Deep Reinforcement Learning based Resource Allocation in Low Latency Edge Computing Networks

Tianyu Yang[1], Yulin Hu[1], M. Cenk Gursoy[2], Anke Schmeink[1] and Rudolf Mathar[1]
[1]Institute for Theoretical Information Technology, RWTH Aachen University, D-52074 Aachen, Germany
Email: {$yang, hu, anke.schmeink, mathar$}@ti.rwth-aachen.de
[2]Department of Electrical Engineering and Computer Science, Syracuse University,
Syracuse, NY 13244, USA. E-mail: $mcgursoy$@syr.edu

***Abstract*—In this paper, we investigate strategies for the allocation of computational resources using deep reinforcement learning in mobile edge computing networks that operate with finite blocklength codes to support low latency communications. The end-to-end (E2E) reliability of the service is addressed, while both the delay violation probability and the decoding error probability are taken into account. By employing a deep reinforcement learning method, namely deep Q-learning, we design an intelligent agent at the edge computing node to develop a real-time adaptive policy for computational resource allocation for offloaded tasks of multiple users in order to improve the average E2E reliability. Via simulations, we show that under different task arrival rates, the realized policy serves to increase the task number that decreases the delay violation rate while guaranteeing an acceptable level of decoding error probability. Moreover, we show that the proposed deep reinforcement learning approach outperforms the random and equal scheduling benchmarks.**

***Keywords*—Edge computing, deep reinforcement learning, finite blocklength coding, ultra-reliable low-latency communications (URLLC)**

I. Introduction

Mobile edge computing (also known as fog computing) has emerged and fast developed in recent years, as it eliminates delays by providing large amounts of computational resources for tasks with high computational demands in applications that require low latency, e.g., industrial automation, augmented & virtual reality (AR & VR), drones and remote control. Especially, in applications involving wireless machine-type communications (MTC) or Internet of Things (IoT), device normally lack the computational resources and thus offload the data to the edge computing nodes. Such computing nodes process the data locally and send time-critical information, e.g., collision warnings or actuation commands, to a wireless terminal [1], [2]. The packet size of the data that needs to be sent to the remote terminal is also generally small [3]. Hence, benefiting from the high processing power of the edge computing nodes, the end-to-end (E2E) delay is reduced and the edge computing systems can provide a near real-time service.

While the edge computing systems reduce the large delays incurred by the long computational time in the MTC or IoT devices, they suffer from the potentially low quality of the wireless channels in uplink and downlink. The E2E reliability therefore not only depends on the delay violation probability but also is influenced by the decoding error probability. In particular, to support ultra-reliable low latency communications (URLLC) [4], the coding blocklength is required to be relatively short, i.e., the impact of finite blocklength (FBL) coding [5] should be taken into account. Unlike the infinite blocklength (IBL) regime where the blocklength is assumed to be unbounded, in the FBL regime the blocklength is so short that the data transmission is no longer arbitrarily reliable. Specifically, the error probability becomes significant even when the data coding rate is below the Shannon limit. Hence, the decision on how to effectively choose the coding rate becomes crucial to make the transmissions meet the required reliability level. This leads to a trade-off within the decision strategy of choosing a proper coding blocklength based on the task packet size and the channel quality so that the required reliability level is satisfied meanwhile the delay limit is not violated.

The computational resources of one mobile edge computing (MEC) node are typically capable of servicing the offloaded tasks of more than one mobile user. Note that in URLLC-supported networks the frame lengths are short, i.e., the time/blocklength for computing/transmissions in each frame is limited. In addition, the data arrival rate of each user is varying over time. Hence, offloaded data may need to wait in a buffer at the MEC node before it can be processed and transmitted. In particular, for the considered edge computing network with a single MEC node but multiple users, the MEC node must have a proper policy for allocating the computational resources to the data from different users. This policy is expected to utilize the channel diversity (over time and among users), this could result in the transmission of the processed data (with given size) via a relatively shorter blocklength and therefore save a bit more time for the computing. Hence, when the quality of the downlink channel of a user is very low, the MEC node potentially allocates none of the computational resources for the task of the corresponding user and let it wait until the realization of an improved channel quality to meet the requirement of the error probability. At the same time, we should also note that the longer waiting time of the tasks in the buffers also prolong their E2E delay and potentially increases the queue length in the buffers since the new offloaded tasks keep arriving. Once the offloaded data is processed in the MEC node, as small a downlink transmission rate as possible (corresponding to as large a coding block as possible) is chosen to decrease the error probability while not violating the delay limit in the downlink channels. Hence the computational resource allocation strategy can directly affect the E2E performance. Therefore in order to maximize

the average successful transmission rate (i.e., the percentage of the transmitted data that meets the required reliability level) and minimize the average delay violation rate, a smart dynamic policy for computational resource allocation for multiple users should be addressed based on the channel quality, data packet size and the existing waiting time.

Conventionally the problem is solved analytically using queuing analysis and optimization theory [6], [7]. However, the problem can easily become intractable due to its stochastic nature. On the other hand, the frame-based edge computing process can be viewed as a Markov decision process (MDP). Consequently, reinforcement learning can enable the MEC node to become an intelligent agent that learns the optimal resource allocation strategy via a large number of trial and error interactions. In this paper, we address a deep reinforcement learning based computational resource allocation policy for a URLLC edge computing network with multiple users. To the best of our knowledge, this paper is the first to investigate the feasibility of deep reinforcement learning methods (particularly deep Q-learning) in a MEC environment within FBL regime.

The remainder of this paper is structured as follows. In Section II, the MEC system model including the channel model, FBL regime and computation model is presented. In Section III, the deep reinforcement learning based approach is described along with the formalization of the Markov decision process and the algorithm of the deep Q-learning. The performance of the proposed deep reinforcement learning based scheme in the MEC system is then numerically evaluated in Section IV and finally the conclusions are drawn in Section V.

II. System Model

We consider a edge computing network with one MEC node, N mobile users and N corresponding control terminals, as depicted Fig. 1. The MEC node is available with C total CPU cores for processing the offloaded tasks. The operating time of the edge computing system is slotted and each time slot is considered as one frame with the length of T_f symbols. One symbol time is denoted by t_s (in seconds). The index of the frames are denoted by k with $k \in \mathcal{K} = \{0, 1, 2, ...\}$. Each frame contains three phases. In the first phase the user U_n offloads a data packet via uplink (UL) with size $D^{(k)}_{\mathrm{UL},n}$ (in bits), length $m^{(k)}_{\mathrm{UL},n}$ (in symbols) and the probability $P^{(k)}_n$. The probability $P^{(k)}_n$ following the principle of the discrete-time On/Off Markov arrival model [8] represents the data arrival level of the user U_n in time slot k. The offloaded data from user n will wait in the corresponding buffer n, where the queues in the buffers are first-in-first-out (FIFO) queues. Subsequently, in the second phase the MEC node allocates the computational resources (CPU cores) for offloaded data. The MEC node also has the choice to allocate none of the resources for the processing of the offloaded data and let it wait for next allocation time in case of a poor channel quality in the corresponding downlink. Nevertheless, the offloaded tasks require an E2E delay limit d_r, where for simplicity we assume $d_r = \alpha T_f$ with a positive integer coefficient α. If the task waits so long that the delay limit is violated, the task is considered as timed out and is dropped from the buffer. Note that this waiting operation makes the problem intractable to be solved using analytic optimization methods or traditional heuristic methods. Finally, in the last phase, the processed data for user U_n with size $D^{(k)}_{\mathrm{DL},n}$ (in bits) and length $m^{(k)}_{\mathrm{DL},n}$ (in symbols) is transmitted from the MEC node to the corresponding control terminal T_n through the downlink channels. We assume that the data size for one task in the uplink and downlink have a linear relation, i.e., $D_{\mathrm{DL}} = \beta D_{\mathrm{UL}}$, where β is a coefficient. Please note that in one frame, uplink and downlink transmissions with data sizes $D^{(k)}_{\mathrm{UL},n}$ and $D^{(k)}_{\mathrm{DL},n}$ could belong to different tasks since the tasks can potentially wait in the buffers for several frames.

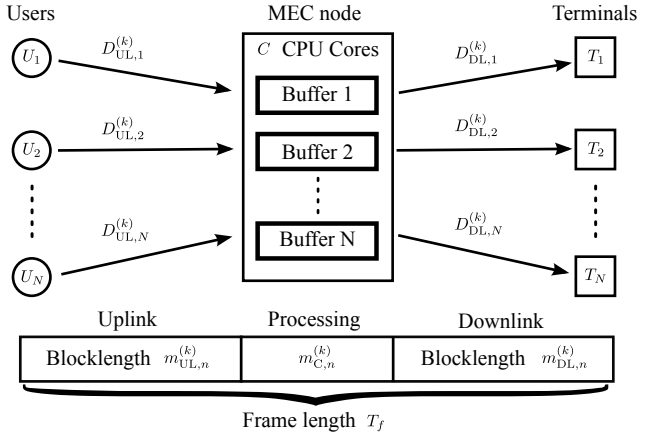

Figure 1. The studied multi-user edge computing system, including N mobile users, N corresponding control terminals and one MEC node with N data buffers and C total CPU cores. Upper-index k is the time slot index.

A. Channel model

We assume that the channels experience time-varying correlated Rayleigh fading and follow a quasi-stationary and frequency flat-fading model. Hence, the channel state in one frequency band is assumed to stay constant in one frame and vary with a certain correlation across frames. Nevertheless, the channel states in different frequency bands (for different users in uplink and downlink) are assumed to be independent. Under the considered channel model we denote the signal-to-noise rations (SNR) of the downlink n in frame k by $\eta^{(k)}_{\mathrm{DL},n} = \bar{\eta}_{\mathrm{DL},n}|h^{(k)}_{\mathrm{DL},n}|^2$, where $\bar{\eta}_{\mathrm{DL},n}$ is the average SNR of the downlink n and $h^{(k)}_{\mathrm{DL},n}$ is the channel fading coefficient of the downlink n in frame k. Note that the channel coefficients are correlated in neighbouring frames. We adopt the Jakes model for the correlation [9]:

$$h^{(k)}_{\mathrm{DL},n} = \rho_{\mathrm{DL},n} h^{(k-1)}_{\mathrm{DL},n} + \sqrt{1-\rho^2_{\mathrm{DL},n}}\,\hat{h}^{(k)}_{\mathrm{DL},n}, \qquad (1)$$

where $\rho_{\mathrm{DL},n}$ is the channel correlation coefficient of downlink n and $\hat{h}^{(k)}_{\mathrm{DL},n} \sim \mathcal{CN}(0,1)$ is a complex Gaussian random variable with zero mean and unit variance. Instantaneous channel state information (CSI) is assumed to be available at the MEC node. The reinforcement learning agent is expected to learn the correlation property of the channel and be able to predict the channel quality in the near future and take advantage of this to optimize the real-time policy.

B. Finite blocklength codes

In the FBL regime, the decoding error probability must be considered. In our edge computing system we assume that the offloaded data in the MEC node can be decoded correctly and we address the reliability in the downlink channels. The

coding rate of downlink in frame k for terminal T_n with blocklength $m_{\mathrm{DL},n}^{(k)}$, SNR $\eta_{\mathrm{DL},n}^{(k)}$ and error probability $\varepsilon_{\mathrm{DL},n}^{(k)}$ is shown to have the following approximation [5]:

$$\frac{D_{\mathrm{DL},n}^{(k)}}{m_{\mathrm{DL},n}^{(k)}} \approx \log_2(1+\eta_{\mathrm{DL},n}^{(k)}) - \sqrt{\frac{V(\eta_{\mathrm{DL},n}^{(k)})}{m_{\mathrm{DL},n}^{(k)}}} Q^{-1}(\varepsilon_{\mathrm{DL},n}^{(k)}), \quad (2)$$

where $V(\eta) = \frac{\eta(\eta+2)}{(\eta+1)^2}\log_2^2 e$ and $Q(x) = \int_x^\infty \frac{1}{\sqrt{2\pi}} e^{-t^2/2} dt$. In order to address the communication reliability, we characterize the impact of the data size, blocklength and SNR on the error probability. From (2), the error probability $\varepsilon_{\mathrm{DL},n}^{(k)}$ with packet size $D_{\mathrm{DL},n}^{(k)}$, SNR $\eta_{\mathrm{DL},n}^{(k)}$ and blocklength $m_{\mathrm{DL},n}^{(k)}$ is given by

$$\varepsilon_{\mathrm{DL},n}^{(k)} = Q\left(\frac{\log_2(1+\eta_{\mathrm{DL},n}^{(k)}) - D_{\mathrm{DL},n}^{(k)}/m_{\mathrm{DL},n}^{(k)}}{\log_2 e\sqrt{(1-(1+\eta_{\mathrm{DL},n}^{(k)})^2)/m_{\mathrm{DL},n}^{(k)}}}\right). \quad (3)$$

C. Computation model in the MEC node

The processing time (in symbols) in MEC node for an offloaded task with data size D_{UL} is given by:

$$m_C = \left\lceil \frac{D_{\mathrm{UL}} L}{c f_0 t_s} \right\rceil, \quad (4)$$

where L denotes the application-dependent required CPU cycles per bit for computation [10], c is the number of allocated CPU cores for the task, f_0 is the CPU-cycle frequency of a single CPU core and the operation $\lceil\cdot\rceil$ is the ceiling function. We assume that f_0 is fixed for all CPU cores in our edge computing system. Note that in (4) we convert the computing time in seconds to the time in symbols by considering the time duration of a single symbol t_s and applying the ceiling function to make sure m_C is an integer.

Once the computation is completed, the data would be sent to the corresponding terminal. Since the frames are synchronized the transmission of the data in the downlink channels must be able to finish within the current frame. With this frame limitation, in order to minimize the error probability, the coding blocklength in downlink transmission is chosen as

$$m_{\mathrm{DL},n}^{(k)} := T_f - m_{\mathrm{C},n}^{(k)} - m_{\mathrm{UL},n}^{(k)}. \quad (5)$$

Note that the MEC node should avoid the situation in which the value of $m_{\mathrm{DL},n}^{(k)}$ is negative for given $m_{\mathrm{C},n}^{(k)}$ after the calculation according to (5).

III. Deep Reinforcement Learning Based Resource Allocation Policy

We consider that an agent in the MEC node interacts with the edge computing environment $\mathcal{E}$[1]. In each time slot the agent observes the environment and gets the following input from the environment $x^{(k)} = \{\mathbf{D}^{(k)}, \mathbf{W}^{(k)}, \mathbf{q}^{(k)}, \boldsymbol{\eta}^{(k)}\}$. The components of the input are described in detail below:

- $\mathbf{D}^{(k)} = \left[D_{\mathrm{UL},1}^{(k)}, D_{\mathrm{UL},2}^{(k)}, \ldots, D_{\mathrm{UL},N}^{(k)}\right]$: a vector of length N containing the data size of the tasks to be processed at the head of buffers.
- $\mathbf{W}^{(k)} = \left[W_1^{(k)}, W_2^{(k)}, \ldots, W_N^{(k)}\right]$: a vector of length N containing the waiting time of the tasks to be processed at the head of buffers.
- $\mathbf{q}^{(k)} = \left[q_1^{(k)}, q_2^{(k)}, \ldots, q_N^{(k)}\right]$: a vector of length N containing the queue length of the buffers.
- $\boldsymbol{\eta}^{(k)} = \left[\eta_{\mathrm{DL},1}^{(k)}, \eta_{\mathrm{DL},2}^{(k)}, \ldots, \eta_{\mathrm{DL},N}^{(k)}\right]$: a vector of length N containing the SNRs in downlinks.

Since the waiting operations of offloaded tasks have strong impact on the E2E delay and the channels are correlated across neighboring frames, we consider the information with W historical slots as the current state, i.e., $s^{(k)} = (x^{(k-W)}, x^{(k-W+1)}, \ldots, x^{(k)})$. For the given state $s^{(k)}$, the agent takes an action $a^{(k)}$ through the policy π, i.e., $a^{(k)} = \pi(s^{(k)})$, and the system's operation is described by the sequence $s^{(1)}, a^{(1)}, s^{(2)}, \ldots, a^{(k-1)}, s^{(k)}$, which formalize a Markov decision process.

A. Action, Reward and Punishment

In each time slot the agent chooses an action $a^{(k)}$ from the action space $\mathcal{A}$, which allocates all CPU cores to the offloaded tasks waiting at the head of each buffer. If a task is allocated with zero CPU core, it would wait for the next allocation.

After applying an action, the agent receives a reward or punishment from the environment. Note that an effective setting of the reward and punishment is important for the learning algorithm to achieve the desired goal. In this work the rules for an effective setting of the reward and punishment follow by experience and multitude of attempts. In each time slot if a processed and transmitted task meets the requirements on the error probability and delay limit, the task is considered to be a successfully completed task and the agent gets a reward of $+1$. However, if the task does not meet the requirement on the error probability, i.e., the actual error probability is larger than the required value, the task is considered as an unsuccessful task and the agent gets a reward of -1. In the worst case that the task waits too long that the delay limit is violated and it has to be dropped off buffer, the task is considered as timed out and the agent gets a punishment of -1.5.

B. Reinforcement learning for MDP

After each interaction with the environment in time slot k, the agent receives a reward $r^{(k)}$ and the goal is to maximize the total future discounted reward $R^{(k)}$:

$$R^{(k)} = r^{(k)} + \gamma r^{(k+1)} + \gamma^2 r^{(k+2)} + \cdots + \gamma^{(K-k)} r^{(K)} \quad (6)$$

$$= r^{(k)} + \gamma R^{(k+1)}, \quad (7)$$

where γ is the discount factor taking values between 0 and 1 and K is the end point of the process[2].

We define the Q-function $Q(s^{(k)}, a^{(k)})$ which represents the quality of a certain action $a^{(k)}$ in a given state $s^{(k)}$. The optimal Q-function should return the maximum expected achievable reward by following a policy π after receiving the state s and taking the action a, i.e., $Q^\star(s,a) = \max_\pi \mathbb{E}[R^{(k)} | s^{(k)} = s, a^{(k)} = a, \pi]$, which can be obtained

[1] Please note that this $\mathcal{E}$ is different from ε which represents the error probability in this work.

[2] K could be infinity in the case without an end point.

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}\left[\left(r^{(k)} + \gamma \max_{a^{(k+1)}} Q\left(s^{(k+1)}, a^{(k+1)}; \theta_{i-1}\right) - Q\left(s^{(k)}, a^{(k)}; \theta_i\right)\right) \nabla_{\theta_i} Q\left(s^{(k)}, a^{(k)}; \theta_i\right)\right] \tag{11}$$

as the solution of the Bellman equation [11]:

$$Q^{\star}(s^{(k)}, a^{(k)}) = \mathbb{E}_{s^{(k+1)}}\left[r^{(k)} + \gamma \max_{a^{(k+1)}} Q^{\star}(s^{(k+1)}, a^{(k+1)}) \middle| s^{(k)}, a^{(k)}\right]. \tag{8}$$

By using the Bellman equation as an iterative update, the Q-function is estimated as:

$$Q^{(k+1)}(s^{(k)}, a^{(k)}) = \mathbb{E}_{s^{(k+1)}}\left[r^{(k)} + \gamma \max_{a^{(k+1)}} Q^{(k)}(s^{(k+1)}, a^{(k+1)}) \middle| s^{(k)}, a^{(k)}\right]. \tag{9}$$

This value iteration algorithm is guaranteed to converge to the optimal point with $Q^{(k)} \to Q^{\star}$ as $k \to \infty$ [12]. However, it would take immeasurably long time for the traditional Q-learning with a Q-table to converge especially in the continuous state space. Therefore, it is common to use a neural network as a non-linear function approximation to estimate the Q-function, where the input and the output of the neural network is the state vector and the Q-value of each possible action.

C. Deep Q-learning

We define $Q(s^{(k)}, a^{(k)}; \theta)$ as the approximated Q-function, i.e., $Q(s^{(k)}, a^{(k)}; \theta) \approx Q^{\star}(s^{(k)}, a^{(k)})$, with the weight parameters θ of a neural network referred to as the Q-network [13]. The Q-network updates its parameters θ_i by minimizing a sequence of loss functions $L_i(\theta_i)$ at iteration i:

$$L_i(\theta_i) = \mathbb{E}\left[\left(y_i - Q(s^{(k)}, a^{(k)}; \theta_i)\right)^2\right], \tag{10}$$

where $y_i = \mathbb{E}[r^{(k)} + \gamma \max_{a^{(k+1)}} Q(s^{(k+1)}, a^{(k+1)}; \theta_{i-1}) | s^{(k)}, a^{(k)}]$ is the target for the iteration i. Note that the parameters of the Q-network from the previous update θ_{i-1} are held fixed when optimizing the loss function $L_i(\theta_i)$.

Differentiating the loss function with respect to the weight parameters θ_i results in the gradient shown in (11). Using the Stochastic Gradient Descent (SGD) method, the parameters of the Q-network is updated iteratively. It is known that using a non-linear function approximator for the Q function may result in an unstable learning process. Therefore we apply a mechanism to improve the performance, namely the experience replay mechanism [13], [14]. It is able to break the similarity of subsequent training samples which might drive the network into a local minimum. Specifically we store the transition at each time slot $e^{(k)} = (s^{(k)}, a^{(k)}, r^{(k)}, s^{(k+1)})$ in a memory pool $\mathcal{D} = \{e^{(1)}, \ldots, e^{(k)}\}$. At each training process a minibatch is sampled randomly from the memory pool and a SGD update is performed over it. Since the proposed deep Q-learning algorithm is an off-policy algorithm for which adequate exploration is necessary, we apply the widely used ϵ-greedy exploration strategy in which the agent chooses a random action with a probability ϵ and otherwise chooses the current optimal action with the highest Q-value. The full algorithm is shown in Algorithm 1.

Algorithm 1 Deep Q-learning based resource allocation scheme

1: Initialize the replay memory $\mathcal{D}$ to capacity M
2: Initialize Q-network with random weights
3: **for** $k = 1, 2, 3, \ldots$ **do**
4: Observe current system state $s^{(k)}$
5: Select random action $a^{(k)}$ with probability ϵ
6: Otherwise select $a^{(k)} = \text{argmax}_a Q(s^{(k)}, a^{(k)}; \theta)$
7: Execute $a^{(k)}$ and observe $r^{(k)}$ and $s^{(k+1)}$
8: Store transition $(s^{(k)}, a^{(k)}, r^{(k)}, s^{(k+1)})$ in $\mathcal{D}$
9: Sample random minibatch of transitions $(s^{(j)}, a^{(j)}, r^{(j)}, s^{(j+1)})$ from $\mathcal{D}$
10: Set $y^{(j)} = r^{(j)} + \gamma \max_{a^{(j+1)}} Q(s^{(j+1)}, a^{(j+1)}; \theta)$
11: Perform a SGD update on θ according to (11)
12: **end for**

IV. Simulation Results

In the simulation, we study a simple 2-user-scenario with an MEC node equipped with a 3-core-CPU where the CPU-cycle frequency of each core f_0 is 3×10^9 cycles per second. The frame length T_f is 600 symbols where the time of one symbol t_s is $4.5\mu s$. The blocklength of uplinks $m^{(k)}_{\text{UL},n}$ are all assumed to be equal to 200 symbols. The packet size in uplinks $D^{(k)}_{\text{UL},n}$ is chosen from the set $\{500, 1000, 1500, 2000\}$ and the coefficient β is 0.5. The required CPU cycles per bit L is 2640. The average SNR of downlink channels $\bar{\eta}_{\text{DL}}$ is 25dB and the channel correlation coefficient ρ_{DL} is 0.7. The E2E delay limit $d_r = \alpha T_f$ is with $\alpha = 3$. The required decoding error probability $\varepsilon^{\star}_{\text{DL}}$ is 10^{-4}. The parameters of the reinforcement learning algorithm are as follows: the number of historical time slots W is 3, ϵ is set to vary from 1 to 0.1 with a decay rate of 0.999, the discount factor γ is 0.95. The values of all parameters are listed in Table I.

We evaluate the proposed deep reinforcement learning based algorithm by considering the success ratio in downlink under different task arrival levels. The successful tasks must not violate the E2E delay limit and meet the required decoding error probability. We compare the proposed deep Q-learning algorithm with two benchmarks: equal and random allocation strategy. The equal allocation strategy always allocates the total CPU cores to all current tasks equally. If the number of CPU cores is not divisible by the task number, the tasks with comparatively worse channel quality receive the remainder of the CPU cores. We node that under the equal allocation strategy the tasks would not wait at the buffers. On the other hand, the random allocation strategy takes a random allocation action from the action space $\mathcal{A}$ in each time slot. In this case the tasks have the possibility to wait at the buffers.

In Fig. 2, the learning curve of the proposed Q-network with the task arrival probability of 0.4 is depicted. The Q-network is tested under 100000 frames after each episode where one episode contains 500 frames. Note that during the testing the algorithm runs without exploration, i.e., $\epsilon = 0$. It is observed that at the beginning of learning process the performance is in general worse than the benchmark of the random strategy and significant fluctuations are exhibited.

Table I. SIMULATION PARAMETERS

Parameter	Value	Parameter	Value
t_s	$4.5\mu s$	γ	0.95
T_f	600 symbols	ϵ	1 to 0.1
m_{UL}	200 symbols	W	3
L	2640 cycles/bit	ρ_{DL}	0.7
f_0	3×10^9 cycles/s	$\varepsilon^{\star}_{\text{DL}}$	10^{-4}
$\bar{\eta}_{\text{DL}}$	25 dB	α	3
$D^{(k)}_{\text{UL},n}$	$\{0.5,1,1.5,2\}\cdot 10^3$ bit	β	0.5

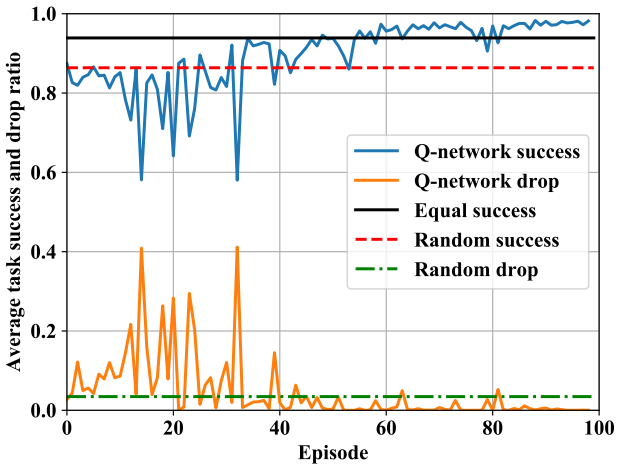

Figure 2. Dynamic performance of the proposed Deep Q-learning algorithm with task arrival probability of 0.4 over the episodes, where each episode contains 500 time slots. The benchmarks of the equal and random allocation strategy are shown for the evaluation.

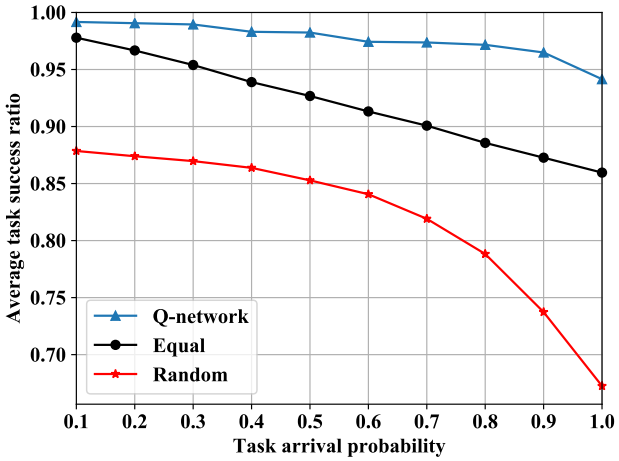

Figure 3. Average task success ratio of the proposed Deep Q-learning algorithm over different task arrival probability compared with the benchmarks of the equal and random allocation strategy.

This is because that the learning in this period is mainly based on the random exploration with a large ϵ. After roughly 40 episodes the performance of the proposed algorithm exceeds the benchmark of the random strategy, and after roughly 60 episodes it outperforms the benchmark of the equal strategy. Finally the algorithm converges to an optimum after almost 100 episodes.

In Fig. 3, the resulting average task success ratio under different task arrival level is depicted. It is observed that the proposed algorithm provides an outstanding performance over a wide range of the task arrival levels. Under low task arrival levels the average task success ratio of the proposed algorithm is around 0.99. While the performances of the equal and random strategies decrease linearly and exponentially respectively, with the increasing task arrival probability, the performance of the proposed algorithm diminishes only slightly and is kept larger than 0.95 for almost all levels. This result shows benefits from the smart waiting decisions for the tasks to avoid the bad channel qualities.

V. CONCLUSION

In this paper, we have developed a deep reinforcement learning based intelligent agent for computational resource allocation in mobile edge computing networks with multiple users aiming to support the demands of URLLC. The Q-network based agent directly takes the current state as the input and learns the best strategy through interactions with the environment without requiring the knowledge of the system model. The simulation results show that the proposed deep Q-learning algorithm converges to an optimum and outperforms the equal and random scheduling benchmarks under a wide range of task arrival levels.

REFERENCES

[1] 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*, pp. 13–16, 2012.

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

[3] W. Yu, F. Liang, X. He, W. G. Hatcher, C. Lu, J. Lin, and X. Yang, "A survey on the edge computing for the internet of things," *IEEE Access*, vol. 6, pp. 6900–6919, 2018.

[4] Y. Hu, M. C. Gursoy, and A. Schmeink, "Relaying-enabled ultra-reliable low-latency communications in 5G," *IEEE Network*, vol. 32, no. 2, pp. 62–68, 2018.

[5] Y. Polyanskiy, H. V. Poor, and S. Verdú, "Channel coding rate in the finite blocklength regime," *IEEE Transactions on Information Theory*, vol. 56, no. 5, pp. 2307–2359, 2010.

[6] C. Liu, M. Bennis, and H. V. Poor, "Latency and reliability-aware task offloading and resource allocation for mobile edge computing," *arXiv preprint arXiv:1710.00590*, 2017.

[7] Y. Mao, J. Zhang, S. Song, and K. B. Letaief, "Power-delay tradeoff in multi-user mobile-edge computing systems," in *IEEE Global Communications Conference*, pp. 1–6, 2016.

[8] M. Ozmen and M. C. Gursoy, "Wireless throughput and energy efficiency with random arrivals and statistical queuing constraints," *IEEE Transactions on Information Theory*, vol. 62, no. 3, pp. 1375–1395, 2016.

[9] Y. Hu, A. Schmeink, and J. Gross, "Optimal scheduling of reliability-constrained relaying system under outdated CSI in the finite block-length regime," *IEEE Transactions on Vehicular Technology*, 2018.

[10] J. Kwak, Y. Kim, J. Lee, and S. Chong, "DREAM: Dynamic resource and task allocation for energy minimization in mobile cloud systems," *IEEE Journal on Selected Areas in Communications*, vol. 33, no. 12, pp. 2510–2523, 2015.

[11] M. L. Puterman, *Markov decision processes: discrete stochastic dynamic programming*, 2014.

[12] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*, vol. 1, no. 1, 1998.

[13] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[14] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.