the size of S_{res} bits is generated to be sent back to the corresponding edge user over the same wireless channel used for offloading.

Task Response Time. To accurately model task response time, we note that each task could experience several types of delays, namely: (1) execution time, (2) transmission/reception time, and (3) queue waiting time. In the following, we calculate each of these delays separately.

(1) Task execution time. From [17], the amount of time it takes a CPU with an operating frequency of f (in cycle per second) to compute a task with size S_{task} (in bits) and computational intensity I_{task} (in cycles per bit) is calculated as:

$$T_e(f, S_{task}, I_{task}) = \frac{S_{task}I_{task}}{f} \quad [Seconds]. \tag{1}$$

Hence, given CPU frequency of the allocated VM (*i.e.,* f_{mec}) and edge user f_{eu}, the execution time for offloaded tasks is calculated as $T_e^{offload} = \frac{S_{task}I_{task}}{f_{mec}}$, and the execution time for locally computed tasks is computed as $T_e^{local} = \frac{S_{task}I_{task}}{f_{eu}}$.

(2) Task transmission/reception time. We consider C different communication channels between the k-th edge user and the n-th base station. We denote $R_{k,n}^c$ and $R_{n,k}^c$ for the uplink and downlink transmission rates over the c-th wireless channel, respectively. Hence, task transmission time and result reception time are calculated using $T_{rx}^c = \frac{S_{res}}{R_{n,k}^c}$ and $T_{tx}^c = \frac{S_{task}}{R_{k,n}^c}$, respectively.

(3) Task queue waiting time. Each task goes through a different set of queues. A locally computed task waits for W_1 seconds in the local processing queue. On the other hand, an offloaded task waits for W_2^c seconds in the transmission queue of the c-th wireless channel to be sent over the edge server, then waits for W_3 seconds in the edge processing unit, and finally waits for W_4^c seconds in the transmission queue for its result to be sent back.

Given the three sources of delay, locally computed tasks only wait in the local processing queue to get service. Thus, we have:

$$T_r^{local} = W_1 + T_e^{local} \quad [Seconds]. \quad (2)$$

On the other hand, offloaded tasks wait in two transmission queues to be transmitted over the air and the edge processing queue to get served. Thus, the response time is:

$$T_r^{offload} = W_2^c + T_{tx}^c + W_3 + T_e^{offload} + W_4^c + T_{rx}^c \quad [Seconds]. \quad (3)$$

B. Energy Consumption Model

The edge user's battery is depleted by three different components, (i) its CPU for local computation, (ii) its wireless transmitter for offloading tasks to the edge server, and (iii) its receiver for receiving results of computations. We model energy consumption of each of these components separately.

The main source of task execution energy consumption on edge devices is the CPU dynamic power usage [18]. We adopt the CPU energy consumption model used in [18] such that the energy consumption by a CPU with frequency f_{eu} to compute a task is $E_e = \kappa S_{task}I_{task}f_{eu}^2$, where κ is the CPU capacitance factor, S_{task} is task size and I_{task} is task computational intensity.

The power consumption for transmission and reception of a task over a wireless channel varies based on the used wireless technology (*e.g.,* Wi-Fi, LTE/5G) [19]. We use a general model in which the transmitter energy consumption is given by:

$$E_{tx}^c = T_{tx}^c P_{k,n}^c \quad [Joule], \quad (4)$$

where T_{tx}^c denotes the task transmission time over the c-th wireless channel, and $P_{k,n}^c$ is the power allocation profile. Similarly, the receiver energy consumption is:

$$E_{rx}^c = T_{rx}^c P_k^c \quad [Joule]. \quad (5)$$

where T_{rx}^c denotes the task result reception time over the c-th wireless channel, and P_k^c is the reception power consumption profile. Hence, the total energy consumed by the edge user is:

$$E_{tot}^c = E_{tx}^c + E_e + E_{rx}^c \quad [Joule]. \quad (6)$$

Note that E_{tx}^c and E_{rx}^c are zero for local computation.

III. Energy-Efficient Deadline-Aware Offloading

Our goal is to design a policy that achieves the maximum energy efficiency (*i.e.,* $T_r E_{tot}^c / S_{task}$) while ensuring that tasks deadlines are always met. This is critical for latency-sensitive applications such as autonomous cars. To this end, there are two decision variables. The first one is u, which corresponds to the offloading decision, and the second one is c, which corresponds to the index of the channel that will be used in case of offloading. Here, we formulate the following stochastic optimization problem:

$$\textbf{E2DA-Opt:} \begin{cases} \max_{u,c} & \mathbb{E}\left[\frac{S_{task}}{T_r E_{tot}^c}\right] & [(Bits/Second)/Joule] \\ \text{subject to:} & T_r \leq T_d \\ & \text{Eqs. 2,3,6,} \end{cases}$$
$$(7)$$

where S_{task} is the task size, T_r denotes the task response time, E_{tot}^c is the task energy consumption, and T_d denotes the task deadline. The first constraint makes sure that each task meets its deadline. The rest of the constraints correspond to the system model as explained in section II. This stochastic optimization problem is difficult and challenging to solve every time a new task arrives. One main challenge is that some task features (*e.g.,* transmission energy consumption, transmission time, waiting time, etc.) are not observable at the decision-making stage. Moreover, the environment is uncertain and stochastic due to time-varying wireless communication channel conditions. Thus, a robust offloading policy needs not only to capture the uncertainty of the environment but also makes its decision based on the partially observable state information at the offloading stage.

Contextual Multi-Armed Bandit (CMAB) Approaches: CMAB solutions belong to the general class of online learning approaches by which a learner converges to the optimal solution through repeated interactions with an environment. In each round, the learner is presented with a set of actions, each of which is associated with a multi-dimensional feature vector (*i.e.,* contextual information). After choosing an action (*i.e.,* playing an arm), the learner will receive a stochastic reward generated from some unknown distribution conditioned on the action's feature vector. During the learning process, the learner tries to balance the trade-off between exploration and exploitation, and its goal is to maximize the expected cumulative rewards over a finite number of trials. CMAB algorithms have been applied to many real-world applications, such as video streaming quality of experience improvement [20] and recommendation systems [21]. Zhou et al. in [22] provide a comprehensive comparison of the regret growth for different synthetic datasets. To handle the trade-off between the implementation simplicity and regret bound, we leverage Neural ε-Greedy algorithm, which uses a neural network to model the expected reward given the contextual information. This, in turn, will enable us to capture the non-linearity of the reward function.

Energy-Efficient Deadline-Aware CMAB: Every CMAB problem is defined using a set of contextual information X and a set of actions A. Here, the contextual information set is a multi-

---

Algorithm 1 Energy-Efficient Deadline-Aware CMAB

Inputs:

   Neural network Q initialized by θ_0; Task replay buffer $\mathscr{B}$;

Algorithm:

1: **for** epoch $e = 1$ to $\mathscr{E}$ **do**
2:   **for** each user k in K **do**
3:     **for** each task upon arrival **do**
4:       Observe contextual info. $x = [S_{task}, I_{task}, T_d]$
5:       $a = \varepsilon\text{--}Greedy(Q(x|\theta))$
6:       **if** $a = 0$ (*local computing*) **then**
7:         Compute T_r using Eq. 2
8:         $E_{tot} = E_e$
9:       **else**
10:         $c = a$
11:         Compute T_r, E_{tx}^c, E_{rx}^c using Eq. 3, 4, 5.
12:         $E_{tot} = E_{tx}^c + E_{rx}^c$
13:       **end if**
14:       $r = \mathbb{1}_{T_r \leq T_d}\left(\frac{S_{task}}{T_r E_{tot}} + \lambda\right) - \lambda$
15:       Add tuple (x, a, r) to replay buffer $\mathscr{B}$
16:       Let $\theta = TrainNN(\mathscr{B}, \theta_0)$
17:     **end for**
18:   **end for**
19: **end for**

---

dimensional vector consisting of task information. The task information, which is available during the decision-making stage, includes task size S_{task}, task computational intensity I_{task}, and task deadline T_d. The action set A is, however, a discrete set with $C + 1$ possibilities (*i.e.,* one possibility for local computation, and one for each of the C wireless communication channels). We designate $a = 0$ to local computation, and $a \in \{1, 2, ..., C\}$ corresponding to the channel chosen for the offloading process.

In order to solve the introduced energy efficient deadline-aware optimization problem, it is important to note that the objective function in Eq. 7 and the objective of bandit problems are analogous to each other in the sense that the objective in both is to maximize the expected value of a term; $\frac{S_{task}}{T_r E_{tot}^c}$ in Eq. 7 and reward in bandit problems. However, bandit problems are designed for unconstrained problems, while Eq. 7 contains some equality and inequality constraints. Equality constraints (*i.e.,* Eqs. 2, 3, and 6) are part of the system model (*i.e.,* MEC environment) and are already incorporated in the reward function, however the only inequality constraint (*i.e.,* $T_r \leq T_d$) is not included in the objective function. Thus, we need to incorporate the first inequality constraint, in Eq. 7, in the reward function in a way that the learner gets penalized whenever the decision it makes leads to not meeting the deadline. Hence, the following reward function is considered:

$$r = \mathbb{1}_{T_r \leq T_d}\left(\frac{S_{task}}{T_r E_{tot}^c} + \lambda\right) - \lambda,$$

where λ is the penalization factor for tasks that have not met their deadlines, and $\mathbb{1}_{T_r \leq T_d}$ is an indicator function that returns one only when the task meets its deadline. Therefore, if a task would meet its deadline, the reward function gives away

a reward equal to $S_{task}/T_r E_{tot}^c$ for the task in hand, and $-\lambda$ otherwise. Note that S_{task} (*i.e.,* task size) for a given task is constant, however, both T_r (*i.e.,* task response time) and E_{tot}^c (*i.e.,* task energy consumption) are stochastic values influenced by wireless channel condition and the number of users using that channel.

The condition of the wireless channel depends on several factors such as user mobility, propagation environment, distances, etc. [23]. These factors introduce uncertainties in the decision-making process. The goal of the E2DA policy is to capture these uncertainties, thereby achieving a policy that is able to make a decision in the presence of these uncertainties. Furthermore, it is important for the E2DA policy to consider the allocation of communication resources, as wireless channels may be utilized by multiple users simultaneously, which leads to a reduced reward per user. Overall, Algorithm 1 presents the complete training process of E2DA that implements the ε-Greedy CMAB algorithm based on the defined contextual information and reward function.

IV. NUMERICAL RESULTS

A. Simulation Setting, Training, and Baseline Algorithms

Communication and Computation Models: To create a set of computing tasks for E2DA training, we designed a simulation framework for MEC using the ns-3 network simulator. This discrete event simulator incorporates several communication technologies, such as LTE, 5G, and Wi-Fi. While MEC is being standardized to be included in 5G-and-Beyond networks [24], we use the ns-3 LTE module that is more developed and stable compared with the ns-3 5G module [25]. The edge users offload their tasks to the nearby base station using one of $C = 3$ channels. Each of these component carriers operates on a different frequency (*i.e.,* 700 MHz, 1500 MHz, and 2600 MHz).

Computation tasks arrive at the k-th edge user according to a Poisson process with the mean arrival rate of $\lambda_k = 40$ tasks per second. The task size is set according to a uniform distribution with min 10 bits per task and max 75 Kbit per task; The task intensity is set according to a uniform distribution with min of 10 cycles per bit and max of 1000 cycles per bit; and the task deadline follows a uniform distribution with min of $10ms$ and max of $18ms$.

Dataset Generation and Training of E2DA: We used the developed simulator to generate a set of computation tasks for E2DA training. For each task generated by the simulator, we measured waiting time, computation time, transmission time, response time, and energy consumption for both transmission and computation alongside task features (*i.e.,* task size, computational intensity, and deadline). We combined all these tasks to form a dataset containing a total of 32,565 tasks. Given the generated dataset, our E2DA agent is modeled by a two-layer neural network with 50 neurons on each layer. We use ReLu as activation function for hidden layers, and Sigmoid as activation function of the output layer. The input is the task features, which is normalized between zero and one, and the output is the predicted reward obtained by performing each

(a) Learning Curve   (b) Percentage Met their Deadlines   (c) Accumulated Energy Consumption   (d) Accumulated Response Time
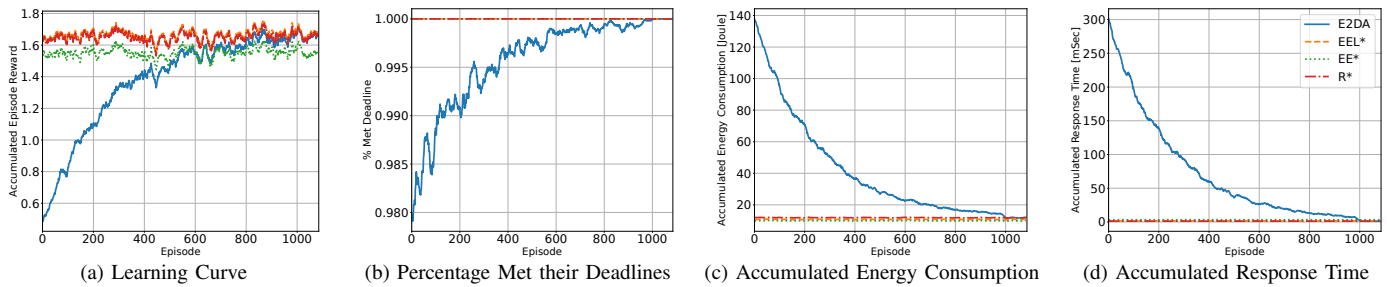
Fig. 2: Performance of E2DA method in comparison with the baselines during the learning process in an environment with $K = 5$ edge users and $N = 3$ base stations. A moving average with a window size of 20 has been applied to these plots to smooth out the fluctuations.

action, which is also normalized between zero and one. We use the mean squared error as the loss function for this network and use the RMSprob optimizer to train the network for 1000 episodes and find the optimal parameters of the network. After the training is done, the E2DA agent is set to exploit (*i.e.,* testing phase) the learned policy for another 100 episodes to evaluate the performance of the learned offloading policy. In each episode, the E2DA agent sees 100 tasks, randomly taken from the dataset, and makes an offloading decision based on the task features.

Baseline Algorithms: We compare E2DA with three baselines, namely *(1) Optimal Energy Efficient and Latency Baseline (EEL*)*, *(2) Optimal Energy Efficient Baseline (EE*)*, and *(3) Optimal Response Time Baseline (R*)*. Each of these policies has access to a piece of unobservable information during the decision-making stage, which makes them superior to E2DA. *EEL** provides the optimal offloading strategy (*i.e.,* the action with the highest value of $S_{task}/(T_r E_{tot}^c)$) for the optimization problem (Eq. 7) without considering the deadline constraint. *EE** makes an offloading decision by choosing the action that yields the highest S_{task}/E_{tot}^c, and *R** makes an offloading decision by choosing the action that yields the lowest task response time T_r.

B. Performance Evaluation

Training Performance: Fig. 2 shows the performance of E2DA in comparison with the optimal baselines during the training and testing phases. To generate a dataset for this case, we set $K = 5$ edge users and $N = 3$ base stations. Fig. 2a shows the E2DA accumulated episode reward. The E2DA, after 1,000 episodes of training, outperforms both R^* and EE^* and converges to EEL^*. Fig. 2b demonstrates the fraction of tasks that have met their deadlines in each episode of learning, which increases as the agent progresses toward the optimal policy. Fig. 2c shows the energy consumption profile of the edge user during the learning process. R^* provides the worst energy consumption, while E2DA provides a performance very close to EEL^*. Fig. 2d shows the accumulated tasks' response time for each episode. As expected, R^* provides the optimal response time, however, E2DA converges to R^* and EEL^* in terms of response time.

Effect of Task Distribution: As mentioned earlier, emerging computationally intensive applications (*e.g.,* AR/VR, autonomous cars, conversational robots) are increasingly being deployed at the network edge. Tasks generated by each application/user have different characteristics. For example, AR/VR

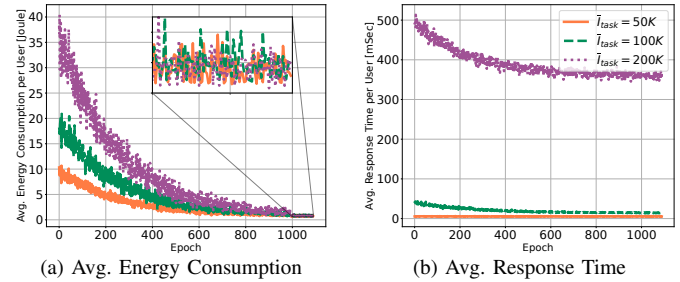(a) Avg. Energy Consumption   (b) Avg. Response Time

Fig. 3: Effect of task computational intensity. $K = 10$ edge users and $N = 3$ edge servers. In all cases, the mean task size is set to $\bar{S}_{task} = 5\,Kbit$ and the mean arrival rate is set to $\lambda_k = 20\,task/sec$.

devices generate tasks at a high rate with large sizes that require high computational power (larger computational intensity); A conversational robot, such as ChatGPT, however, generates tasks at a lower rate with small sizes (*i.e.,* a question in the form of a sentence asked by a user) that need higher computational power. In this section, we analyze the effect of each of the tasks features on energy consumption and response time.

Fig. 3a reveals that a 4X increase in computational intensity (50K vs. 200K) results in an average of 1.04X increase in energy consumption during the testing phase. This means that an increasing task computational intensity has little to no effect on the energy consumption. Fig. 3b, on the other hand, shows that a 2X and 4X increase in computational intensity results in an average of 2.6X and 65.6X increase in response time, respectively. This means that computational intensity has a significant impact on the response time due to the increased computation and queuing times on both edge and local processing units. In this analysis, the mean task size is set to $\bar{S}_{task} = 5Kbit$ and the mean arrival rate for all the ten users is set to $\lambda_k = 20\,task/sec$.

Fig. 4 demonstrates the effect of increasing task size on the average task energy consumption and task response time. The results show that increasing the task size by 2X leads to an almost 50X increase in energy consumption and 7X increase in response time. This reveals that task size is a major contributing factor in increasing both energy consumption and response time. In this analysis, the mean tasks' arrival rate is set to $\lambda_k = 60\,task/sec$ and the mean computational intensity is set to $\bar{I}_{task} = 5K\,cycle/bit$ (*e.g.,* AR/VR applications).

(a) Avg. Energy Consumption (b) Avg. Response Time

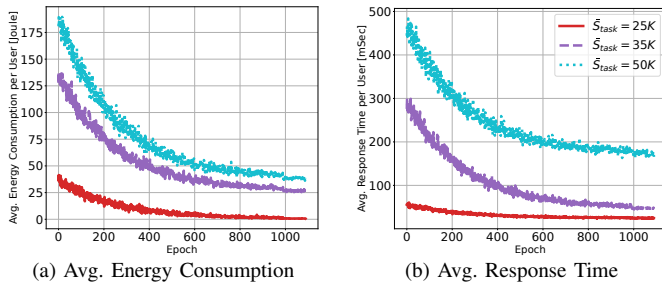Fig. 4: Effect of task size. $K = 10$ edge users and $N = 3$ edge servers. In all cases, the mean tasks' arrival rate is set to $\lambda_k = 60\, task/sec$ and the mean computational intensity is set to $I_{task} = 5K\, cycle/bit$.

V. Conclusion

In this paper, we considered the problem of partially observable decentralized multi-channel task offloading, where K edge users choose to offload their tasks to a nearby base station using one of the C wireless communication channels or compute the task locally ($C + 1$ actions in total). The overall objective is to optimize energy efficiency while meeting each task deadline. However, the system is partially observable in the sense that not all the system states (channel rates, network traffic, etc.) are available at the decision-making stage. Leveraging the CMAB models, we developed the E2DA algorithm that utilizes task features (*i.e.,* task size, computational intensity, and deadline) to make an offloading decision using one of the C available channels, or perform the computation locally, without a need for full state information. Through numerical results we showed that the E2DA policy learns the offloading policy that converges to optimal baseline solutions. Moreover, we demonstrated that task computational intensity has little impact on edge users' energy consumption and a larger impact on response time, while task size has a significant impact on both energy consumption and task response time due to an increased transmission and computation time.

VI. Acknowledgement

References

[1] SeaGate, "State of the Edge," https://www.seagate.com/www-content/enterprise-storage/it-4-0/images/Data-At-The-Edge-UP1.pdf, 2019, [Online].

[2] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge Computing: Vision and Challenges," *IEEE Internet of Things Journal*, 2016.

[3] G. Yao, M. Hashemi, R. Singh, and N. B. Shroff, "Delay-Optimal Scheduling for Integrated mmWave Sub-6 GHz Systems with Markovian Blockage Model," *IEEE Tran. on Mob. Comp.*, 2022.

[4] M. Giordani, M. Mezzavilla, S. Rangan, and M. Zorzi, "Multi-connectivity in 5G mmWave cellular networks," in *Mediterranean Ad Hoc Networking Workshop (Med-Hoc-Net)*, 2016.

[5] X. Wang, J. Ye, and J. C. Lui, "Decentralized Task Offloading in Edge Computing: A Multi-User Multi-Armed Bandit Approach," in *IEEE INFOCOM*, 2022.

[6] T. Ouyang, R. Li, X. Chen, Z. Zhou, and X. Tang, "Adaptive User-managed Service Placement for Mobile Edge Computing: An Online Learning Approach," in *IEEE INFOCOM*, 2019.

[7] M. Molina, O. Muñoz, A. Pascual-Iserte, and J. Vidal, "Joint scheduling of communication and computation resources in multiuser wireless application offloading," in *IEEE 25th PIMRC*, 2014.

[8] B. Wu, T. Chen, K. Yang, and X. Wang, "Edge-Centric Bandit Learning for Task-Offloading Allocations in Multi-RAT Heterogeneous Networks," *IEEE Tran. on Vehicular Technology*, 2021.

[9] Y. Li, T. Wang, Y. Wu, and W. Jia, "Optimal dynamic spectrum allocation-assisted latency minimization for multiuser mobile edge computing," *Digital Communications and Networks*, 2022.

[10] Z. Jia, Z. Zhou, X. Wang, and S. Mumtaz, "Learning-Based Queuing Delay-Aware Task Offloading in Collaborative Vehicular Networks," in *IEEE ICC*, 2021.

[11] S. Huang, L. Li, Q. Pan, W. Zheng, and Z. Lu, "Fine-Grained Task Offloading for UAV via MEC-Enabled Networks," in *IEEE 30th International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC Workshops)*, 2019.

[12] W. Liu, Y. Xu, N. Qi, K. Yao, Y. Zhang, and W. He, "Joint Computation Offloading and Resource Allocation in UAV Swarms with Multi-access Edge Computing," in *International Conference on Wireless Communications and Signal Processing*, 2020.

[13] J. Zhang, H. Guo, and J. Liu, "Energy-Aware Task Offloading for Ultra-Dense Edge Computing," in *IEEE iThings and IEEE GreenCom and IEEE CPSCom and IEEE SmartData*, 2018.

[14] S. Zhu, L. Gui, J. Chen, Q. Zhang, and N. Zhang, "Cooperative Computation Offloading for UAVs: A Joint Radio and Computing Resource Allocation Approach," in *IEEE International Conference on Edge Computing (EDGE)*, 2018.

[15] A. Sacco, F. Esposito, G. Marchetto, and P. Montuschi, "A Self-Learning Strategy for Task Offloading in UAV Networks," *IEEE Transactions on Vehicular Technology*, 2022.

[16] ——, "Sustainable Task Offloading in UAV Networks via Multi-Agent Reinforcement Learning," *IEEE Trans. on Vehi. Tech.*, 2021.

[17] B. Badnava, T. Kim, K. Cheung, Z. Ali, and M. Hashemi, "Spectrum-Aware Mobile Edge Computing for UAVs Using Reinforcement Learning," in *IEEE/ACM SEC*, 2021.

[18] W. Zhang, Y. Wen, K. Guan, D. Kilper, H. Luo, and D. O. Wu, "Energy-Optimal Mobile Cloud Computing under Stochastic Wireless Channel," *IEEE Trans. on Wireless Comm.*, 2013.

[19] B. Dusza, C. Ide, L. Cheng, and C. Wietfeld, "CoPoMo: a context-aware power consumption model for LTE user equipment," *Trans. on Emerging Telecomm. Tech.*, 2013.

[20] B. Badnava, S. Reddy Chintareddy, and M. Hashemi, "QoE-Centric Multi-User mmWave Scheduling: A Beam Alignment and Buffer Predictive Approach," in *IEEE ISIT*, 2022.

[21] L. Tang, Y. Jiang, L. Li, C. Zeng, and T. Li, "Personalized Recommendation via Parameter-Free Contextual Bandits," in *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2015.

[22] D. Zhou, L. Li, and Q. Gu, "Neural Contextual Bandits with UCB-Based Exploration," in *Proceedings of 37th ICML*, 2020.

[23] T. 36.777, *Technical Specification Group Radio Access Network; Study on Enhanced LTE Support for Aerial Vehicles.* 3GPP TSG RAN: V15.0.0, 2017.

[24] S. Kekki, W. Featherstone, Y. Fang, P. Kuure, A. Li, A. Ranjan, D. Purkayastha, F. Jiangping, D. Frydman, G. Verin, K.-W. Wen, K. Kim, R. Arora, A. Odgers, L. M. Contreras, and S. Scarpina, "MEC in 5G networks," in *ETSI*, 2018. [Online]. Available: https://www.etsi.org/images/files/ETSIWhitePapers/etsi_wp28_mec_in_5G_FINAL.pdf

[25] K. Koutlia, B. Bojovic, Z. Ali, and S. Lagén, "Calibration of the 5G-LENA system level simulator in 3GPP reference scenarios," *Simulation Modelling Practice and Theory*, 2022.