2024 International Conference on Intelligent Computing, Communication, Networking and Services (ICCNS)

COOLER: Cooperative Computation Offloading in Edge-Cloud Continuum under Latency Constraints via Multi-agent Deep Reinforcement Learning

Anastasios Giannopoulos
Dept. of Ports Management and Shipping
National and Kapodistrian University of Athens
Psachna, Greece
angianno@uoa.gr

Ilias Paralikas
Research & Development Department
Four Dot Infinity
Athens, Greece
paralikasilias@gmail.com

Sotirios Spantideas
Dept. of Ports Management and Shipping
National and Kapodistrian University of Athens
Psachna, Greece
sospanti@uoa.gr

Panagiotis Trakadas
Dept. of Ports Management and Shipping
National and Kapodistrian University of Athens
Psachna, Greece
ptrakadas@pms.uoa.gr

***Abstract*—In the burgeoning domain of the edge-cloud continuum (ECC), the efficient management of computational tasks offloaded from mobile devices to edge nodes is paramount. This paper introduces a Cooperative cOmputation Offloading scheme for ECC via Latency-aware multi-agent Reinforcement learning (COOLER), a distributed framework designed to address the challenges posed by the uncertain load dynamics at edge nodes. COOLER enables each edge node to autonomously make offloading decisions, optimizing for non-divisible, delay-sensitive tasks without prior knowledge of other nodes' task models and decisions. By formulating a multi-agent computation offloading problem, COOLER aims to minimize the expected long-term latency and task drop ratio. Following the ECC requirements for seamless task flow both within Edge layer and between Edge-Cloud layers, COOLER considers that task computation decisions are three-fold: (i) local computation, (ii) horizontal offloading to another edge node, or (iii) vertical offloading to the Cloud. The integration of advanced techniques such as long short-term memory (LSTM), double deep Q-network (DQN) and dueling DQN enhances the estimation of long-term costs, thereby improving decision-making efficacy. Simulation results demonstrate that COOLER significantly outperforms baseline offloading algorithms, reducing both the ratio of dropped tasks and average delay, and better harnessing the processing capacities of edge nodes.**

***Index Terms*—6G network, deep reinforcement learning, edge computing, edge-cloud continuum, resource management, task offloading**

I. Introduction

A. The Edge-Cloud Continuum paradigm

The advent of the Edge-Cloud Continuum (ECC) marks a transformative era in distributed computing [1], where the seamless integration of edge computing and cloud services fosters a new paradigm of ubiquitous, low-latency, and scalable computing [2]. The primary goal of ECC is to leverage the proximity of edge nodes to end-users, thereby reducing latency, conserving bandwidth, and enabling real-time data processing and analytics [3]. Key elements of the ECC architecture include edge nodes, cloud data centers, and the orchestration layer that manages resources and task distribution.

As we stand on the cusp of the 6G communications era, expectations are set for unprecedented advancements in network capabilities, catering to the high demands of time-sensitive applications and managing the deluge of traffic from a burgeoning number of IoT devices [4]. The envisioned 6G network is anticipated to be characterized by ultra-reliable low-latency communication (URLLC), enhanced mobile broadband (eMBB), and massive machine-type communications (mMTC) across ground, air, and sea [5]. In this context, the ECC paradigm is poised to play a pivotal role, acting as a dynamic and adaptive framework capable of meeting these diverse and intensive computational needs. By intelligently offloading tasks from the IoT layer to the most appropriate computational resources, ECC will be instrumental in fulfilling the latency-sensitive and high-throughput requirements of future 6G networks, ensuring seamless connectivity and optimal performance.

B. Task Offloading in ECC

In this 6G-enabled ECC architecture, task offloading emerges as a critical mechanism, allowing tasks to be dynamically allocated to the most suitable computing layer [6]. This not only optimizes resource utilization but also ensures that computational demands are met with minimal delay, which is paramount for time-sensitive applications. Thus, task

This work was supported in part by the ICOS (Towards a functional continuum operating system) Project funded from the European Union's HORIZON research and innovation programme under grant agreement No 101070177 (https://www.icos-project.eu/).

offloading stands at the core of ECC, embodying its goals of efficiency and responsiveness. Moreover, in the ECC era, task offloading transcends the confines of traditional vertical architectures, embracing a hybrid model that facilitates both vertical and horizontal communication. This shift enables tasks to flow not only from one layer to another, such as from the Edge to the Cloud, but also laterally among Edge nodes themselves. Such a versatile approach allows for a more fluid distribution of computational loads, optimizing the use of available resources and enhancing system resilience. It reflects the evolving requirements of ECC, where the ability to make swift, decentralized offloading decisions becomes crucial for maintaining seamless operations across the network's various layers.

C. Related Work

Task offloading in the domains of mobile edge and edge-cloud computing has been the focus of numerous studies [7]. Dinh et al. [8], [9] tackled the problem by proposing an optimization framework for offloading from a single mobile device to multiple edge devices, aiming to minimize both execution latency and energy consumption. Li et al. [10] addressed the deployment of mobile edge servers using a clustering approach, significantly reducing average completion time, power consumption, and overhead in edge server deployment issues. Ullah et al. [11] employed Deep Reinforcement Learning (DRL) to optimize offloading and resource allocation in edge-cloud networks, demonstrating improved resource utilization and task offloading. Liu et al. [12] proposed a fast and efficient task offloading approach in edge-cloud collaboration environments, achieving a near-optimal solution with low time overhead. Lastly, Shu et al. [13] designed a multi-user task offloading algorithm that supports dividing tasks into subtasks and offloading them to edge servers to reduce end-to-end task execution time.

D. Paper Outline and Contributions

The above works collectively underscore the importance of efficient task offloading mechanisms and highlight the benefits of leveraging computational resources at the edge. However, they often rely on the assumption of known task models and offloading decisions of other nodes, as well as they consider the conventional vertical task flow, where each computing task can be vertically transferred across the system layer. In this work, we propose a Cooperative cOmputation Offloading scheme for ECC via Latency-aware multi-agent Reinforcement learning (COOLER). By considering a multi-server ECC system, COOLER scheme aims to address these limitations by enabling decentralized offloading decisions without the knowledge of task models and others' decisions, thereby resolving some of the common constraints identified in the studies. Following COOLER suggestions, each computing node employs a double and dueling DRL model and targets to make proper offloading decisions so as to minimize the task latency and eliminate the task drop probability.

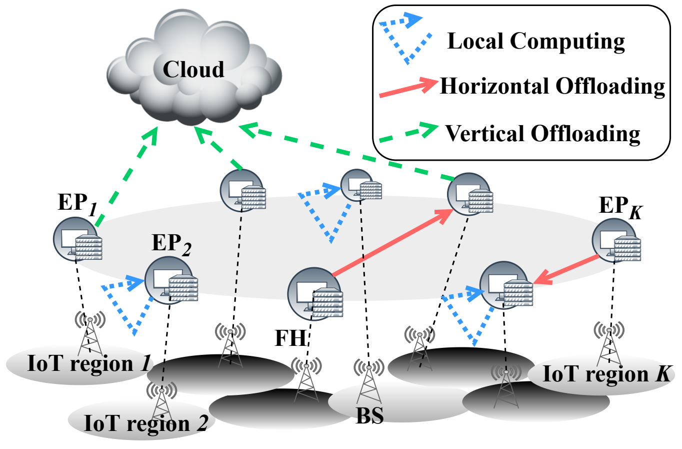

Fig. 1. An IoT-Edge-Cloud Continuum system, where IoT regions generate tasks to be executed by the Edge Points (EPs). Each EP can locally compute the task or offload it horizontally (or vertically) to another EP (or Cloud).

The key contributions of this work may be summarized as follows:

- Decentralized and model-free decision-making: The proposed scheme enables edge nodes to autonomously make offloading decisions, fostering a decentralized approach that aligns with the dynamic nature of ECC environments. Also, it does not require extensive knowledge of neither of task models, nor global observability at the single-server side of the others' decisions.
- Hybrid offloading capability: The COOLER model supports both vertical and horizontal offloading decisions, adhering to the principles of the ECC by allowing task flow between- and within- layers of the multi-server architecture.
- Integration of LSTM/DRL Techniques: LSTM model, double and dueling DQN techniques are combined to improve the estimation of long-term costs, enhancing the offloading decision-making process.
- Numerical validation: Simulation results validate that COOLER outperforms multiple baseline algorithms, reducing the ratio of dropped tasks and average delay, thus optimizing the use of processing resources and response task of ECC.

II. System Model

In this section, all the modelling components for task offloading in the ECC are described, including the system architecture, the task features, the decision process and the queues that are considered for both computation and forwarding of the tasks.

A. System Architecture of ECC

In the proposed IoT-Edge-Cloud network model, depicted in Fig. 1, we consider a structure with K Edge Points (EPs) and a single Cloud entity to manage multiple IoT regions. The Cloud is indexed by $K+1$, and the EPs identifiers are drawn from the set $\mathcal{K} = \{1, 2, \ldots, K\}$. Each EP $k \in \mathcal{K}$ is tasked with processing all applications from its IoT region, with tasks being non-divisible and requiring complete computation by either an EP or the Cloud. The burgeoning 6G network traffic necessitates an adept policy for task computations.

EPs can compute tasks locally, offload them horizontally to another EP, or vertically to the Cloud, with each task having a defined timeout period. Decentralized decision-making is facilitated by a DRL model [14] within each EP, which processes task features and traffic data to minimize the Task Computation Latency (TCL) and the Task Throw Rate (TTR). The network includes M Edge Monitoring Units (EMUs) for monitoring, with each EMU being in charge of monitoring a cluster of EPs ($M < K$). EMUs also manage data transfers between same-cluster EPs and enable inter-EMU communication for data sharing across EP clusters. Without loss of generality, a single-cluster network ($M = 1$) with multiple EPs is assumed for the rest of the paper.

We focus on an episode encompassing time slots $\mathcal{T} = \{1, 2, \ldots, T\}$, each with a duration Δ. Communication is maintained through wireless links between IoT devices and base stations, and wired fronthaul (FH) links connect base stations to their respective EPs, with all EPs linked to the Cloud via the Internet.

Fig. 2 illustrates the queues located in EPs. Each time slot begins with a new task arrival at EP k with probability $\mathcal{P}$. EPs have dual functionality, acting as local computing nodes or hosting offloaded tasks. An EP $k \in \mathcal{K}$ has K FIFO computation queues and one forwarding queue (FQ). The k^{th} computation queue of EP k is the internal queue (IQ) for local tasks, while the remaining $K-1$ external queues (EQs) host tasks from other EPs. For example, a task offloaded from EP j to EP k is placed in j^{th} external queue of EP k. The Cloud has K external queues for tasks offloaded by each EP. Upon task completion, the next task is processed or offloaded in the subsequent time slot.

B. Tasks and Decision Process

In the proposed ECC model, each EP $k \in \mathcal{K}$ is assigned a unique task identifier $u_k(t) \in \mathbb{Z}_+$ for tasks arriving at time $t \in \mathcal{T}$. Task arrival is indicated by a binary variable $x_k(t)$, where $x_k(t) = 1$ if a new task arrives. The task ID and size, $\eta_k(t)$, are set to zero if no task arrives. Task sizes are drawn from a set $\mathcal{H}$, and each task $u_k(t)$ is associated with a processing density $\rho_k(t)$ (in CPU cycles per bit) and a timeout index ϕ_k (in time slots).

The offloading decision process involves two decision maker (DM) modules: DM$^{(1)}$ determines whether a task is computed locally or offloaded, and DM$^{(2)}$ decides the offloading destination. The decisions are represented by binary variables $d_k^{(1)}(t)$ and $d_{k,n}^{(2)}(t)$, where $d_k^{(1)}(t) = 1$ indicates local computation, and $d_{k,n}^{(2)}(t) = 1$ signifies offloading from EP k to node n (another EP or Cloud). A task from EP k can be offloaded to only one destination, as shown by $\sum_{n \neq k} d_{k,n}^{(2)}(t) \leq 1$. Finally, a decision tuple $\mathbf{D}_k(t) = \left(d_{k,n}^{(2)}, n\right)$ encapsulates the computing destination of task $u_k(t)$, where n is the node ID to which the task is offloaded. For instance, if $u_j(t)$ is offloaded to the Cloud, then $\mathbf{D}_j(t) = \left(1, K+1\right)$.

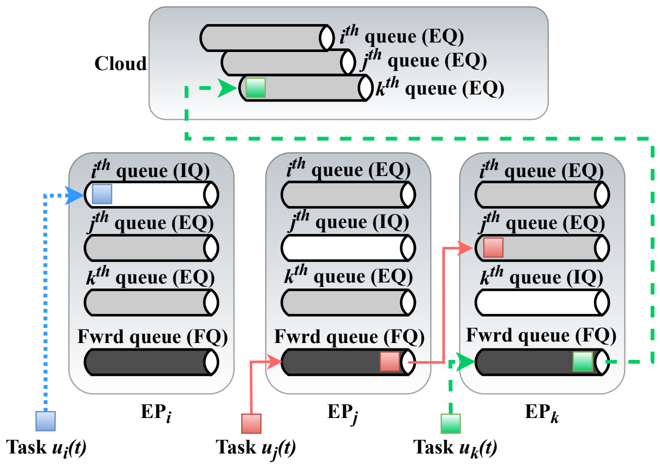

Fig. 2. The queues inside Edge Points (EPs) and Cloud. Three examples of the computation path followed by a task that is decided to (i) be executed locally at EP i (blue path), (ii) be offloaded horizontally from EP j to EP k (red path), (iii) be offloaded vertically from EP k to Cloud (green path).

C. Computation and Forwarding Queues

1) Internal Queue: The tasks designated for local processing are queued in the FIFO IQ of their respective EP. The processor unit (CPU) of each EP k, with a fixed capacity of f_k^{IQ} Hz, processes these tasks. The completion time slot $\psi_k^{IQ}(t)$ is determined for each task $u_k(t)$, with $\psi_k^{IQ}(t) = 0$ if no task is queued at time t. The waiting time $w_k^{IQ}(t)$ is the duration a task remains in the queue before processing or being discarded due to timeout.

The waiting time $w_k^{IQ}(t)$ is calculated as the maximum completion time of previous tasks, ensuring it's non-negative, as in the next formula:

$$w_k^{IQ}(t) = \max\left\{0, \max_{t'<t}\{\psi_k^{IQ}(t')\} - t + 1\right\} \tag{1}$$

The completion time slot $\psi_k^{IQ}(t)$ is the minimum of the time slot when processing finishes or when the task is dropped due to timeout, as expressed in the equation:

$$\begin{aligned}\psi_k^{IQ}(t) = \min\Big\{ & t + w_k^{IQ}(t) + \left\lceil \frac{\eta_k(t) \cdot \rho_k(t)}{f_k^{IQ} \cdot \Delta} \right\rceil - 1, \\ & t + \phi_k(t) - 1 \Big\}\end{aligned} \tag{2}$$

Specifically, the processing start time is $t + w_k^{IQ}(t)$, and the duration is the ceiling of the task size times processing density over the CPU capacity and time slot duration, or until the task's timeout.

2) External Queues: Each EP has $K-1$ EQs for processing external tasks from the rest EPs. The Cloud has K EQs to process tasks from all EPs. Thus, the i^{th} EQ i at a node is the designated offloading destination for tasks from EP i. Tasks offloaded by EP k and arriving at node n at time t are queued in the k^{th} EQ at $t+1$. Each task receives a unique ID $u_{k,n}(t)$ upon queuing, where t is the time slot of queuing (i.e. $u_{k,n}(t) = u_k(t' < t)$.

The size of the task (in bits), $\eta_{k,n}(t)$, and the length of the EQ, $l_{k,n}(t)$, are also defined. An EQ k at node n is active at time t if it receives a new task or has tasks from the previous slot. The set of active EQs in node n at time t is $\mathcal{A}_n(t)$ and contains the indices of the EQs that are non-empty.

Each EP k has another CPU f_k^{EQ} for processing external tasks, and the Cloud's CPU is f^{Cloud}, with $f^{Cloud} > f_k^{EQ}$, $\forall k \in \mathcal{K}$. The CPU capacity dedicated for external tasks is equally distributed among active EQs. The length of the k^{th} EQ at node n is updated as:

$$l_{k,n}(t) = \max\Big\{0, l_{k,n}(t-1) + \eta_{k,n}(t) - \\ -m_{k,n}(t) - \frac{\Delta \cdot f_k^{EQ}}{\rho_k(t)\cdot|\mathcal{A}_n(t)|}\Big\}, \tag{3}$$

where $m_{k,n}(t)$ is the number of bits dropped by the EQ k of node n at the end of time slot t. Note that, we assumed equal CPU distribution for fairness purposes, but the model can be modified to prioritize amongst specific EPs. The processing capacity depends on the number of active queues and is not predetermined, but we assumed that EPs are aware of each other's external CPU capacities (i.e. CPU capacity of all nodes is globalized).

3) Forwarding Queue: Each EP has a forwarding queue operating on a FIFO basis, which stacks the tasks for offloading. When EP k selects a task $u_k(t)$ for offloading, FQ is connected to the destination EQ of another EP or Cloud via a wired link. The data rates for EP-to-EP (R_H) and EP-to-Cloud (R_V) communications are constant, with $R_H > R_V$.

The waiting time $w_k^{FQ}(t)$ and completion time slot $\psi_k^{FQ}(t)$ for a task in the FQ are defined similarly to the IQ case. The completion time slot marks when a task is sent or dropped. The waiting time in the FQ is given by:

$$w_k^{FQ}(t) = \max\left\{0, \max_{t'<t}\{\psi_k^{FQ}(t')\} - t + 1\right\} \tag{4}$$

This reflects the time slots a task waits in the FQ, calculated before deciding the queue placement for $u_k(t)$. The completion time slot $\psi_k^{FQ}(t)$ for a task placed in FQ of EP k is computed using:

$$\psi_k^{FQ}(t) = \min\Big\{t + w_k^{FQ}(t) + \\ + \Big\lceil \sum_{n\neq k} \frac{d_{k,n}^{(2)}(t)\cdot\eta_k(t)}{R_{k,n}\cdot\Delta}\Big\rceil - 1, t + \phi_k(t) - 1\Big\} \tag{5}$$

Here, $d_{k,n}^{(2)} = 1$ if EP k offloads to node n (EP or Cloud). The completion time slot is the minimum between the time slot for successful offloading and the task timeout. The offloading time is the sum of the arrival time slot t, waiting time $w_k^{FQ}(t)$, and the time to forward the task. The data rate for offloading the task is $R_{k,n} \in \{R_H, R_V\}$.

III. Hybrid Task Offloading in ECC

In this section, we formulate the optimization decision-making problem for multi-agent decentralized task offloading in a multi-server ECC system. Next, we outline the proposed COOLER scheme for finding a sub-optimal solution to this problem.

A. Problem Formulation

Each EP agent must judiciously choose to process tasks locally, offload them to another peer EP agent, or send them up to the Cloud. The crux of this problem lies in devising a cost function that encapsulates the agents' objective: to make offloading decisions that deftly minimize the long-term computation latency (TCL) and the probability of task drops (TTR), thereby ensuring efficient and reliable task handling in a dynamic computational landscape.

At a given time slot $t \in \mathcal{T}$, the global system state is described by the set $\mathcal{S} = \{\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_K\}$, where $\mathcal{S}_k \in \mathcal{S}$ is the local state of EP $k \in \mathcal{K}$. At a given episode, each EP passes through a series of states $\{\mathbf{s}_k(1), \mathbf{s}_k(2), \ldots, \mathbf{s}_k(T)\}$. For a given time slot t, each EP k observes the local environment state $\mathbf{s}_k(t)$ and takes an action $\mathbf{a}_k(t)$. The action $\mathbf{a}_k(t)$ causes a new state $\mathbf{s}_k(t+1)$, whereas a scalar reward $r_k(t+1)$ that reflects whether the action was beneficial or not. In specific, the state, action and reward are defined as follows:

1) State: The state of EP k at time slot t is given by:

$$\mathbf{s}_k(t) = \Big[\eta_k(t), w_k^{IQ}(t), w_k^{FQ}(t), \mathbf{l}_k^{EQ}(t-1), \mathbf{L}(t)\Big] \tag{6}$$

where $\mathbf{l}_k^{EQ}(t-1)$ is a row vector that contains the length of the EQs hosting tasks of EP k at the end of $t-1$. Also, $\mathbf{L}(t)$ is a matrix of size $W \times (K+1)$ that records the W (lookback window) previous load values of each computing node (EPs and Cloud). As load values, we considered the number of active queues.

2) Action: From a given $\mathbf{s}_k(t)$, EP k chooses an action as:

$$\mathbf{a}_k(t) = \Big[d_k^{(1)}(t), \mathbf{D}_k(t)\Big] \tag{7}$$

Evidently from (7), the decision is two-step, meaning that it is, firstly, decided whether to offload or not and, secondly, the offloading destination.

3) Reward: Upon taking an action $\mathbf{a}_k(t)$ from state $\mathbf{s}_k(t)$, the reward that is received is given by:

$$r_k(t) = \begin{cases} 0, & \text{No task arrived} \\ \psi_k^{IQ}(t) - t + 1, & \text{Local processing} \\ \sum\limits_{k \neq n} d_{k,n}^{(2)}(t)\Big(\psi_{k,n}(\tau) - t + 1\Big), & \text{Offloading} \\ C, & \text{Task thrown} \end{cases} \tag{8}$$

It is evident from (8) that the reward function is four-fold, with each EP receiving (i) zero reward if no task arrived, (ii) a positive value of the delay for processing the task locally, (iii) a positive value of the delay for processing the task in another node, or (iv) a high-valued constant C if the task was thrown. Note that $\psi_{k,n}(\tau)$ is the completion time slot of the offloaded task $u_k(t)$, which arrived at the destination EQ k of node $n \in \mathcal{K} \cup \{K+1\}$ at time $\tau > t$.

Hence, letting π_k denote the policy (i.e. mapping from states to actions) learned by the DRL agent $k \in \mathcal{K}$, the following optimization problem implies that each DRL agent k targets an optimal policy π_k^* for minimizing the expected cumulative (in time) reward:

$$\begin{aligned} \pi_k^* = \arg\min_{\pi_k} \mathbb{E}\Big\{\sum_{t \in \mathcal{T}} \gamma^{t-1} \cdot r_k(t) \Big| \pi_k \Big\} \\ \textit{subject to: } \text{constraints}(1) - (5) \end{aligned} \tag{9}$$

where $\mathbb{E}\{\cdot\}$ is the expectation over the random and dynamic task arrivals or characteristics, and the others' decisions. Also, $\gamma \in (0, 1]$ is the discount factor which scales the future rewards [15].

B. COOLER Scheme Solution

The COOLER scheme aims to facilitate decentralized task offloading in the ECC, guaranteeing that the offloading decisions jointly minimize the long-term cost (TCL and TTR). COOLER considers that EPs act independently, assessing the system's state to optimize task handling, aiming to reduce latency (TCL) and task loss (TTR) simultaneously. Note that latency is usually correlated with the power consumption required for executing a task and, thus, COOLER partly reduces the system's power consumption.

According to the COOLER scheme, each EP k employs a DQN which is trained based on the principles of double and dueling Q-learning [16]. At a given time slot t, the input of the DQN model k (with parameters θ_k) is the state $\mathbf{s}_k(t)$ and the output is the Q-value of all possible actions $\mathbf{a} \in \{0, 1\}^{K+1}$. The Q-value of action $\mathbf{a}$ of agent k are updated as:

$$\begin{aligned} Q_k\Big(\mathbf{s}_k(t), \mathbf{a}\Big|\theta_k\Big) = V_k\Big(\mathbf{s}_k(t)\Big|\theta_k\Big) + \Big[A_k\Big(\mathbf{s}_k(t), \mathbf{a}\Big|\theta_k\Big) \\ - \frac{1}{2^{K+1}} \sum_{\mathbf{a}' \in \{0,1\}^{K+1}} A_k\Big(\mathbf{s}_k(t), \mathbf{a}'\Big|\theta_k\Big)\Big] \end{aligned} \tag{10}$$

where the Q-value is computed as the sum of the state-value $V(\cdot)$ and the action-advantage value $A(\cdot)$, with the latter being relative to the mean action-advantage value across all possible actions. Details on dueling Q-learning can be found in [16].

Algorithm 1 DDDQN Training for COOLER agent k

```
1: Initialize replay memory $D$ to capacity $N_R$
2: Initialize action-value function $Q$ with random weights $\theta$
3: Initialize target action-value function $\hat{Q}$ with weights $\hat{\theta}$
4: for episode $= 1$ to $N_E$ do
5:     Reset the environment and empty all queues
6:     for $t = 1$ to $T$ do
7:         With probability $\epsilon$ select a random action $a_t$
8:         otherwise select $a_t = \arg\max_a Q(s_t, a; \theta)$
9:         Execute action $a_t$ and observe $r_{t+1}$ and $s_{t+1}$
10:        Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in $D$
11:        Sample random minibatch of transitions
           $(s_j, a_j, r_{j+1}, s_{j+1})$ from $D$
12:        Set $y_j = r_{j+1}$ if episode terminates at step $j+1$
           else $y_j = r_{j+1} + \gamma\hat{Q}(s_{j+1}, \arg\max_{a'} Q(s_{j+1}, a'); \hat{\theta})$
13:        Perform a gradient descent step on
           $(y_j - Q(s_j, a_j; \theta))^2$ with respect to $\theta$
14:        Every $C_{clone}$ steps reset $\hat{Q} = Q$
15:    end for
16: end for
17: Output: Optimal policy $\pi_k^*$ for action-value function $Q$
```

Algorithm 1 outlines training a single-agent of COOLER scheme using experience replay and double and dueling DQN (DDDQN), where state transitions are stored as tuples $\Big(\mathbf{s}_k(t), \mathbf{a}_k(t), r_k(t+1), \mathbf{s}_k(t+1)\Big)$ in a memory of N_R entries. Over N_E episodes and T slots, the agent uses two neural networks: the action-selecting Q-model with θ_k, and the reward-estimating Target Q-model with $\hat{\theta}_k$, which stabilizes learning by providing consistent targets for Q-value predictions. The Target Q-model's $\hat{\theta}_n$ updates less frequently to maintain stability (every C_{clone} episodes, Q-model weights are cloned to target Q-model).

IV. Numerical Results

In this section, we numerically evaluate the performance of the COOLER scheme in an ECC system. We first present the training dynamics and hyperparameter stabilization of the DRL agents, followed by a comparative analysis with existing baseline methods.

A. Training the DRL Agents

For the efficient implementation of COOLER, various system parameters were set, with a specific focus on fine-tuning hyperparameters critical to optimizing the multi-agent DRL scheme. Key among these was the learning rate a, tested at values $a = [10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}]$. The selection of the learning rate was instrumental in balancing the speed of convergence against the stability of the learning process. For fine-tuning the vale of a, we considered the system parameters tabulated in Table I.

Fig. 3 illustrates the learning curve of the COOLER scheme for these different learning rates. Performance metrics such as Task Completion Latency (TCL) and Task Throw Rate

TABLE I
SYSTEM AND LEARNING PARAMETERS

Parameter	Symbol	Value
Task Arrival Probability	$\mathcal{P}$	0.4
Horizontal Data Rate	R_H	10 Mbps
Vertical Data Rate	R_V	20 Mbps
Task size	$\eta_k(t)$	$2-5$ bits
Task deadline	ϕ_k	10 time slots
Task processing density	$\rho_k(t)$	0.297 cycles/bit
Number of EPs	K	3
CPU frequency in EP's IQ	f_k^{IQ}	2.5 GHz
CPU frequency in EP's EQs	f_k^{EQ}	5 GHz
CPU frequency in Cloud	f^{Cloud}	30 GHz
Number of Training Episodes	N_E	12000
Number of Time slots	T	100
Time slot duration	Δ	0.1 sec
Discount factor	γ	0.8
Q-network hidden layers	N_L	3×20 neurons
Optimizer	Opt	Adam
Loss function	L	MSE
Update frequency	C_{clone}	2000 iterations
LSTM lookback window	W	10 steps
LSTM hidden layers	N_L	1×20 neurons
Replay Memory size	N_R	10000 samples
Task Drop Penalty	C	40
Batch size	B	64 samples

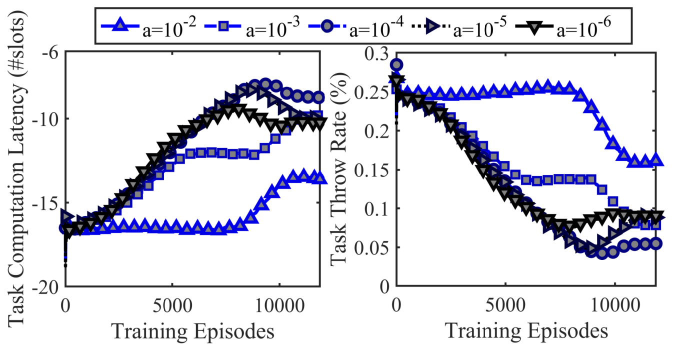

Fig. 3. Task Computation Latency (left) and Task Throw Rate (right) as a function of the training episodes for different learning rates a.

(TTR) are plotted, providing insights into how the learning rate impacts the efficacy and efficiency of task offloading decisions made by the agents. TCL has been considered negative, thus the ideal value of TCL is zero. This is the reason why TCL curve is increasing.

Notably, a learning rate $a = 0.0001$ emerged as optimal, achieving a balanced compromise between rapid learning and minimal overshoot, thereby minimizing both TCL and TTR. As expected, TCL and TTR show anti-symmetrical pattern, which means that maximizing TCL corresponds to minimizing TTR, and vice verca. On average, COOLER results in 8-slot latency for completing the tasks under this system setting, which corresponds to around 5% TTR.

To investigate how the scalability of the network and the intensity of task traffic influence the performance of the COOLER scheme, Fig. 4 depicts the relationship between TCL

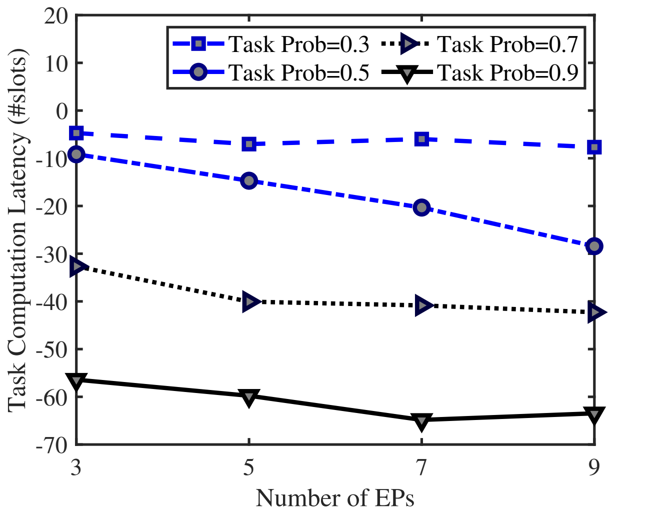

Fig. 4. Task Computation Latency (TCL) as a function of the number of EPs (K) for different task probabilities $\mathcal{P}$.

and the number of EPs, alongside varying levels of task arrival probability $\mathcal{P} = 0.3, 0.5, 0.7, 0.9$.

As depicted, the TCL deteriorates slightly as the number of EPs increases. This trend suggests that while adding more EPs potentially increases the computational capacity of the network, it also introduces greater complexity in coordination and data transmission among the agents. This complexity likely results in quite higher delays, particularly as the distance that data must travel increases, thereby exacerbating the computation latency.

Moreover, we observed a direct correlation between higher task arrival probabilities and increased delays. At lower probabilities, the system manages tasks more efficiently, likely due to lower demands on resource allocation and less frequent decision-making requirements for the DRL agents. However, as the task arrival probability escalates, the system becomes increasingly burdened. This burden is manifested in slower task execution times, highlighting a critical challenge in managing high volumes of data traffic within the ECC.

B. Comparison with baselines

To evaluate the efficacy of the COOLER scheme, this subsection compares its performance against five baseline offloading strategies. These comparisons help underscore the benefits of the proposed multi-agent DRL approach in managing task offloading within an ECC system. For the following simulations, we set the best hyperparameters of Table I (Section IV-A) considering $K = 9$ DRL agents (EPs).

For comparison purposes, we considered the following baselines:

1) **Random:** Each EP offloads tasks randomly, choosing to execute locally, vertically to the cloud, or horizontally to another EP, each with a probability $1/3$. If horizontally offloaded, the destination EP is selected randomly from the remaining EPs.

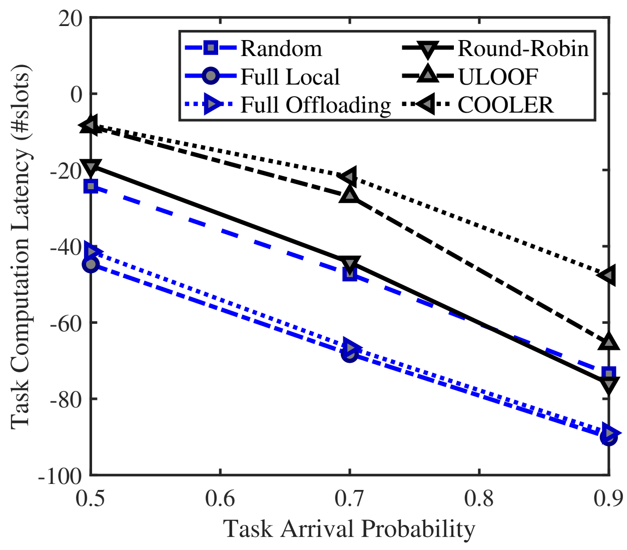

Fig. 5. Task Computation Latency (TCL) as a function of the task arrival probability $\mathcal{P}$ for different offloading schemes.

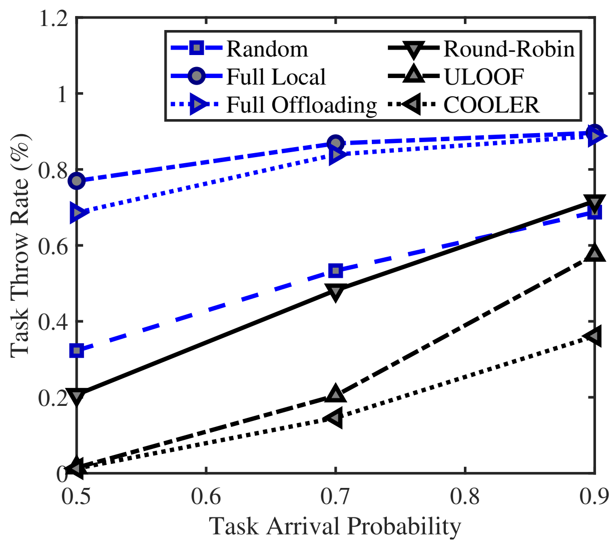

Fig. 6. Task Throw Rate (TTR) as a function of the task arrival probability $\mathcal{P}$ for different offloading schemes.

2) **Full Local:** Each EP executes all tasks locally without offloading.
3) **Full Offloading:** All tasks are offloaded from the local EP to a randomly selected destination, either another EP or the cloud. **Round-Robin:** Offloading decisions cycle through all possible destinations, including local execution, in a fixed order.
4) **ULOOF [17]:** A task offloading framework for non-divisible tasks and multi-agent systems, which bases offloading decisions on capacity estimations from historical data.

Fig. 5 illustrates the TCL as a function of the task arrival probability (0.5 to 0.9). Evidently, *Full Local* and *Full Offloading* schemes yield the poorest TCL across all tested probabilities, underperforming significantly compared to other schemes. Notably, COOLER and ULOOF emerge as the top performers. Interestingly, while COOLER matches ULOOF's performance at a task probability of 0.5, it exceeds ULOOF at higher probabilities (0.6 to 0.9), with the performance gap widening as task arrival probability increases. This suggests that COOLER adapts more efficiently to higher task loads, highlighting its robustness in more demanding scenarios.

Similarly, Fig. 6 presents the TTR as a function of task arrival probabilities (0.5 to 0.9). The performance ranking among the schemes mirrors the results for TCL, with COOLER consistently outperforming the baseline approaches, including ULOOF, particularly at higher task probabilities. This consistency in performance across both TCL and TTR metrics underscores COOLER's superior task handling and efficiency under varying operational conditions.

V. Conclusion

A. Summary

This paper presented the COOLER scheme, cooperative multi-agent framework proposed for efficient task offloading in the ECC. Utilizing latency-aware multi-agent DRL, COOLER addresses the dynamic and uncertain load characteristics at edge nodes by enabling autonomous offloading decisions optimized for non-divisible, delay-sensitive tasks. The COOLER's integration of LSTM, double DQN, and dueling DQN (DDDQN) techniques significantly enhances the accuracy of long-term cost estimations, thus improving the overall decision-making process. Simulation results have confirmed that COOLER excels in reducing task completion latency and task throw rates, substantially outperforming various baseline strategies. By minimizing expected long-term latency and the ratio of dropped tasks, COOLER demonstrates its efficacy in utilizing the computational resources of edge nodes more effectively and maintaining robust performance under different task traffic loads.

B. Possible Extensions

There are immediate extensions of the present work, aiming to enhance and evaluate the COOLER scheme, such as:

1) *Interoperability and Cross-Layer Optimization:* Future extensions could explore the interoperability between different edge computing platforms and cloud services. Enhancing COOLER to dynamically adapt based on cross-layer feedback might further optimize latency and resource utilization across the continuum.
2) *Energy Efficiency:* Another valuable extension would be to incorporate energy efficiency metrics into the COOLER decision-making process. This could involve

optimizing the trade-off between computation offloading and energy consumption, particularly crucial for battery-operated mobile devices.

3) *Prioritization-Based Task Execution:* Enhancing the COOLER scheme to include a prioritization mechanism where computational resources are not uniformly distributed but allocated based on priority scores of tasks could significantly optimize performance. This approach would allow for dynamic prioritization of tasks, where tasks with higher importance or urgency receive more computational resources or faster processing times.

Acknowledgment

The authors warmly thank the partners of ICOS project for their contribution in the system model and architectural aspects of this article.

References

[1] W. Tärneberg, E. Fitzgerald, M. Bhuyan, P. Townend, K.-E. Årzén, P.-O. Östberg, E. Elmroth, J. Eker, F. Tufvesson, and M. Kihl, “The 6g computing continuum (6gcc): Meeting the 6g computing challenges,” in *2022 1st International Conference on 6G Networking (6GNet)*. IEEE, 2022, pp. 1–5.

[2] P. Trakadas, X. Masip-Bruin, F. M. Facca, S. T. Spantideas, A. E. Giannopoulos, N. C. Kapsalis, R. Martins, E. Bosani, J. Ramon, R. G. Prats *et al.*, “A reference architecture for cloud–edge meta-operating systems enabling cross-domain, data-intensive, ml-assisted applications: Architectural overview and key concepts,” *Sensors*, vol. 22, no. 22, p. 9003, 2022.

[3] P. Gkonis, A. Giannopoulos, P. Trakadas, X. Masip-Bruin, and F. D’Andria, “A survey on iot-edge-cloud continuum systems: Status, challenges, use cases, and open issues,” *Future Internet*, vol. 15, no. 12, p. 383, 2023.

[4] M. Zetas, S. Spantideas, A. Giannopoulos, N. Nomikos, and P. Trakadas, “Empowering 6g maritime communications with distributed intelligence and over-the-air model sharing,” *Frontiers in Communications and Networks*, vol. 4, p. 1280602, 2024.

[5] A. Angelopoulos, A. Giannopoulos, N. Nomikos, A. Kalafatelis, A. Hatziefremidis, and P. Trakadas, “Federated learning-aided prognostics in the shipping 4.0: Principles, workflow, and use cases,” *IEEE Access*, 2024.

[6] H. Liu, R. Xin, P. Chen, and Z. Zhao, “Multi-objective robust workflow offloading in edge-to-cloud continuum,” in *2022 IEEE 15th International Conference on Cloud Computing (CLOUD)*. IEEE, 2022, pp. 469–478.

[7] L. Meng, Y. Wang, H. Wang, X. Tong, Z. Sun, and Z. Cai, “Task offloading optimization mechanism based on deep neural network in edge-cloud environment,” *Journal of Cloud Computing*, vol. 12, no. 1, p. 76, 2023.

[8] T. Q. Dinh, J. Tang, Q. D. La, and T. Q. Quek, “Offloading in mobile edge computing: Task allocation and computational frequency scaling,” *IEEE Transactions on Communications*, vol. 65, no. 8, pp. 3571–3584, 2017.

[9] T. Q. Dinh, Q. D. La, T. Q. Quek, and H. Shin, “Learning for computation offloading in mobile edge computing,” *IEEE Transactions on Communications*, vol. 66, no. 12, pp. 6353–6367, 2018.

[10] W. Li, J. Chen, Y. Li, Z. Wen, J. Peng, and X. Wu, “Mobile edge server deployment towards task offloading in mobile edge computing: A clustering approach,” *Mobile Networks and Applications*, vol. 27, no. 4, pp. 1476–1489, 2022.

[11] I. Ullah, H.-K. Lim, Y.-J. Seok, and Y.-H. Han, “Optimizing task offloading and resource allocation in edge-cloud networks: a drl approach,” *Journal of Cloud Computing*, vol. 12, no. 1, p. 112, 2023.

[12] L. Liu, H. Zhu, T. Wang, and M. Tang, “A fast and efficient task offloading approach in edge-cloud collaboration environment,” *Electronics*, vol. 13, no. 2, p. 313, 2024.

[13] C. Shu, Z. Zhao, Y. Han, G. Min, and H. Duan, “Multi-user offloading for edge computing networks: A dependency-aware and latency-optimal approach,” *IEEE Internet of Things Journal*, vol. 7, no. 3, pp. 1678–1689, 2019.

[14] A. Giannopoulos, S. Spantideas, N. Kapsalis, P. Karkazis, and P. Trakadas, “Deep reinforcement learning for energy-efficient multi-channel transmissions in 5g cognitive hetnets: Centralized, decentralized and transfer learning based solutions,” *IEEE Access*, vol. 9, pp. 129 358–129 374, 2021.

[15] A. Giannopoulos, S. Spantideas, N. Nomikos, A. Kalafatelis, and P. Trakadas, “Learning to fulfill the user demands in 5g-enabled wireless networks through power allocation: A reinforcement learning approach,” in *2023 19th International Conference on the Design of Reliable Communication Networks (DRCN)*. IEEE, 2023, pp. 1–7.

[16] Y. Ji, Y. Wang, H. Zhao, G. Gui, H. Gacanin, H. Sari, and F. Adachi, “Multi-agent reinforcement learning resources allocation method using dueling double deep q-network in vehicular networks,” *IEEE Transactions on Vehicular Technology*, 2023.

[17] J. L. D. Neto, S.-Y. Yu, D. F. Macedo, J. M. S. Nogueira, R. Langar, and S. Secci, “Uloof: A user level online offloading framework for mobile edge computing,” *IEEE Transactions on Mobile Computing*, vol. 17, no. 11, pp. 2660–2674, 2018.