Placing Computational Tasks within Edge-Cloud Continuum: a DRL Delay Minimization Scheme

Anastasios Giannopoulos[1][0000−0002−8602−7401], Andrés L. Suárez-Cetrulo[2], Xavi Masip-Bruin[3], Francesco D'Andria[4], and Panagiotis Trakadas[1]

[1] Department of Ports Management and Shipping, National and Kapodistrian University of Athens, Athens, Euboea, Greece
{angianno,ptrakadas}@uoa.gr

[2] Ireland's Centre for Applied AI (CeADAR), University College Dublin, D04 V2N9 Dublin, Ireland
andres.suarez-cetrulo@ucd.ie

[3] CRAAX Laboratory, Universitat Politècnica de Catalunya (UPC), Vilanova i la Geltrú, Spain
xavier.masip@upc.edu

[4] Eviden, BDS INN R&D (Formerly Called ATOS), 08800 Barcelona, Spain
francesco.dandria@eviden.com

Abstract. In the rapidly evolving landscape of IoT-Edge-Cloud continuum (IECC), effective management of computational tasks offloaded from mobile devices to edge nodes is crucial. This paper presents a Distributed Reinforcement Learning Delay Minimization (DRL-DeMi) scheme for IECC task offloading. DRL-DeMi is a distributed framework engineered to tackle the challenges arising from the unpredictable load dynamics at edge nodes. It empowers each edge node to independently make offloading decisions, optimizing for non-divisible, latency-sensitive tasks without reliance on prior knowledge of other nodes' task models and decisions. By framing the problem as a multi-agent computation offloading scenario, DRL-DeMi aims to minimize expected long-term latency and task drop ratio. Adhering to IECC requirements for seamless task flow within the Edge layer and between Edge-Cloud layers, DRL-DeMi considers three computation decision avenues: local computation, horizontal offloading to another edge node, or vertical offloading to the Cloud. Integration of advanced techniques such as long short-term memory (LSTM), double deep Q-network (DQN), and dueling DQN enhances long-term cost estimation, thereby refining decision-making efficacy. Simulation results validate DRL-DeMi's superiority over baseline offloading algorithms, showcasing reductions in both task drop ratio and average delay.

Keywords: IoT-edge-cloud continuum (IECC), Reinforcement learning, Resource allocation, Task offloading.

1 Introduction

1.1 Shifting to IoT-Edge-Cloud Continuum

The IoT-Edge-Cloud Continuum (IECC) represents a transformative shift in distributed computing [5], integrating edge computing with cloud services to deliver low-latency, scalable computing [13]. IECC leverages the proximity of edge nodes to end-users, reducing latency, conserving bandwidth, and enabling real-time data processing and analytics [5]. Key components include edge nodes, cloud data centers, and an orchestration layer for resource management and task distribution.

As we approach the 6G era, IECC is expected to meet the high demands of time-sensitive applications [12] and the massive traffic from IoT devices [15]. The 6G network aims to provide ultra-reliable low-latency communication (URLLC), enhanced mobile broadband (eMBB), and massive machine-type communications (mMTC) [1, 11]. IECC will be crucial in this context, enabling dynamic and adaptive task offloading to optimize computational resources and meet the stringent requirements of 6G networks.

1.2 Task Placement within IECC

Task offloading is essential in the IECC architecture, dynamically allocating tasks to the most suitable computing layer [8]. This optimizes resource utilization and minimizes delay, crucial for time-sensitive applications. IECC supports both vertical and horizontal task offloading, allowing tasks to be distributed not only from edge to cloud but also among edge nodes. This flexibility ensures efficient resource usage and enhances system resilience, meeting the evolving needs of IECC.

Task offloading in mobile edge and edge-cloud computing has been widely studied [10]. Dinh et al. [2] proposed an optimization framework for offloading from mobile to edge devices to minimize latency and energy consumption. Li et al. [7] used clustering to deploy mobile edge servers, reducing completion time and power consumption. Ullah et al. [14] applied DRL for optimizing offloading and resource allocation, improving resource utilization. Liu et al. [9] introduced a fast task offloading approach in edge-cloud environments, achieving near-optimal solutions with low overhead.

1.3 Paper Summary

While existing works highlight the benefits of efficient task offloading, they often assume known task models and conventional vertical task flow, not fitting with IECC principles. In this work, we propose the DRL-DeMi (Deep Reinforcement Learning for Delay Minimization) scheme to address these limitations. DRL-DeMi enables decentralized offloading decisions without knowledge of task models or other nodes' decisions, targeting a multi-agent system [3, 4]. Each computing node uses a double and dueling DRL model to optimize offloading decisions, minimizing task latency and drop probability. The main contributions of this work are: (i) DRL-DeMi allows edge nodes to autonomously make offloading decisions, suitable for dynamic IECC environments, without requiring knowledge of task models or global observability, (ii) DRL-DeMi supports both vertical and horizontal offloading, facilitating task flow within and between layers of the multi-agent computing architecture, (iii) Initial numerical validations demonstrate that DRL-DeMi outperforms baseline algorithms, reducing task drop rates and average delay, thus optimizing processing resource utilization in IECC.

2 System Model

This section describes the system model elements considered for the development of DRL-DeMi scheme.

2.1 DRL-DeMi System Architecture

The DRL-DeMi framework operates within the IoT-Edge-Cloud network model, comprising K Edge Agents (EAs) and a single Cloud entity, facilitating efficient task management across multiple IoT regions. Each EA k processes tasks locally or offloads them horizontally to another EA or vertically to the Cloud. Decisions are made autonomously by DRL models embedded within each EA, aiming to minimize the Task Computation Delay (TCD) and Task Drop Rate (TDR). The network is augmented with M Edge Controllers (EDs) for monitoring and facilitating inter-controller communication for data sharing. We focus on a time-slotted episode $\mathcal{T} = \{1, 2, \ldots, T\}$ (each time slot has $\mathit{\Delta}$ sec duration), maintaining communication via wireless links between IoT devices and base stations and wired fronthaul (FH) links connecting base stations to EAs, with all EAs linked to the Cloud via the Internet. The communication assumptions can be safely changed without loss of generality.

Fig. 1 depicts a general IECC system model engaging K DRL-DeMi agents in which, when a new task is arrived from an IoT zone, the respective EA is assisted by a local DRL-DeMi model to make a decision. The task placement decisions can enforce the task to be locally computed, vertically forwarded to Cloud, or horizontally offloaded to another EA.

2.2 Task Representation and Decision Process

In the DRL-DeMi system model, each EA k is assigned a unique task identifier $u_k(t)$ (integer) upon task arrival at time t. Task arrival is indicated by a binary variable $x_k(t)$ (1 if task arrived at t). Tasks have discrete sizes drawn from a set $\mathcal{H}$, associated with processing densities $\rho_k(t)$ (CPU cycles per bit) and deadlines ϕ_k (time slots). The offloading decision process is two-step and is achieved by two decision-maker (DM) modules: DM$_1$ determines local computation or offloading, and, if DM$_1$ decides offloading, then DM$_2$ selects the offloading destination.

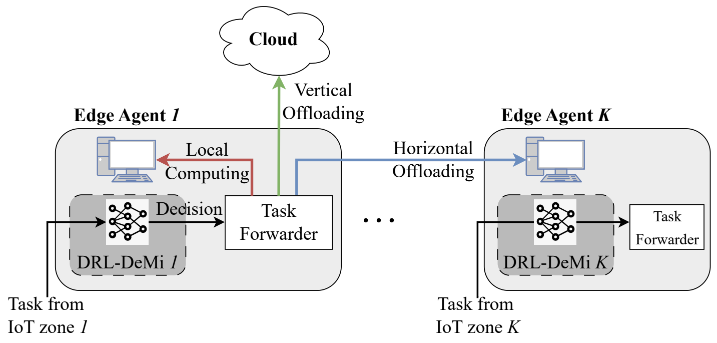

Fig. 1. A multi-agent and single-Cloud IECC system. Each Edge Agent employs a DRL-DeMi model to make a delay-aware task placement decision.

2.3 Task Queuing Mechanisms

Local Computing at Internal Queues: Tasks designated for local processing are stacked in the FIFO internal queue (IQ) of their respective EA. Each EA k has a CPU with fixed capacity f_k^{IQ} Hz for processing local tasks. The completion time slot $\psi_k^{IQ}(t)$ for task $u_k(t)$ is determined, with $\psi_k^{IQ}(t) = 0$ if no task is queued at time t. The waiting time $w_k^{IQ}(t)$, ensuring non-negativity, is computed as:

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

The completion time slot $\psi_k^{IQ}(t)$, indicating task completion or timeout, is given by:

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

Host Computing at External Queues: Each EA has $K-1$ external queues (EQs) for processing external tasks from other EAs, while the Cloud has K EQs for hosting tasks from all EPs. Each EQ i at node n serves as the offloading destination for EA i. Tasks offloaded by EA k and arriving at node n at time t are stacked in the k^{th} EQ at $t+1$. Each task receives a unique ID $u_{k,n}(t) = u_k(t' < t)$ upon queuing (at time t). The length of EQ k 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 EQ k of node n at the end of time slot t. This equation recursively updates the EQ length at time t as the previous EQ length at time $t-1$ plus the size of new task arrived at time t, minus the sum of all bits dropped and processed at time t.

Transferring Tasks with Forwarding Queue Each EA has another FIFO forwarding queue (FQ) for offloading tasks. When EA k selects task $u_k(t)$ for offloading, the FQ is connected to the destination EQ of another EA or Cloud via a wired link. The waiting time $w_k^{FQ}(t)$ and completion time slot $\psi_k^{FQ}(t)$ of task stored in FQ are computed similarly to the IQ case, ensuring timely task forwarding:

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

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

Here, $d_{k,n}^{(2)} = 1$ (decision made by decision-maker 2) indicates offloading from EA k to node n (EA or Cloud), and $R_{k,n}$ denotes the data rate (bits per sec) for offloading. Data rate can be either equal to R_V (EA-to-Cloud transfer) or R_H (EA-to-EA transfer).

3 DRL-DeMi Algorithm

3.1 Task Placement Problem Formulation

Each EA agent faces the task of deciding whether to process tasks locally, offload them to another peer EA agent, or send them to the Cloud. The key challenge lies in formulating a cost function that reflects the agents' objective: making offloading decisions to minimize both long-term computation delay (TCD) and the probability of task drops (TDR), ensuring efficient and reliable task handling in a dynamic computational environment.

At time slot $t \in \mathcal{T}$, the global system state is denoted by $\mathcal{S} = \{\mathcal{S}_1, \ldots, \mathcal{S}_K\}$, where $\mathcal{S}_k \in \mathcal{S}$ denotes the local state of EA $k \in \mathcal{K}$. Within an episode, each EA transitions through states $\{\mathbf{s}_k(1), \mathbf{s}_k(2), \ldots, \mathbf{s}_k(T)\}$. For a given t, EA k observes the local environment state $\mathbf{s}_k(t)$ and takes action $\mathbf{a}_k(t)$. This action leads to a new state $\mathbf{s}_k(t+1)$ and a scalar reward $r_k(t+1)$ reflecting the action's benefit. The state of EA k at t is defined as:

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

Here, $\mathbf{l}_k^{EQ}(t-1)$ is a vector containing EQ lengths for the EQs hosting tasks of EA k, and $\mathbf{L}(t)$ records previous load (i.e. number of queues containing at least one task) values of computing nodes. EA k selects action $\mathbf{a}_k(t)$ based on $\mathbf{s}_k(t)$:

$$\mathbf{a}_k(t) = \Big[d_k^{(1)}(t), d_{k,n}^{(2)}(t), n\Big] \tag{7}$$

This two-step decision determines whether to offload ($d_k^{(1)}(t) = 0$ for offloading) and the destination ($d_{k,n}^{(2)}(t) = 1$ for offloading and n is the destination EA). After taking $\mathbf{a}_k(t)$ from $\mathbf{s}_k(t)$, the received reward $r_k(t)$ is:

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

Defining π_k as the policy of EA k, which maps the states to actions, means that optimizing π_k for EA k involves minimizing the expected cumulative reward:

$$\pi_k^* = \arg\min_{\pi_k} \mathbb{E}\Big\{\sum_{t \in \mathcal{T}} \gamma^{t-1} \cdot r_k(t) \Big| \pi_k\Big\} \tag{9}$$

where expectation is taken over random arrivals and decisions, and γ is a discount factor. This tells us that the optimal policy π_k^* of EA k is the one that ensures that the long-term delay cost is minimized.

3.2 DRL-DeMi Algorithmic Scheme

DRL-DeMi aims to facilitate decentralized task offloading, jointly minimizing TCD and TDR. Each EA employs a DQN trained using double and dueling Q-learning principles for stabilizing learning. At time t, the DQN takes $\mathbf{s}_k(t)$ as input, producing Q-values for all actions. Training involves experience replay and two networks: the action-selecting Q-model and the reward-estimating Target Q-model. Fig. 2 demonstrates the Neural Network architecture used for DRL-DeMi.

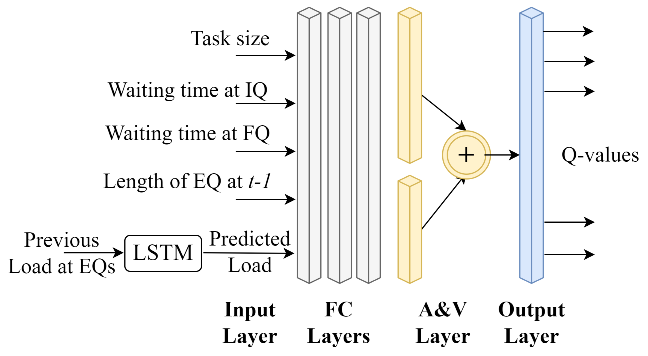

Fig. 2. Single-agent DRL-DeMi model architecture.

The DRL-DeMi algorithm trains K DQN models to assist decision-making for each EA, aiming at a distributed task offloading mechanism within the IECC.

Its goal is to minimize task latency and task drops. The DRL-DeMi training process for each EA k begins with the initialization of the policy Q-network (with parameters θ_k) and the target Q-network (θ'_k) with random weights. At each time slot t, the EA observes the environment and forms the current state $s_k(t)$. Based on this state, the policy Q-network selects an action $a_k(t)$ (random action or action with the minimum Q-value). The action is then executed, resulting in a transition to $s_k(t+1)$ and receiving an immediate cost $C_k(t)$. This transition $(s_k(t), a_k(t), C_k(t), s_k(t+1))$ is stored in a replay buffer with capacity N_R. A mini-batch of transitions is sampled from the replay buffer to update the Q-network. For each sampled transition, the $Q(\cdot)$ and target Q-values are computed using both networks and the double Q-learning equation [6]. The policy Q-network is updated by minimizing the loss between the predicted and the target Q-value. The target Q-model weights are updated every N_{clone} episodes to slowly track the policy network, following the update rule $\theta^*_k \leftarrow \tau\theta_k + (1-\tau)\theta^*_k$, where τ is an update constant in $[0, 1]$. This process iterates over multiple episodes and time slots t until convergence, ensuring continuous learning and adaptation of each DRL-DeMi agent to minimize long-term discounted cost.

4 Simulations

In this section, we numerically assess the performance of the DRL-DeMi scheme in an multi-agent setup. We first discuss the training dynamics and stabilization of DRL agent hyperparameters, then provide a comparative analysis against existing baseline methods. We consider the following system parameters: Task arrival probability is set at 0.7, with horizontal and vertical data rates at 10 Mbps and 20 Mbps, respectively. Task sizes vary between $2-5$ bits, and tasks have a deadline of 10 time slots. The task processing density is 0.297 cycles/bit. The system includes 3 EA. The CPU frequencies are 2.5 GHz for the IQs, 5 GHz for EQs, and 30 GHz for the Cloud.

4.1 DRL-DeMi Training

The training involves 12000 episodes, each comprising 100 time slots of 0.1 seconds each. The Q-network consists of three hidden layers with 20 neurons each, optimized using Adam and MSE as the loss function. The target network is updated every 2000 iterations. The LSTM model uses a lookback window of 10 steps and one hidden layer with 20 neurons. The replay memory size is 10000 samples, with a task drop penalty of 40. The batch size for training is 64 samples.

Fig. 3 shows the learning curve (in terms of cumulative reward) of the DRL-DeMi scheme for various learning rates and discount factors. Considered negative, reward ideally reaches zero, hence the increasing TCL curve. A learning rate of $a = 0.0001$ proved optimal, offering a balance between rapid learning and minimal overshoot, effectively minimizing the task delays. On average, DRL-DeMi achieves an 8-slot latency for task completion, resulting in about a 5% drop rate under the given system settings. The value of $\gamma = 0.99$ was proved

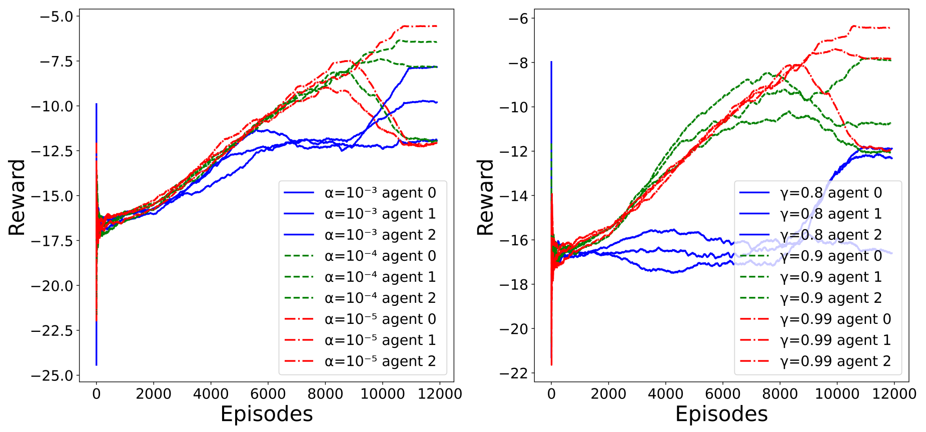

Fig. 3. The influence of learning rate (a) and discount factor (b) on DRL-DeMi reward convergence for three agents.

the optimal discount factor, meaning that DRL-DeMi agents prefer future rewards to optimize the task placement policy, rather than following actions with immediate rewards (low values of γ).

4.2 DRL-DeMi Validation

Considering the optimally-configured DRL-DeMi network at each EA, this section quantifies the performance of the overall DRL-DeMi scheme. To evaluate the scalability and task traffic impact on the DRL-DeMi scheme, Fig. 4a shows the relationship between Reward (i.e. TCD) and the number of EAs under varying task arrival probabilities. The results indicate that reward slightly worsens as the number of EAs increases. This suggests that although additional EAs enhance computational capacity, they also introduce greater coordination complexity and data transmission delays among agents, due to high traffic. Furthermore, there is a clear correlation between higher task arrival probabilities and increased delays. At lower probabilities, the system handles tasks more efficiently due to lower resource demands and less frequent decision-making by the DRL agents. However, as task arrival probability rises, the system becomes increasingly strained, resulting in slower task execution.

For validation purposes, DRL-DeMi was also compared against baseline methods. We considered the following schemes: (i) **Random:** Each EA offloads tasks randomly, with equal probability for local execution, vertical or horizontal offloading. When offloading horizontally, the destination EA is chosen randomly from the available EAs. (ii) **Full Local:** All tasks are executed locally by each EA. (iii) **Full Offloading:** Each EA offloads all tasks to a randomly chosen destination. (iv) **Round-Robin:** Offloading decisions follow a fixed order, cycling through all possible destinations, including local execution.

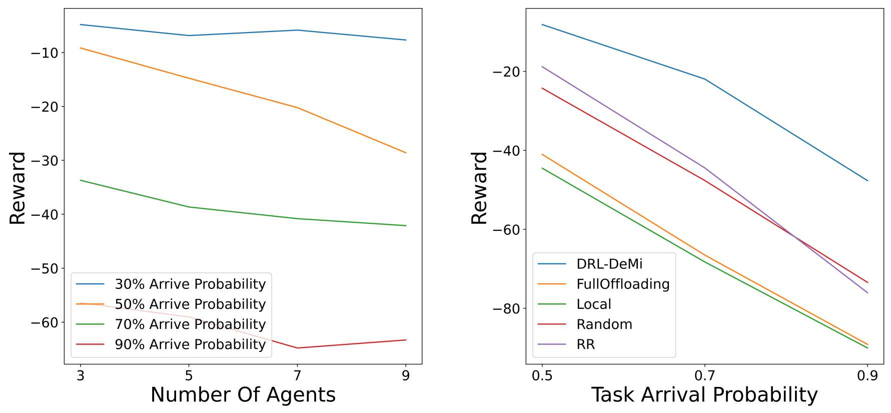

Fig. 4. a. The impact of the task arrival probability on the DRL-DeMi performance for increasing number of agents. **b.** Performance comparison of DRL-DeMi against baselines for different task arrival probabilities.

Evidently from Fig. 4b, DRL-DeMi outperformed the baselines, mainly due to its adaptive learning approach, allowing it to make more efficient offloading decisions based on real-time system states and task demands. Unlike static or random strategies, DRL-DeMi continuously optimizes task allocation by predicting long-term rewards, thereby minimizing latency. Also, Round-Robin and Random schemes showed better task placement than the other two deterministic methods, given that they both allow some degree of non-static decisions, thus preventing overloading. Overall, the use of DRL enables dynamic resource management and efficient utilization of computational resources, leading to superior performance in terms of task completion latency and overall system throughput.

Acknowledgments. This study was funded by the "ICOS Project" (Towards a functional continuum operating system), funded from the European Union's HORIZON research and innovation programme under grant agreement No 101070177.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Angelopoulos, A., Giannopoulos, A., Nomikos, N., Kalafatelis, A., Hatziefremidis, A., Trakadas, P.: Federated learning-aided prognostics in the shipping 4.0: Principles, workflow, and use cases. IEEE Access (2024)
2. Dinh, T.Q., Tang, J., La, Q.D., Quek, T.Q.: Offloading in mobile edge computing: Task allocation and computational frequency scaling. IEEE Transactions on Communications **65**(8), 3571–3584 (2017)

3. Giannopoulos, A., Nomikos, N., Ntroulias, G., Syriopoulos, T., Trakadas, P.: Maritime federated learning for decentralized on-ship intelligence. In: IFIP International Conference on Artificial Intelligence Applications and Innovations. pp. 195–206. Springer (2023)
4. Giannopoulos, A., Spantideas, S., Nomikos, N., Kalafatelis, A., Trakadas, P.: 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). pp. 1–7. IEEE (2023)
5. Gkonis, P., Giannopoulos, A., Trakadas, P., Masip-Bruin, X., D'Andria, F.: A survey on iot-edge-cloud continuum systems: Status, challenges, use cases, and open issues. Future Internet **15**(12), 383 (2023)
6. Ji, Y., Wang, Y., Zhao, H., Gui, G., Gacanin, H., Sari, H., Adachi, F.: Multi-agent reinforcement learning resources allocation method using dueling double deep q-network in vehicular networks. IEEE Transactions on Vehicular Technology (2023)
7. Li, W., Chen, J., Li, Y., Wen, Z., Peng, J., Wu, X.: Mobile edge server deployment towards task offloading in mobile edge computing: A clustering approach. Mobile Networks and Applications **27**(4), 1476–1489 (2022)
8. Liu, H., Xin, R., Chen, P., Zhao, Z.: Multi-objective robust workflow offloading in edge-to-cloud continuum. In: 2022 IEEE 15th International Conference on Cloud Computing (CLOUD). pp. 469–478. IEEE (2022)
9. Liu, L., Zhu, H., Wang, T., Tang, M.: A fast and efficient task offloading approach in edge-cloud collaboration environment. Electronics **13**(2), 313 (2024)
10. Meng, L., Wang, Y., Wang, H., Tong, X., Sun, Z., Cai, Z.: Task offloading optimization mechanism based on deep neural network in edge-cloud environment. Journal of Cloud Computing **12**(1), 76 (2023)
11. Skianis, K., Giannopoulos, A., Gkonis, P., Trakadas, P.: Data aging matters: Federated learning-based consumption prediction in smart homes via age-based model weighting. Electronics **12**(14), 3054 (2023)
12. Spantideas, S., Giannopoulos, A., Cambeiro, M.A., Trullols-Cruces, O., Atxutegi, E., Trakadas, P.: Intelligent mission critical services over beyond 5g networks: Control loop and proactive overload detection. In: 2023 International Conference on Smart Applications, Communications and Networking (SmartNets). pp. 1–6. IEEE (2023)
13. Trakadas, P., Masip-Bruin, X., Facca, F.M., Spantideas, S.T., Giannopoulos, A.E., Kapsalis, N.C., Martins, R., Bosani, E., Ramon, J., Prats, R.G., 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 **22**(22), 9003 (2022)
14. Ullah, I., Lim, H.K., Seok, Y.J., Han, Y.H.: Optimizing task offloading and resource allocation in edge-cloud networks: a drl approach. Journal of Cloud Computing **12**(1), 112 (2023)
15. Zetas, M., Spantideas, S., Giannopoulos, A., Nomikos, N., Trakadas, P.: Empowering 6g maritime communications with distributed intelligence and over-the-air model sharing. Frontiers in Communications and Networks **4**, 1280602 (2024)