# Joint Caching and Computing Service Placement for Edge-Enabled IoT based on Deep Reinforcement Learning

Yan Chen, Yanjing Sun, *Member, IEEE,* Bin Yang, and Tarik Taleb, *Senior Member, IEEE,*

***Abstract*—By placing edge service functions in proximity to IoT facilities, edge computing can satisfy various IoT applications' resource and latency requirements. Sensing-data-driven IoT applications are prevalent in IoT systems, and their task processing relies on sensing data from sensors. Therefore, to ensure the quality of service (QoS) of such applications in an edge-enabled IoT system, dedicated caching functions (CFs) are required to cache necessary sensing data. This paper considers an edge-enabled IoT system and investigates the joint caching and computing service placement (JCCSP) problem for sensing-data-driven IoT applications. Then, deep reinforcement learning (DRL) is exploited since it can adapt to a heterogeneous system with limited prior knowledge. In the proposed DRL-based approaches, a policy network based on the encoder-decoder model is constructed to address the issue of varying sizes of JCCSP states and actions caused by different numbers of CFs related to applications. Then, an on-policy REINFORCE-based method is adopted to train the policy network. After that, an off-policy training method based on the twin-delayed (TD) deep deterministic policy gradient (DDPG) is proposed to enhance the training efficiency and experience utilization. In the proposed DDPG-based method, a weight-averaged twin-Q-delayed (WATQD) algorithm is introduced to reduce the bias of Q-value estimation. Simulation results show that our proposed DRL-based JCCSP approaches can achieve converged performance that is significantly superior to benchmarks. Moreover, compared with the original TD method, the proposed WATQD method can significantly improve the training stability.**

***Index Terms*—Edge computing, IoT, joint service placement, sensing-data-driven applications, deep reinforcement learning.**

## I. Introduction

Edge computing enables IoT devices to quickly obtain cached content and computing services from nearby edge servers (ESs), which can greatly enhance reliability and reduce service latency. Thus, edge-enabled IoT has been identified as one fundamental architecture of the future IoT systems [1]. In recent years, edge computing has shown great potential for supporting applications with high resource requirements (e.g., CPU cycles, memory) and performance requirements (e.g., latency, reliability), such as autonomous vehicles and intelligent factories [2]. In the edge-enabled IoT systems, the caching and computing service placements are of fundamental importance to ensure the Quality of Service (QoS) by optimally placing them on nearby resource-limited ESs.

Existing works mainly studied the caching and computing service placements separately. For the caching service placement that places popular content on ESs, these works aim at optimizing the content hit ratio [3], [4] and the content caching cost [5], [6] by designing various schemes based on convex optimization, game theory and heuristic algorithms. On the other hand, the studies on computing service placement aim to optimally assign task processing of applications to ESs and place corresponding dependent computing functions (SF) to optimize system performances in terms of service latency [7], [8], computing overhead [9], [10], and other designed rewards or costs [11], [12]. Some research efforts seek to address a joint optimization of caching and computing service placements in IoT systems [13]–[15]. However, the above works mainly studied systems where each application only requires either one SF or one caching service function (CF). Sensing-data-driven IoT applications are prevalent in IoT systems, which require sensing data captured by deployed sensors to grasp the status of collaborative facilities/subsystems and make decisions based on it. Thus, to ensure the QoS of each sensing-data-driven application in an edge-enabled IoT system, necessary sensing data needs to be cached on ESs and managed by dedicated CFs. Meanwhile, every CF can pre-process the cached sensing data and quickly provide the requested sensing data when receiving a request from the corresponding SF during task processing. Therefore, considering resource-constrained ESs, it is essential to investigate the joint caching and computing service placement (JCCSP) for sensing-data-driven IoT applications when they are performed in an edge-enabled IoT system since the QoS of each of them jointly depends on the orchestration of SF and multiple related CFs.

Service placement problems are generally formulated as combinatorial optimization problems, which are challenging to address, especially considering the system heterogeneity in terms of resources and QoS requirements. Moreover, traditional approaches generally require prior knowledge of the

This work was supported in part by the National Natural Science Foundation of China under Grant 62071472; in part by the Fundamental Research Funds for the Central Universities under Grant 2020ZDPY0304; in part by the Chinese Government Scholarship under Grant 202006420096; in part by the Academy of Finland 6Genesis Project under Grant 318927; and in part by the IDEA-MILL under Grant 335936. An earlier version of part work in this paper was accepted by the IEEE International Conference on Communications (ICC), Seoul, South Korea, 16-20 May 2022. *(Corresponding author: Yanjing Sun, Bin Yang)*

Yan Chen and Yanjing Sun are with the School of Information and Control Engineering, China University of Mining and Technology, Xuzhou, 221116 China. (e-mail: chyan@cumt.edu.cn, yjsun@cumt.edu.cn).

Bin Yang is with the School of Computer and Information Engineering, Chuzhou University, Anhui, 239000 China. (e-mail: yangbinchi@gmail.com)

Tarik Taleb is with the Center of Wireless Communications, University of Oulu, 90570 Oulu, Finland, and also with the Department of Computer and Information Security, Sejong University, Seoul 05006, South Korea. (e-mail: Tarik.Taleb@oulu.fi).

whole system, including underlay communication networks like routing configurations and bandwidth consumption, which requires colossal overhead. Furthermore, it may not be feasible since the providers of the underlay communication networks and the edge service ones may be opaque in some actual IoT systems. Fortunately, deep reinforcement learning (DRL) can learn to manage an unknown system without prior knowledge [16]–[18]. Therefore, recent works have employed DRL to solve similar challenging optimization problems like resource allocation and service placement [19]–[27]. However, the JCCSP for multiple sensing-data-driven applications in an edge-enabled IoT system is challenging yet not studied in existing works. There are three main challenges in JCCSP for multiple sensing-date-driven IoT applications. First, the number of CFs related to each application is different, which results in the variable size of the system state used to represent each application. Meanwhile, the size of output JCCSP actions also varies and relies on the size of the input state representing an application. Thus, the simple fully-connected neural networks with fixed input and output sizes are unsuitable for our problem. Besides, various features of each services function and application (e.g., QoS requirement, resource consumption, number of related CFs) dramatically increase the system heterogeneity and the dimension of states used to represent all applications. Moreover, the JCCSP action of each application is the combination of actions selected for all related SF and CFs, which may result in a vast discrete action space with millions of candidate actions for just one application.

Therefore, this paper investigates the JCCSP for multiple sensing-data-driven IoT applications in an edge-enabled IoT system, aims to maximize the system reward relevant to the number of accepted applications and corresponding service latency. Then, considering the system heterogeneity and the limited prior knowledge of underlay communication networks, DRL-based approaches are proposed to train a policy that can optimize the expected accumulated reward of JCCSP actions from any observed initial state. The main contributions of this paper are summarized as follows.

- We investigate the JCCSP problem for multiple sensing-data-driven IoT applications and formulate it as a finite-steps Markov Decision Process (MDP) with the object of maximizing the accumulated reward achieved from providing low-latency services for all accepted applications.
- Considering the system heterogeneity and limited prior knowledge of communication networks, We exploit DRL to train a policy to generate the JCCSP action for any observed system state. A policy network based on the encoder-decoder model is constructed to tackle the issue of variable sizes of input states and output JCCSP actions caused by different numbers of CFs required by each application. Then, a REINFORCE-based on-policy approach is employed to train the policy.
- After that, to enhance the training efficiency and experience utilization in actual implementation, we propose an off-policy approach based on the twin-delayed (TD) deep deterministic policy gradient (DDPG). In the DDPG-based training method, we propose a weight-averaged twin-Q-delayed (WATQD) method to reduce the Q-estimation bias and improve the training stability.
- Simulation results show that the DRL-based approaches can converge after training and achieve optimal performances compared with benchmarks. Moreover, the proposed WATQD method can improve training stability compared with the original TD method.

The rest of this paper is organized as follows. Section II introduces related works. The system model and problem formulation are presented in Section III. Section IV details the proposed DRL-based JCCSP approaches. Simulations are conducted and discussed in Section V. Section VI discusses two extended research directions of this work. This paper is concluded in Section VII.

## II. Related Works

The available works on caching and computing service placements can be classified into two types: service placements based on traditional approaches (e.g., convex optimization, heuristic algorithm) and recent approaches based on DRL.

### *A. Service Placements with Traditional Approaches*

Under the traditional approaches, these studies mainly focus on either caching service placement, computing service placement, or a joint placement of them in edge computing systems. Regarding the caching service placement, it explores how to cache these frequently requested contents on ESs instead of the cloud server for improving various system performances. By relaxing the original problem to a convex optimization problem, the work in [3] designs a caching placement mechanism in cellular networks for maximizing the cache hit ratio (i.e., percentile of requested content obtained from ESs), and meanwhile for minimizing the data requested load from the cloud server. In [4], the authors propose a belief propagation based caching placement algorithm for minimizing the service cost averaging over the content downloading latency and transmission cost. The work in [5] utilizes the Lyapunov optimization approach with perturbation to optimize the placement of caching contents aiming to minimize the average content downloading latency. Similarly, based on the Lyapunov optimization approach, a collaborative edge data caching algorithm is further proposed to minimize the total system cost equal to the sum of data caching/migration cost and QoS penalty [6].

Through computing service placement, the tasks of an application can be assigned to be processed on a nearby ES for improving system performances. To this end, the work in [7] attempts to optimize the computing service placement for minimizing the average service latency of each application by designing three heuristic algorithms, i.e., latency aware heuristic placement algorithm, clustering enhanced heuristic placement algorithm, and substitution enhanced heuristic placement algorithm. In [8], an application-aware service placement mechanism is proposed to minimize the service latency by optimally assigning applications to ESs with different amount of computing resources. In [11], the diversity of ESs and applications is considered, and the maximization of system utility is formulated as an NP-hard optimization problem with the constraint of computing service placement, which is solved by a deterministic approximation algorithm. The work in [9] jointly optimizes the service placement and task routing in multi-cell edge computing networks by an algorithm based on the randomized rounding technique for maximizing the number of accepted applications. A decentralized algorithm is proposed to optimize the computing service placement in dense small cell edge computing network, where base stations and ESs can collaborate to execute tasks [10].

Some initial works have dedicated to the study of joint caching and computing service placements in a simple scenario where each application only request either one caching service or one computing service [13], [14]. The work in [13] investigates the joint placements for minimizing the total service latency using a convex optimization algorithm, where the service latency represents the sum of communication and computing latency. The work in [14] maximizes the system utility defined as the combination of the bandwidth usage and network latency by jointly optimizing caching and computing service placements. The work in [15] studies the duplicated computing tasks and results share among multiple users condition, and a computing result caching enhancement scheme is proposed to minimize the system execution delay.

### *B. Service Placements with DRL Approaches*

In [21], the authors propose a mean-field game guided deep Q-learning approach for achieving service latency minimization by optimizing computing service placement in cooperative multi-access edge computing system. A deep Q-learning-based computing service placement approach is proposed in [20] to maximize the system utility defined as the difference between the utility obtained from applications and the penalty of network congestion. The services in their work can be mapped to the physical network with prior knowledge. In [22], a novel two-timescale DRL-based approach is designed to minimize the cost of system task execution time and resource usage via jointly optimizing task offloading, resource allocation, and service placement. Meanwhile, federated learning is employed in their work for the purpose of data privacy. Notice that the deep Q-learning-based approach belongs to a class of value-based DRL approaches. However, in such approaches, each action is selected from all candidate actions for a given state, making them hard to apply in a system with a large or continuous state and action spaces. [18].

It is notable that the policy-based DRL approaches can handle the problem with continuous or large-scale discrete action and state spaces [23], [24]. Accordingly, the work in [25] employs a DDPG-based approach to optimally place the virtual network function (VNF) forwarding graph for maximizing the service chain acceptance ratio. The goal of [26] is to optimize the VNF placement and routing for achieving the maximization of system utility related to resource cost and service latency using a DDPG-based approach. In [27], the authors utilize a policy gradient method to select paths for service function chains from a predetermined candidate path set, which could maximize the system utility related to service benefits, cost and quality.

Different from the above works on the studies of the service placement for applications requiring only one service function or virtual network function chain, this paper investigates the JCCSP for sensing-data-driven IoT applications. Each application requires one SF and multiple CFs simultaneously, and its QoS depends on the joint placement of all related SF and CFs. Then, to cope with the issue that sensing-data-driven applications require different numbers of CFs, we construct an actor policy network based on the encoder-decoder model to realize the sequence-to-sequence mapping function from various applications to JCCSP actions. Different from the work in [28] considering the service placement for only one easily represented VNF chain, we investigate the JCCSP for multiple sensing-data-driven applications, in which the system state consists of the stationary and dynamic parts. The stationary part is an application to be placed. The dynamic part is the real-time resource consumption of ESs, updated after selecting an action for a service function at each decoding step and after implementing the JCCSP action of an application. Moreover, to cope with the challenge of enormous state and action spaces, we employ a REINFORCE-based on-policy training method and an off-policy DDPG-based method to train the policy. In the DDPG-based method, we propose a new WATQD Q-estimation method to improve training stability.

## III. System Model and Problem Formulation

### *A. System Model*

As shown in Fig. 1, we consider an edge-enabled IoT system consisting of sets of radio access networks (RANs), various sensors, and ESs $\mathcal{H}$ directly associated with some of these RANs. In the system, RANs provide communication services for IoT facilities, and various sensors are responsible for monitoring the status of IoT facilities and environment. Meanwhile, multiple sensing-data-driven IoT applications $\mathbb{A}$ are executed by IoT facilities to realize intelligent manufacturing, and every task processing relies on the sensing data captured by sensors deployed in the system. Moreover, ESs can satisfy the resource requirements (e.g., computing and storage resources) of executing IoT applications.

Each sensing-data-driven IoT application needs one SF and multiple CFs. The SF is responsible for task processing, and each CF manages the necessary sensing data cached from a related sensor. Thus, a certain application can be represented as $\mathcal{A}_i=\{\mathrm{SF}_i, \mathrm{CF}_i^1, \mathrm{CF}_i^{K_i-1}\}$, where $K_i-1$ indicates the number of sensors whose data is required by $\mathcal{A}_i$. The JCCSP refers placing the service functions (i.e., SF and CFs) on ESs so that the QoS of the application can be satisfied when being executed in the edge system. We use a binary indicator $\varphi_{i,k}^h = 1$ to represent the $k^{th}$ service function of $\mathcal{A}_i$ is placed on ES $h$, and $\varphi_{i,k}^h = 0$ otherwise. Meanwhile, each service function can only be placed on at most one ES, i.e.,

$$\textbf{C1:} \sum_{h\in\mathcal{H}} \varphi_{i,k}^h \le 1, 1 \le \forall i \le |\mathbb{A}|, 1 \le \forall k \le |\mathcal{A}_i|, \quad (1)$$

and

$$\textbf{C2:}\ \varphi_{i,k}^h = \{0,1\}, 1 \le \forall i \le |\mathbb{A}|, 1 \le \forall k \le |\mathcal{A}_i|. \quad (2)$$

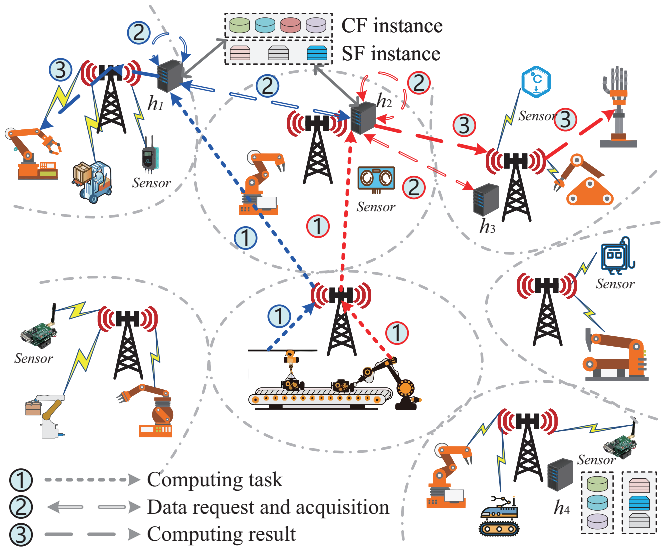


Fig. 1. An example of edge-enabled IoT system running sensing-data-driven IoT applications ,where $h_j \in \mathcal{H}$ $(1 \leq j \leq 4)$.

After placing a SF/CF on an ES, the resources are consumed by performing the service function. This paper considers two kinds of resources, i.e., the computing resource and the storage resource. We use $c_i^k$ and $s_i^k$ to represent the computing resource and the storage resource consumed by performing the $k^{th}$ service function of $\mathcal{A}_i$, respectively. Then, on any ES $h$, the resources consumed by all SFs and CFs placed on it can not exceed its maximum resource capacities, i.e.,

$$\textbf{C3:}\ \sum_{i=1}^{|\mathbb{A}|}\sum_{k=1}^{|\mathcal{A}_i|}\varphi_{i,k}^h c_i^k \leq C_h, \forall h \in \mathcal{H}, \tag{3}$$

and

$$\textbf{C4:}\ \sum_{i=1}^{|\mathbb{A}|}\sum_{k=1}^{|\mathcal{A}_i|}\varphi_{i,k}^h s_i^k \leq S_h, \forall h \in \mathcal{H}, \tag{4}$$

where $C_h$ represents the maximum computing resource capacity of ES $h$ and $S_h$ is the maximum storage resource capacity.

After successfully placing the SF and all CFs of an application on ESs, every related sensor sends the sensing data to the corresponding CF. The CF manages the caching of sensing data according to a pre-configured caching policy and provides required data when receiving a request. It is worth noting that the design of caching policies depends on exact applications and system conditions, which is also a challenging work due to dynamic and heterogeneous systems [29] and is beyond the scope of this paper. Thus we only consider the general resource consumption and placement of CFs. As shown in Fig. 1, after placing the SF and all CFs of a sensing-data-driven IoT application, the procedures of executing a task are as follows. A task is initiated by an IoT facility (initiator). First, the task is forwarded following the routing configured by the underlay communication networks to the ES where the corresponding SF is placed (i.e., the line labeled with circled 1). Then, the task is received by the corresponding SF. After that, the SF will request necessary sensing data from all related CFs placed on nearby ESs. The required sensing data is transmitted from every related CF to the SF via configured routing (i.e., the line labeled with circled 2). Once the SF fetches all the needed sensing data from the relevant CFs, it processes the task. After completing the task processing, the SF obtains a computing result, which would be forwarded to the executor of the task for execution (i.e., the line labeled with circled 3).

When sensing-date-driven IoT applications are performed, all related data flows are transmitted via communication links $\mathcal{L}$ established among RANs and follows the routing rules configured by the underlay communication networks. We use binary indicators $y_i^l = 1$ and $z_i^l = 1$ to separately indicate the computing tasks and the computing results of application $\mathcal{A}_i$ are transmitted via communication link $l$, otherwise $y_i^l = 0$ and $z_i^l = 0$. We use $x_{i,k}^l = 1$ to indicate the data caching from the related sensor to the $(k-1)^{th}$ CF of $\mathcal{A}_i$ is transmitted on link $l$, and use $\rho_{i,k}^l = 1$ to indicate the data request and acquisition between the SF and the $(k-1)^{th}$ CF of $\mathcal{A}_i$ is transmitted via link $l$. Otherwise, $x_{i,k}^l = 0$ and $\rho_{i,k}^l = 0$ separately. Then, the bandwidth consumed on a link $l$ is the sum of bandwidth consumed by all data flows allocated on it, which includes the sensing data caching flows (sensor→CF), the sensing data request and acquisition flows (SF↔CF), the transmission of computing tasks (initiator→SF), and the transmission of computing results (SF→executor). Moreover, the bandwidth consumed by the flows allocated on a link cannot exceed the maximum bandwidth capacity of the link $(B_l)$, i.e.,

$$\textbf{C5:}\ \sum_{i=1}^{|\mathbb{A}|}(y_i^l b_i^T + z_i^l b_i^R + \sum_{k=2}^{|\mathcal{A}_i|}(x_{i,k}^l b_{i,k}^C + \rho_{i,k}^l b_{i,k}^M)) \leq B_l, \forall l \in \mathcal{L}, \tag{5}$$

where $b_i^T$ and $b_i^R$ are the bandwidths required by transmitting computing tasks and computing results of $\mathcal{A}_i$, separately. $b_{i,k}^C$ represents the bandwidth consumed by the data caching flow from the sensor to the $(k–1)^{th}$ CF of $\mathcal{A}_i$, and $b_{i,k}^M$ represents the bandwidth required by the sensing data request and acquisition flow between the $(k-1)^{th}$ CF and the SF of $\mathcal{A}_i$.

Meanwhile, according to the above work procedures, the edge computing service latency of a sensing-data-driven IoT application $\mathcal{A}_i$ can be calculated as

$$\begin{gathered} D_i = \sum_{l \in \mathcal{L}}(y_i^l d_i^T + z_i^l d_i^R) + \max_{k \in \mathcal{K}}\{\sum_{l \in \mathcal{L}} \rho_{i,k}^l d_{i,k}^M\} + d_i^P, \\ \mathcal{K} = \{2, 3, \cdots, |\mathcal{A}_i|\} \end{gathered} \tag{6}$$

where $d_i^T$ and $d_i^R$ are the latency of transmitting the computing tasks and results on each one-hop communication link separately under required bandwidths. $d_i^P$ is the time consumed by processing the computing tasks when satisfying the required computing resources. $d_{i,k}^M$ represents the one-hop transmission delay of fetching sensing data from the $(k-1)^{th}$ CF to the SF. It is worth noting that the data caching from a sensor to the corresponding CF is actively performed by the sensor and independent from the edge computing service process. Meanwhile, we assume that all required data has already been cached on ESs and SFs do not request data from sensors directly. Thus, the data caching process contributes nothing to the service latency. Moreover, we assume that every SF sends short control frames to request sensing data, and every CF can replay the required sensing data quickly. Thus, we ignore the negligible delay of sensing-data requests and data preparation. To ensure QoS, the service latency of an application $\mathcal{A}_i$ should not be greater than its predefined latency constraint $\tau_i$, i.e.,

$$\textbf{C6: } D_i \leq \tau_i, \forall i \in \{1, 2, \cdots, |\mathbb{A}|\} \tag{7}$$

### *B. JCCSP Problem Formulation*

Due to the limited resources in computing, storage, and bandwidth, an edge system can only support limited applications, which makes some applications' requirements unable to be satisfied and have to be supported by other systems. Therefore, we define an application as accepted by the edge system if all of its service functions are placed on ESs without breaking any resource constraint and its QoS (i.e., service latency constraint) can be satisfied. Otherwise, we set the application is rejected by the system, and any of its service functions cannot be placed on ESs maintained by the system. We use a binary indicator $\zeta_i = 1$ to indicate the application $\mathcal{A}_i$ is accepted and $\zeta_i = 0$, otherwise. Then, we have

$$\textbf{C7: } \zeta_i = \begin{cases} 1, & \text{if } \sum_{k=1}^{|\mathcal{A}_i|} \sum_{h\in\mathcal{H}} \varphi_{i,k}^h = |\mathcal{A}_i|, \\ 0, & \text{otherwise.} \end{cases} \tag{8}$$

For a certain application $\mathcal{A}_i$, $\varphi_{i,k}^h$ indicates the final JCCSP result, which is jointly determined by the JCCSP policy generated by the edge service controller and the executing result of the policy. After the controller generate a JCCSP policy for $\mathcal{A}_i$ (i.e., $\varphi_{i,k}^h, 1 \leq k \leq |\mathcal{A}_i|, h \in \mathcal{H}$), there are two results of executing the policy in the IoT system: (1) the policy is executed successfully, and the application is accepted. (2) implementing the policy breaks the resource constraint or the QoS constraint, and the system rejects the application. If the application is rejected, the JCCSP policy for the application is cleared, i.e., $\varphi_{i,k}^h = 0, 1 \leq \forall k \leq |\mathcal{A}_i|, \forall h \in \mathcal{H}$.

From the view of an edge service provider, the primary objective is to fully utilize maintained resources (i.e., ESs) to enable higher-performance service for more applications. Therefore, we define a system reward that combines the number of accepted applications and the performance reward obtained for providing low-latency service for accepted IoT applications. Thus, we define the objective of our investigated JCCSP problem as maximizing the system reward, i.e.,

$$\begin{aligned} \textbf{P1: } & \max_{\boldsymbol{\varphi}} \sum_{i=1}^{|\mathbb{A}|} (\zeta_i W_i \frac{\tau_i}{D_i} + \zeta_i), \\ & \textit{s.t. } \textbf{C1} \sim \textbf{C7}, \end{aligned} \tag{9}$$

where $\tau_i/D_i$ is the reward obtained from performing application $\mathcal{A}_i$, which can reflect that a lower latency brings a higher reward. Moreover, $W_i$ is an additional weight value used to indicate the importance of the application, which applies a gain to the base reward obtained from performing the application, i.e., providing low-latency service for an application with higher importance factor can achieve more reward. We defined $W_i \tau_i / D_i$ as the latency performance reward achieved from accepting and performing $\mathcal{A}_i$. Besides, $\sum_{i=1}^{|\mathbb{A}|} \zeta_i$ can indicate the number of accepted applications.

The problem **P1** is a typical combinatorial optimization problem, which is challenging to address. Meanwhile, the difficulty is dramatically increased when considering the heterogeneity of the edge system and applications in terms of resource consumption, QoS, number of related service functions. Moreover, traditional methods based on global system knowledge cannot be applied to our considered systems, where the edge service controller can only access limited prior knowledge. Therefore, in the next section, we address the problem by exploiting the advantage of DRL, which can learn to solve a complex problem without the requirement of prior knowledge and can work efficiently after being trained.

## IV. DRL-based JCCSP Approaches

In this section, we first formulate the considered JCCSP problem as a finite-steps MDP. Then, we give the designs of state, action, and reward. After that, DRL-based JCCSP approaches are proposed based on these designs to address the JCCSP problem for multiple sensing-data-driven applications.

### *A. Problem Analysis and Approach Design*

For our considered JCCSP problem of multiple sensing-data-driven IoT applications in a heterogeneous edge-enabled IoT system, it is hard to orchestrate the service placement of all applications at once time. The reason is that the high-dimension state that can represent all applications is hard constructed due to applications' heterogeneity in terms of sequence length, service type (i.e., SF and CF), QoS, and resource consumption. Besides, it is difficult to determine which applications should be rejected when they cannot be all satisfied due to the limited resources of the edge system since their JCCSP policies are coupled and interfere with each other when being considered simultaneously. Therefore, we set the controller to arrange the JCCSP for multiple applications one-by-one. For simplification and maintaining potential fairness, the processing order of applications is according to their generation order. Then, the process of JCCSP for multiple applications can be modeled as an MDP with finite steps, in which the system state depends on one application and system resource consumption. Meanwhile, the state transition is only based on the JCCSP action taken for the application and current system state. Then, we can address the problem by exploiting the advantage of the potential that DRL can learn to adapt to an unknown system without prior knowledge.

In a DRL-based JCCSP approach, an actor policy $\pi$ with parameters $\theta$ is executed by the edge service controller, which is responsible for interacting with the IoT system to generate a JCCSP action $a_t$ for any observed system state $s_t$, i.e., $\pi_\theta : s_t \rightarrow a_t, \forall t$. After performing the action in IoT system, the controller receives a reward $r_t$, and the system transits to a new state $s_{t+1}$. In the MDP of our considered problem, the JCCSP trajectory can be represented as a sequence on state, action and reward, i.e., $(s_1, a_1, r_1), \cdots, (s_t, a_t, r_t), \cdots, (s_d, a_d, r_d)$. Note that $(s_d, a_d, r_d)$ denotes that the system reaches a done state, with which the controller stops making any JCCSP decision. We set the system reaches a done state only if satisfying any one of the following two conditions: (1) all applications are accepted by the system successfully; (2) the implementation of an JCCSP action breaks any resource/performance constraint introduced in Section III.

Based on the investigated problem, we define the state, the action, and the reward function used in our DRL-based approaches as follows.

**State:** Due to limited prior knowledge of the underlay communication networks, the system state $s_t$ observed by the edge service controller only consists of two parts: a sensing-data-driven IoT application to be placed and the real-time resource consumption of managed ESs, i.e.,

$$s_t = \{\boldsymbol{s}^{\boldsymbol{H}}_t, \boldsymbol{s}^{\boldsymbol{a}}_t\}, \tag{10}$$

where $\boldsymbol{s}^{\boldsymbol{H}}_t$ is the resource consumption of all managed ESs (i.e., $\boldsymbol{s}^{\boldsymbol{H}}_t = [C_h, S_h, C^h_{o,t}, S^h_{o,t}]_{4\times|\mathcal{H}|}$), and $\boldsymbol{s}^{\boldsymbol{a}}_t$ represents the sequence of SF and CFs of an applications $\mathcal{A}_t$ to be placed. $C^h_{o,t}$ and $S^h_{o,t}$ separately represent the real-time resource consumption of ES $h$ at time $t$. Moreover, each $\boldsymbol{s}^{\boldsymbol{a}}_t$ consists of one SF and multiple CFs (i.e., $s^a_t = [\text{SF}_t, \text{CF}^1_t, \text{CF}^2_t, \cdots]$), and every SF/CF is packaged with detail of relevant resources and QoS requirements. The state of each $\text{SF}_t$ includes the $b^T_t, b^R_t, d^T_t, d^R_t, d^P_t, W_t, \tau_t, s^1_t, c^1_t$ as well as the indexes of RANs directly associated with the initiator and executor of the application. The state of $\text{CF}^{k-1}_t$ consists of $b^M_{t,k}, b^C_{t,k}, d^C_{t,k}, c^k_t, s^k_t$ and the index of each related sensor.

**Action:** At each time $t$, the edge service controller generates a JCCSP action $a_t$ according to the observed state $s_t$. The action is represented by a sequence, and each element of the sequence represents an ES on which the corresponding service function of $\mathcal{A}_t$ will be placed. Thus,

$$a_t = [a^j_t]_{1\times|\mathcal{A}_t|}. \tag{11}$$

**Reward:** A reward $r_t$ is obtained depending on if the application $\mathcal{A}_t$ is accepted and its service latency after the system execute $a_t$ to place the SF and all CFs of $\mathcal{A}_t$ on ESs. Based on **P1**, we define the reward function as

$$r_t = \begin{cases} \beta_1 w_t(\frac{\tau_t}{d_t}) + \beta_2 \frac{t}{|\mathbb{A}|}, & \text{if } \mathcal{A}_t \text{ is accepted,} \\ 0, & \text{otherwise,} \end{cases} \tag{12}$$

where $w_t = \frac{W_t}{\sum_{j=1}^{|\mathbb{A}|} W_j}$ is the normalized weight value of the application $\mathcal{A}_t$. Meanwhile, $\beta_1$ and $\beta_2$ are the weight factors used to balance the two parts of the reward function.

Since different applications may require different numbers of CFs, making the service function sequences representing applications have different lengths. Meanwhile, the size of an output JCCSP action depends on the length of the corresponding input application sequence (i.e., the number of services required by the application). Therefore, neural networks with fixed input and output sizes are inappropriate for such a sequence-to-sequence transform problem. The encoder-decoder model is an efficient architecture widely employed in the natural language processing community and can handle such sequence-to-sequence issues that output depends on the input sequence [30], [31]. Meanwhile, it has shown its potential in solving the placement and combination optimization problems [28], [32]. Therefore, we construct the actor policy network in our DRL-based JCCSP approaches based on the encoder-decoder model. As shown in Fig. 2, the structure of the constructed actor policy network consists of four main parts. The SF/CF Embedding component is responsible for embedding the information of corresponding service functions (i.e., SF and CF) in the input system state. The encoder and decoder are two main components used to realize the function of sequence-to-sequence mapping from state to JCCSP action. The action generation component with constraint filter can generate the final JCCSP action based on the output of the decoder. The encoder and decoder are constructed by long short-term memory (LSTM) networks.

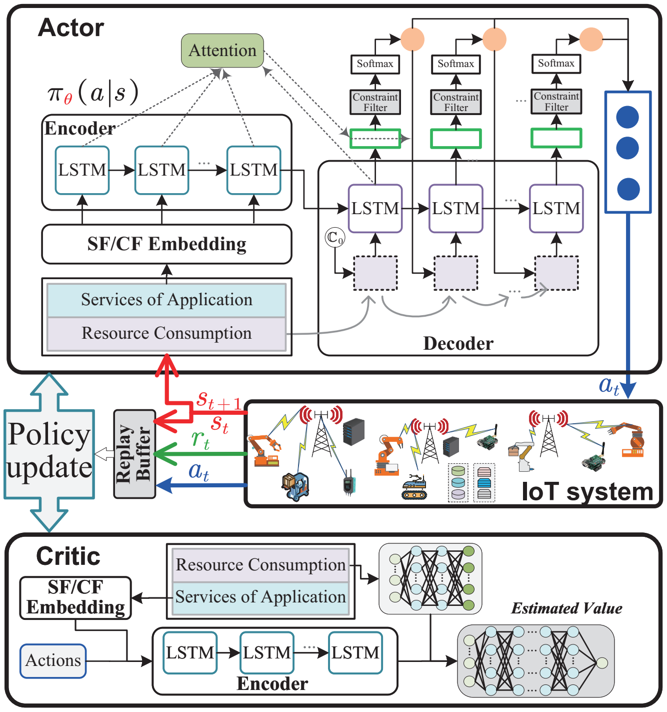


Fig. 2. The training architecture of DRL-based JCCSP approaches based on the Actor-Critic framework.

The work procedure of the actor policy network that generate the JCCSP action for one application is detailed in Algorithm 1. After receiving the observed system state, the service functions (i.e., SF and CF) are embedded by corresponding SF/CF embedding layer. Then, as shown in line 5 the embedded results are concatenated to a sequence ($s^E_t$). At the last encoding step, a final hidden state ($\mathbb{Y}_0$) is obtained and hidden states ($\mathbb{H}_t$) at each encoding step are also obtained at the same time (line 7). After that, the decoding process is triggered. At each decoding step, a hidden state ($\mathbb{Y}_j$) is generated with the inputs of the decoder network (i.e., LSTM) including the hidden state obtained from the previous decoding step ($\mathbb{Y}_{j-1}$) and an input tuple ($\boldsymbol{s}^{\boldsymbol{h}}_{t,j-1}, \mathbb{C}_{j-1}$), i.e.,

$$\mathbb{Y}_j = \text{LSTM}(\mathbb{Y}_{j-1}, (s^h_{t,j-1}, \mathbb{C}_{j-1})), \tag{13}$$

where $\boldsymbol{s}^{\boldsymbol{h}}_{t,j-1}$ represents the real-time estimated resource consumption of ESs (represented by ($\boldsymbol{C}^{\boldsymbol{h}}_{\boldsymbol{o}}, \boldsymbol{S}^{\boldsymbol{h}}_{\boldsymbol{o}}$) in lines 6, 11, 13 and 17) if implementing the action obtained from previous decoding step. Moreover, the real-time estimated resource consumption is processed by a one-layer fully connected

**Algorithm 1:** Single JCCSP action generation

```
   Input: IoT system, actor network $\theta$, service sequence
          of an application $\mathcal{A}_t$ to be placed
   Output: $a_t \leftarrow$ service placement action for $\mathcal{A}_t$,
           $\mathcal{P}_t \leftarrow$ probability of all candidate actions
1  $\boldsymbol{s_t^H} \leftarrow$ Resource consumption state of all ESs
2  $\boldsymbol{s_t^a} \leftarrow$ Service function sequence of $\mathcal{A}_t$
3  $s_t = \{\boldsymbol{s_t^H}, \boldsymbol{s_t^a}\}$
4  Input $s_t$ to the actor network
5  $s_t^E \leftarrow$ Embedded SF and CFs sequence
6  $(\boldsymbol{C_o^h}, \boldsymbol{S_o^h}) \leftarrow \boldsymbol{s_t^H}$
7  $(\mathbb{H}_t, \mathbb{Y}_0)$ = Encoder($s_t^E$)
8  $\mathbb{C}_0 \leftarrow$ The trainable start input context
9  for $j = 1, \cdots, |\mathcal{A}_t|$ do
10     $(c_t^j, s_t^j) \leftarrow$ Resources required by the $j^{th}$ service
11     $(\mathbb{Y}_j, \mathbb{P}_j) = f^D(\mathbb{Y}_{j-1}, \mathbb{H}_t, (\boldsymbol{C_o^h}, \boldsymbol{S_o^h}))$
12     $\mathbb{P}_j \leftarrow [p_1, p_2, \cdots, p_{|\mathcal{H}|}]$
13     $\boldsymbol{g} = [(\boldsymbol{C_o^h} + c_t^j) > \boldsymbol{C_h}] + [(\boldsymbol{S_o^h} + s_t^j) > \boldsymbol{S_h}]$
14     $\mathbb{P}_j = \mathbb{P}_j(1-g) + (\mathbb{P}_j - \infty)\boldsymbol{g}$
15     $\boldsymbol{P}_j$ = softmax($\mathbb{P}_j$)
16     $a_t^j$= action_sample($\boldsymbol{P}_j$)
17     Update $(\boldsymbol{C_o^h}, \boldsymbol{S_o^h})$
18 Return: $a_t = [a_t^1, \cdots, a_t^{|\mathcal{A}_t|}], \mathcal{P}_t = [\boldsymbol{P}_1, \cdots, \boldsymbol{P}_{|\mathcal{A}_t|}]$
```

neural network before being concatenated with the context vector $\mathbb{C}_{j-1}$ generated in previous decoding step (i.e., the rectangles with dashed edges in the decoder). In each decoding step, the context vector is obtained by performing an attention mechanism [33] on the output hidden state of LSTM, i.e.,

$$\mathbb{C}_j = \text{Attention}(\mathbb{Y}_j, \mathbb{H}_t). \tag{14}$$

Then, the context vector is passed through a linear layer (i.e., represented by the green rectangle above the decoder) whose output size equals the number of ESs in the system. We use a map function to represent the above procedures (line 11). The initial context vector ($\mathbb{C}_0$) used at the first decoding step is a set of trainable variables. Then, the output of the linear layer $\mathbb{P}_j$ is processed by a softmax function to obtain the probability ($\boldsymbol{P}_j$) of selecting each ES to place the service (line 15). Moreover, a constraint filter is added before executing the softmax function to avoid selecting an ES that will break its resource constraint after placing the service. If placing the service on one ES will break its resource constraint, the constraint filter sets the corresponding score value to be $-\infty$ (line 13 and line 14). Then, the probability of selecting the ES after the softmax calculation is 0. Finally, the final action ($a_t^j$) of selecting an ES to place the $j^{th}$ service function required by $\mathcal{A}_t$ is determined based on these probabilities (line 16). Besides, the estimated resource consumption of ESs is updated based on the resource requirements of the service and selected action (line 17). After finishing all the decoding steps, the action for placing all service functions of the application (i.e., $a_t = [a_t^j]_{1\times|\mathcal{A}_t|}$) is obtained (i.e., the blue rectangle filled with blue cycles on the right of the decoder). The IoT system will implement the concatenated JCCSP action of the application. After placing the SF and all CFs of an application, the underlay communication networks will report the link congestion, and the edge service controller can observe latency. Then, if any resource/performance constraint is unsatisfied, the system rejects the application by removing its SF and CFs from ESs. Otherwise, the SF and CFs of the application will be performed on ESs, and the resource consumption state will be updated. Meanwhile, according to the defined reward function (12), the controller will obtain an immediate reward based on the result of implementing the JCCSP action and the latency performance of the application. Then, a new system state $s_{t+1}$ consists of the updated resource consumption state of ESs, and the subsequent application to be placed is constructed and processed by the actor policy network by repeating the above procedures until the system enters the done state.

The objective of DRL-based approaches is to train an optimized actor policy, which can generate optimized JCCSP actions for given observed system states and maximize the expected accumulated reward of any JCCSP trajectory, i.e.,

$$\theta^* = \arg\max_{\theta} \mathbb{E}_{\{s;\pi_\theta\}} R(s), \tag{15}$$

where $R(s) = \sum_{t=1}^{d} r_t$ is the accumulated reward.

The JCCSP action of any sensing-data-driven IoT application is the combination of service placement actions of all related SF and CFs, resulting in a vast discrete action space with even millions of candidate actions. Besides, any observed system state consists of an uncertain application and the uncertain resource consumption of ESs, which increases the randomness of enormous state space. Therefore, it is necessary to use the policy-based DRL method in our investigated JCCSP problems with massive action and state spaces. The reason is that value-based reinforcement learning methods (e.g., Q-learning, Deep Q-learning) can only address problems with limited discrete action space. Therefore, this paper employs policy-based approaches to train the actor policy network, i.e., an on-policy approach based on REINFORCE method and an off-policy approach based on the DDPG method.

To train the parameters of the actor policy networks, as shown in Fig. 2, the critic network is also constructed to estimate state values or state-action values [19]. The estimated values are separately used to assist the training process in the REINFORCE-based approach and the DDPG-based approach.

### B. REINFORCE-based Model Training

Based on the well-known REINFORCE policy gradient algorithm, the gradient of (15) can be approximated as [34]

$$\nabla_\theta J(\theta) = \mathbb{E}_{\{s;\pi_\theta\}}\left[(R(a|s) - \tilde{v}(s))\nabla_\theta \log p_\theta(a|s)\right], \tag{16}$$

where each $s$ and $a$ are experienced state and corresponding action generated following policy $\pi_\theta$ in an JCCSP trajectory. Each action for placing a service is sampled based on the probability distribution of selecting each ES (i.e., $\boldsymbol{P}_j$ in line 15 of Algorithm 1). The $\tilde{v}(s)$ is a baseline, which generally employed as the estimated state value generated by an independent auxiliary critic policy network, as shown in Fig. 2. It should be noted that actions marked in the critic presented in Fig. 2 is not used in the REINFORCE-based approach, i.e., only the input states are processed by the critic network in this approach. The $p_\theta(a|s)$ is the probability of taking action $a$ given the input system state is $s$ according to the policy $\pi_\theta$. In an actual implementation, the $R(a|s)$ is usually employed with a discounted format of all immediate reward in the following steps of a placement trajectory, i.e.,

$$G_t = \sum_{n=t}^{\infty} \gamma^{n-t} r_n, \tag{17}$$

where $\gamma$ is the discount factor and $\gamma \in [0, 1)$.

According to the work procedures of the actor policy network, a final JCCSP action and probabilities of selecting each ES to place every service function can be obtained by the controller (line 18 in Algorithm 1). Then, by using the chain rule [30], [31], the probability of selecting the JCCSP action $a_t$ for the corresponding system state $s_t$ can be calculated as

$$p_\theta(a_t|s_t) = \prod_{j=1}^{|\mathcal{A}_t|} p_\theta(a_t^j | a_t^{(i<j)}, s_t; \theta), \tag{18}$$

where $p_\theta(a_t^j | a_t^{(i<j)}, s_t; \theta)$ represents the conditional probability of selecting action $a_t^j$ at $j^{th}$ decoding step given all actions $a_t^i$ have been selected in all previous $i^{th}$ decoding steps satisfying $i < j$.

**Algorithm 2:** Training process based on REINFORCE

```
Input: IoT system, an actor network ($\theta$), and a critic
       network ($\psi$)
Output: The trained JCCSP Policy network.
1  for $n = 1, 2, \cdots$ do
2      System initialization
3      $\mathbb{A} \leftarrow$ A set of applications to be placed.
4      for $t = 1, 2, \cdots, |\mathbb{A}|$ do
5          $s_t = \{\boldsymbol{s}_t^{\boldsymbol{H}}, \boldsymbol{s}_t^{\boldsymbol{a}}\}$
6          $\tilde{v}_t$=Critic($s_t$) $\leftarrow$ Estimated state value.
7          Obtain $a_t$ and $\mathcal{P}_t$ by performing Algorithm 1
8          $r_t \leftarrow$ Reward obtained after executing $a_t$
9          Calculate $p_\theta(a_t|s_t) = \prod_{j=1}^{|\mathcal{A}_t|} \mathcal{P}_t(j, a_t^j)$
10         Record $(s_t, a_t, r_t, p_\theta(a_t|s_t), \tilde{v}_t)$
11         if $r_t == 0$ then
12             break
13         else
14             Update system to $s_{t+1}$
15     Calculate $G_t$ for the experienced trajectory.
16     $J(\theta) = -\mathbb{E}[(G_t - \tilde{v}_t) \log p_\theta(a_t|s_t)]$
17     $\mathcal{L}(\psi) = \mathbb{E}[(G_t - \tilde{v}_t)^2]$
18     $\theta$ = RMSprop($\nabla_\theta(J(\theta))$)
19     $\psi$ = RMSprop($\nabla_\psi(\mathcal{L}(\psi))$)
```

The procedure of training the JCCSP policy based on the REINFORCE algorithm is illustrated in Algorithm 2. At the beginning of each training episode, the IoT system is initialized to a state without performing any applications. And a set of applications $\mathbb{A}$ is waiting for placing their related SFs and CFs on ESs. At each step, the controller constructs a system state based on the observed resource consumption of ESs and an application to be placed (line 5). Then, the state is processed by the actor policy network to obtain JCCSP action and probability of selecting every action for placing services related to the application (line 7). Meanwhile, a state value is estimated by the critic network with the input of the same state (line 6). After that, the IoT system executes the JCCSP action $a_t$, and a reward is obtained after the execution (line 8). The probability of taking the JCCSP action $a_t$ is calculated in line 9. Before executing the JCCSP action in the IoT system, the edge service controller cannot check the performance and bandwidth constraints since it has no authority to access the configuration and states of the underlay communication networks. Besides, breaking a resource constraint of the underlay communication networks may cause unpredictable influences on other parts of the system. Therefore, we set the system to transit into the done state and reject the subsequent actions once the system rejects a JCCSP action and obtains a 0 reward. Otherwise, the system will transit into a new state $s_t$ and re-perform the above process until all applications are accepted. Every experienced state, action, reward, action probability, and estimated state value in one trajectory are recorded (line 10). After reaching a done state, the DRL agent updates its policy networks based on recorded experiences of states, actions, rewards, action probabilities, and state values (lines 15-19) in the trajectory. The RMSprop optimizer is employed to update the parameters, and an adaptive gradient clipping method is used to enhance the training stability [35].

### *C. Deep Deterministic Policy Gradient-based Model Training*

REINFORCE is an on-policy training method that updates parameters only after finishing a whole JCCSP trajectory and can only use the data recorded in the just experienced trajectory, which may result in low data utilization and inefficient parameter updating. Meanwhile, the actor policy network needs to be updated after finishing each trajectory, resulting in too frequent but may inefficient updates of policy installed on the edge service controller in actual implementations. In contrast, off-policy like DDPG can re-utilize not only the data of experience that has just been experienced but even that of the earlier experience. Meanwhile, the parameters can be updated at any time without waiting for the finish of a whole trajectory, which significantly improves the training efficiency and data utilization. Besides, a resource-sufficient cloud server can conduct the off-policy training process. Moreover, in TD DDPG [36], delayed update on actor policy network is employed to improve the training stability, which can reduce the frequency of updating the actor policy model installed on the edge service controller. Therefore, we further propose an off-policy training algorithm for the JCCSP problem based on a modified TD DDPG method.

As discussed above, the objective of DRL is training a policy that can maximize the expected accumulated reward from any given initial state. The expected accumulated reward of any given state under a given policy is also named the state-action values (i.e., Q-value). It depends on the actions taken following the policy $\pi_\theta$ and the state transition only depends on employed actions. According to the Bellman equation,

$$\begin{aligned} Q^{\pi}(s_t, a_t) &= r(s_t, a_t) + \gamma \mathbb{E}_{\{s,\pi_\theta\}}[V(s_{t+1}|\pi)] \\ &= r(s_t, a_t) + \gamma \mathbb{E}_{a_{t+1}\sim\pi_\theta}[Q^{\pi}(s_{t+1}, a_{t+1})], \end{aligned} \tag{19}$$

where $V(s_{t+1}|\pi)$ represents the state value of state $s_{t+1}$ when the system is controlled according to policy $\pi$, which is the expectation of state-action value. Then under the deterministic policy method, the training objective is to maximize the state-action value, i.e.,

$$\mathcal{L}(\theta) = \mathbb{E}_{(s,a,r,d)\sim\mathcal{D}}[Q(s, \pi_\theta(s))], \tag{20}$$

where $(s, a, r, d) \sim \mathcal{D}$ represents the experiences sampled from the replay buffer which records the interaction experiences with the IoT system over the training process.

The Q-value (i.e., state-action value) is estimated by a critic network as shown in Fig. 2. The parameters of the critic network can be updated via the policy gradient method by minimizing the Bellman residential loss, i.e.,

$$\mathcal{L}(\psi) = \mathbb{E}_{\mathcal{D}}[(Q(s_t, a_t) - y_t)^2], \tag{21}$$

where

$$y_t = r(s_t, a_t) + \gamma Q(s_{t+1}, \pi_\theta(s_{t+1})), \tag{22}$$

represents the estimation of Q-value composed of the real immediate reward $r(s_t, a_t)$ and the discounted Q-value of the next state. $\psi$ represents the parameters of the critic network. To improve training stability, twin critic/Q networks are employed in TD DDPG method to reduce the overestimation of Q-values by selecting the minimum value of two Q-values separately estimated by two critic networks. Meanwhile, in order to improve training stability, the Q-value of the next state (i.e., $s_{t+1}$) is estimated by target critic networks ($\tilde{Q}_{\tilde{\psi}_i}$), i.e.,

$$y_t = r(s_t, a_t) + \gamma \min(\{\tilde{Q}_{\tilde{\psi}_i}(s_{t+1}, \tilde{a}_{t+1})\}|_{i=1,2}), \tag{23}$$

where $\tilde{a}_{t+1}$ is the action for state $s_{t+1}$ generated by the target actor network with policy $\tilde{\pi}_{\tilde{\theta}}$. However, such a Q estimation method still suffers from estimation variation and bias at the later training stage in some scenarios. Inspired by the motivation of average DQN [37], [38], we propose a weight-averaged twin-Q-delayed (WATQD) method, introducing averaged Q-values to reduce estimation bias, and the delayed estimation of Q-values to mitigate the bias caused by estimating mutation. The calculation of $y_t$ in the WATQD method is rewritten as

$$y_t = r_t + \gamma[\alpha \min(\tilde{Q}_{\tilde{\psi}_i}) + \frac{(1-\alpha)}{2\Delta} \sum_{\delta=0}^{\Delta-1} \tilde{Q}^{\delta}_{\tilde{\psi}_i}]_{i=1,2}, \tag{24}$$

where $\tilde{Q}^{\delta}_{\tilde{\psi}_i}$ represents the target critic networks with the earlier parameters of $\delta$ steps before the current update step, which is cached by the edge service controller. When $\delta = 0$, $\tilde{Q}^{\delta}_{\tilde{\psi}_i}$ represents the current updated target critic networks (i.e., $\tilde{Q}^{0}_{\tilde{\psi}_i} \equiv \tilde{Q}_{\tilde{\psi}_i}$). $\Delta - 1$ is the number of cached previous twin critic networks. Besides $\alpha$ is the balance weight factor used to control under estimate and overestimate of Q-values.

The whole training process based on the WATQD DDPG approach is illustrated in Algorithm 3. For clear expression and to maintain a fair comparison with the above RENFORCCE-based approach, in this paper, we set the update of parameters (lines 19-34) executed after each time of finishing a whole JCCSP trajectory of multiple arrived applications (lines 5-18). However, in actual implementation, the parameter update can be triggered at any time to accelerate the training process. Besides, the subscribe symbol $t$ in the parameter update steps (lines 19-30) is used to index the state, action, and reward of each sampled experience, which is not the same as that used in the JCCSP process performed by the actor (lines 6-18). Since the original DDPG algorithm is designed for problems with continuous action and the update of actor parameters requires the derivable of actions, we use the probabilities generated

**Algorithm 3:** Training based on WATQD DDPG

```
Input: IoT system, an actor network ($\pi_\theta$), twin critic
       networks ($Q_{\psi_1}, Q_{\psi 2}$), a target actor network
       ($\tilde{\pi}_\theta$), target twin critic networks ($\tilde{Q}_{\tilde{\psi}_1}, \tilde{Q}_{\tilde{\psi}_2}$), a
       replay buffer, and the $\epsilon$-greedy exploration
       parameters ($\epsilon_{\min}, \epsilon_{\max}, \epsilon_d$)
Output: The trained JCCSP Policy network.
1  Initialized a buffer to cache target critic networks
2  for $n = 1, 2, \cdots$ do
3      System initialization
4      $\epsilon = \epsilon_{\min} + (\epsilon_{\max} - \epsilon_{\min}) e^{-\frac{n}{\epsilon_d}}$
5      $\mathbb{A} \leftarrow$ A set of applications to be placed.
6      for $t = 1, 2, \cdots, |\mathbb{A}|$ do
7          $s_t = \{\boldsymbol{s}^{\boldsymbol{H}}_t, \boldsymbol{s}^{\boldsymbol{a}}_t\}$; $d_t = 0$; $\epsilon_s$ = Random(0, 1)
8          if $\epsilon_s > \epsilon$ then
9              Obtain $a_t$ by Algorithm 1 w/o exploration
10         else
11             Obtain $a_t$ by Algorithm 1 with exploration
12         $\hat{a}_t = \arg\max_h a_t$
13         $r_t \leftarrow$ Reward obtained after executing $\hat{a}_t$
14         if $r_t == 0$ then
15             $d_t = 1$; break
16         else
17             Update system to $s_{t+1}$
18         Store $(s_t, a_t, r_t, s_{t+1}, d_t)$ to replay buffer
19     if $n > N_c$ then
20         Update critic networks:
21         $\mathcal{D} = \{s_t, a_t, r_t, s_{t+1}, d_t\}$          // Sampled experiences
22         $\tilde{a}_{t+1} = \tilde{\pi}_{\tilde{\theta}(s_{t+1})}$
23         $y_t = r_t + \gamma(1-d_t)[\alpha \min(\tilde{Q}_{\tilde{\psi}_i}) + \frac{(1-\alpha)}{2\Delta} \sum_{\delta=0}^{\Delta-1} \tilde{Q}^{\delta}_{\tilde{\psi}_i}]]$
24         $\mathcal{L}(\psi) = \sum_i [\mathbb{E}_{\mathcal{D}}[(Q_{\psi_i}(s_t, a_t) - y_t)^2]]$   // $i = 1, 2$
25         $\psi_i$ = RMSprop($\nabla_{\psi_i}(\mathcal{L}(\psi))$)
26         if $n \% \lambda == 0$ then                    // Delayed update
27             Update actor network:
28             $\mathcal{D} = \{s_t, a_t, r_t, s_{t+1}, d_t\}$  // Sampled experiences
29             $\mathcal{L}(\theta) = \mathbb{E}_{\mathcal{D}}[-Q_{\psi_1}(s_t, \pi_\theta(s_t))]$
30             $\theta$ = RMSprop($\nabla_\theta(\mathcal{L}(\theta))$)
31             Soft update target networks:
32             $\tilde{Q}^{\delta}_{\tilde{\psi}_i} = \tilde{Q}^{\delta-1}_{\tilde{\psi}_i}$ // Update cached target critic networks
33             $\tilde{\theta} = \mu\tilde{\theta} + (1-\mu)\theta$
34             $\tilde{\psi}_i = \mu\tilde{\psi}_i + (1-\mu)\psi_i|_{i=1,2}$
```

by the softmax function as the action for policy training (i.e., $\boldsymbol{P}_j$) and index with the maximum probability as the action executed in a real IoT system (line 16 in Algorithm 1), i.e.,

$$\hat{a}_t^j = \arg\max_h \boldsymbol{P}_j. \tag{25}$$

It should be noted that, to simplify expression, $a_t$ in Algorithm 3 is equivalent to the $\mathcal{P}_t$ obtained from performing Algorithm 1. Besides, clipped Gaussian noise is introduced in the action selection of TD DDPG to enhance exploration. However, for our problem with discrete action space, clipped Gaussian noise may cause the probabilities of selecting multiple ES to be 1, which would confuse the final action selection operation. Meanwhile, we also want the sampled actions to follow the probability distribution of original actions when the actor policy has been trained to be sufficiently deterministic. Therefore, we employ the $\epsilon$-greedy exploration method by adding Gumbel noise [39] on the output action $\boldsymbol{P}_j$ (line 4 and lines 8-11). When executing exploration (line 11), a new $\boldsymbol{P}_j$ after line 15 in Algorithm 1 is calculated as

$$\boldsymbol{P}_j = \text{softmax}(\log(\boldsymbol{P}_j) + \mathcal{N}), \tag{26}$$

where $\mathcal{N}$ is a set of Gumbel noise generated based on an uniform distribution $U_i \sim U(0,1)$, i.e.,

$$\mathcal{N}_i = -\log(-\log(U_i)), U_i \sim U(0,1), i \in [1, |\mathcal{H}|]. \tag{27}$$

Otherwise, $\boldsymbol{P}_j$ is calculated following Algorithm 1 without (w/o) exploration. Before updating parameters, the system first runs for $N_c$ episodes for data collection. After that, the citric networks are updated every episode. A mini-batch data of experienced states, actions, rewards, and state transitions are sampled from the replay buffer (line 21). Then, all sampled next states ($s_{t+1}$) are processed by the target actor policy network ($\tilde{\pi}_{\tilde{\theta}}$) with the latest parameters to generate new actions for each next state (line 22). Then, $y_t$ is calculated in line 23 with these new generated actions, and loss values of critic networks are calculated in line 24. After that, parameters of the critic networks are updated by gradient descent algorithm with the RMSprop optimizer. The actor and target networks are delayed updated, i.e., update one time after every $\lambda$ training episodes. The parameters of the actor policy network are updated by maximizing the expected Q-value of all sampled states and corresponding actions regenerated by the actor with the latest parameters. The target networks employ a soft update method (i.e., lines 33-34), where $\mu$ is the soft update coefficient and $\mu < 1$ and usually employ a value very close to 1. Before the soft update of target networks, the cached target critic networks are updated by removing the earliest cached one and adding the latest one (line 32).

## V. Performance Evaluation

### A. Simulation Setup

We conduct simulations to evaluate our proposed DRL-based JCCSP approaches depend on the Pytorch 1.81 environment. The topology data of the concerned edge-enabled IoT system is from an online one[1]. In the IoT system, we set 33 nodes as RANs. Meanwhile, seven randomly created ESs with different resource capacities are deployed to 7 arbitrary selected RANs marked by orange in Fig. 3. Besides, 50 sensors are also randomly placed in the system. The detailed information of these ESs and sensors is listed in Appendix A. We use the Dijkstra algorithm to configure the routing rules among RANs. The capacity of each link is set to be 1000 Mbps. In Encoder, we employ three-layer LSTM cells and the bidirectional setting. The embedding size is 128, and the hidden size of LSTM and neural networks is set to be 64. The learning rates for REINFORCE-based and WATQD DDPG-based algorithms are set to be 0.001 and $1e^{-4}$, respectively. Meanwhile, the learning rates are reduced to 0.1 times of initial settings after 2000 episodes to improve training stability at the later training stages. The delayed update frequency ($\lambda$) in WATQD DDPG is 20, and $\mu = 0.999$. We randomly generate 40 applications at each training episode as the IoT system is resource-limited. Meanwhile, each application are randomly generated following the parameters detailed in Table I. The values of resources (i.e., computing and storage) used in this paper are all normalized values, which can represent the number of resource blocks in actual systems.

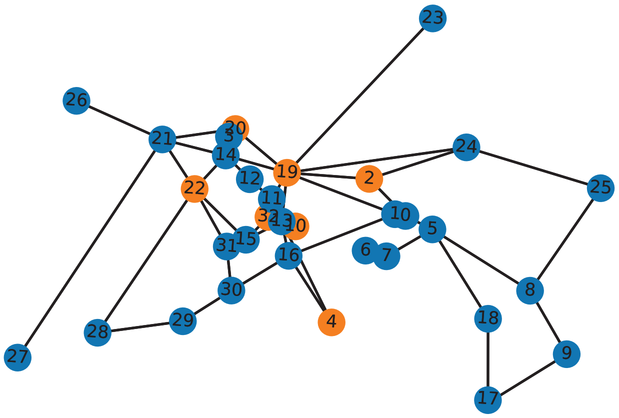


Fig. 3. Network topology for simulation.

TABLE I
SIMULATION PARAMETERS

| Parameter | Value | Parameter | Value |
|---|---|---|---|
| $\lvert\mathcal{K}\rvert$ | [3, 10] | $W_i$ | [1, 5] |
| $\tau_i$ | [300, 400] ms | $d_i^P$ | [50, 80] ms |
| $d_i^T$ | [15, 20] ms | $d_i^R$ | [2, 10] ms |
| $d_{i,k}^C$ | [10, 15] ms | $b_{i,k}^C$ | [10, 20] Mbps |
| $b_i^T$ | [20, 30] Mbps | $b_i^R$ | [2, 8] Mbps |
| $b_{i,k}^M$ | [5, 10] Mbps | $\beta_1$ | 10 |
| $c_i^1$ | [5,15]×10 | $s_i^1$ | [2, 8] |
| $c_i^k, (k > 1)$ | [2, 8] | $s_i^k, (k > 1)$ | [2, 8]×3 |

We compare our proposed DRL-based approaches with two benchmarks which can also perform JCCSP without the prior knowledge of the underlay communication networks.

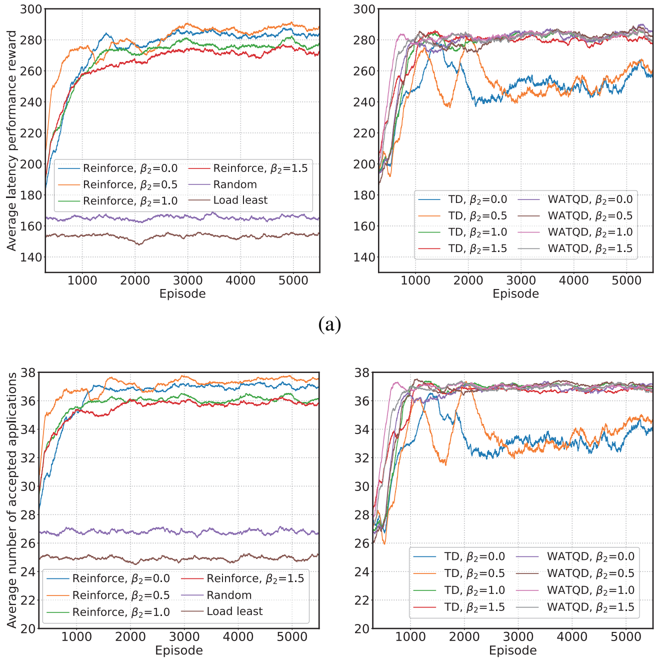
**Random:** The controller randomly selects an ES to place a service of an application. Besides, only ESs whose available resources can satisfy the service requirements are selected as candidates to postpone breaking the resource constraint.

[1] http://www.topology-zoo.org/files/Bics.gml

**Load least:** The controller selects the ES with minimum resource consumption ($h^*$) to place the current $j^{th}$ service of $\mathcal{A}_t$ based on the minimum resource consumption ratio after placing the service, i.e.,

$$h^* = \arg\min_{h\in\mathcal{H}} \max\{\frac{C_o^h + c_t^j}{C_h}, \frac{S_o^h + s_t^j}{S_h}\}.$$

### B. Simulation Results

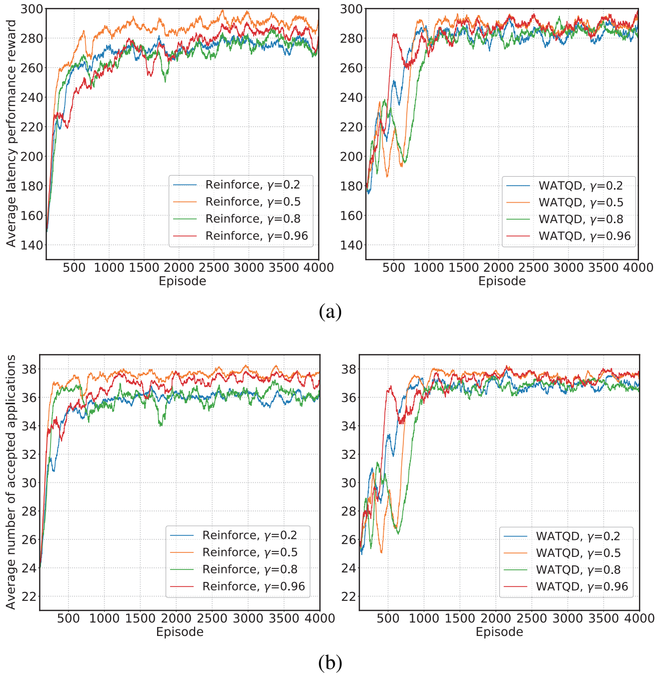


(a)

(b)

Fig. 4. Performance over the training process ($\gamma$= 0.9, $\alpha$ = 0.8). (a) The average latency performance reward of accepted applications. (b) The average number of accepted applications.

Fig. 4 shows the training convergence and performances over the training process under different reward function balance factors. We set $\gamma = 0.8$ and the WATQD DDPG weight factor $\alpha = 0.9$. We used the simple average method to process the results of every 300 training episodes. Fig. 4 reveals that the load least algorithm does not perform better than the random method considering the resource constraint of ESs. They have the lowest performance since they only consider the resource consumption of services and resource capacity constraints of ESs. However, the DRL-based algorithms can learn to adapt the system by comprehensively considering the resource consumption of ES and applications' features in resource consumption and QoS. We can find that, after training, DRL-based approaches can all achieve converged performances that are significantly higher than those achieved before training. Besides, the training stability of TD DDPG (i.e., $\alpha = 1$) and the achieved performances are lower than that of WATQD DDPG in conditions with $\beta_2 = 0$ and $\beta_2 = 0.5$. Moreover, compared with WAQTD DDPG, the REINFORCE-based approach is more sensitive to the balance factor $\beta_2$ of the reward function. In the REINFORCE-based approach, we can find that a little incentive reward from the number of accepted applications can help obtain higher performance, while too high may result in lower convergence performance.



(a)

(b)

Fig. 5. Impact of the discount factor on the convergence ($\beta_2 = 1$, $\alpha = 0.5$). (a) The average latency performance reward of accepted applications. (b) The average number of accepted applications.

Fig. 5 exhibits the impact of the discount factor on the convergence performance in terms of the average latency performance reward and the average number of accept applications in the condition of $\beta_2 = 1$ and WATQD $\alpha = 0.5$. The simulation results are moving averaged every 100 training episodes. In each sub-figure, the left side shows the performance achieved by the REINFORCE-based approach, and the right side shows that obtained by the WATQD DDPG approach. Fig. 5 shows that DRL-based algorithms can always converge under different discount factors after training. However, the achieved coverage performance is affected by the value of the discount factor. Compared with the training processes of the REINFORCE-based approach, the WATQD DDPG method performs a little more stable as it can achieve similar performances under different discount factors. Moreover, we can observe that WATQD can reach the optimal convergence performance faster than the REINFORCE-based process.

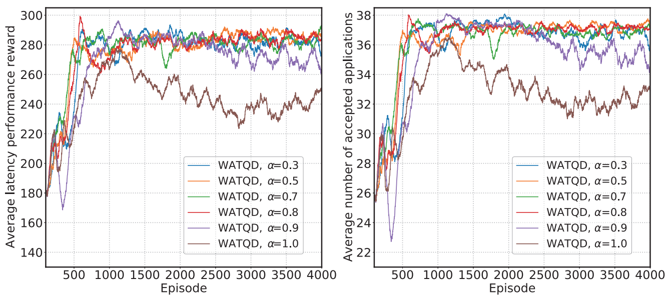


Fig. 6. Impact of the WATQD weight factor $\alpha$ ($\gamma = 0.95, \beta_2 = 1$).

Fig. 6 shows the convergence performance under different WATQD weight factors with the setting of $\gamma = 0.95$ and $\beta_2 = 1$. We can observe that the training process is unstable and performance reduced after reaching a sub-optimal performance when $\alpha = 1$, i.e., the TD DDPG. However, when using the WATQD method, the training stability is significantly improved, and the achieved performances can converge to higher values. After 2000 training episodes, the performance under the condition with $\alpha = 0.9$ starts to reduce. This phenomenon reveals that a WATQD can increase the stability of the training process, but an excessively high balance factor may cause the WATQD approach to behave similarly to the TD algorithm. Moreover, the performance achieved in the condition with $\alpha = 0.3$ exposes a little higher instability than that in other conditions satisfying $\alpha < 0.9$. The potential reason is that the TD method employs the minimum value as the estimated Q-value, which can significantly reduce the overestimation of the Q-value. When employing a smaller weight factor $\alpha$ in WATQD, the average value contributes more to the estimated Q-value, which increases the probability of Q-value overestimation. Thus, the training instability increased, although the delayed update on Q-values has mitigated estimation bias. When setting $\alpha$ to be 0.5, 0.7, and 0.8, the system achieves approximate converge results, and the training stability is significantly improved.

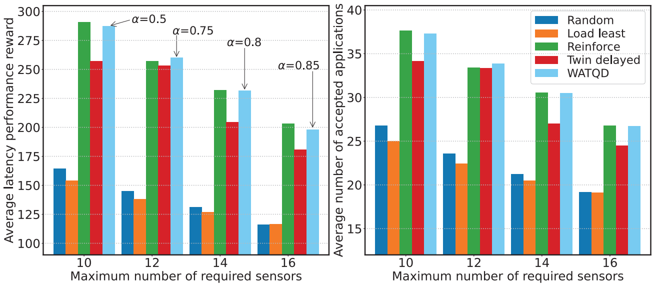


Fig. 7. Impact of the maximum number of sensors required by applications.

Fig. 7 shows the performance variation versus the maximum number of sensors required by applications, which affects the lengths of input application state sequence and corresponding JCCSP action size. The WATQD weight factor employed in each scenario is labeled on the left sub-figure. Moreover, the discount factors used in REINFORCE-based and WATQD DDPG approaches are separately set to be 0.5 and 0.9. The parameters adopted by the twin-delayed (TD) algorithm in each scenario are the same as those set by the WATQD algorithm. We averaged the results of the subsequent 500 training episodes starting from the $3000^{th}$ training episode. Fig. 7 shows that performances obtained by all algorithms are reduced as the maximum number of required sensors increases since more resources are consumed by the transmission and caching of more sensing data as well as the running of CFs. Besides, all DRL-based algorithms can achieve significantly higher performance than the random method and the load least algorithm. Besides, compared with the twin-delayed DDPG approach, the REINFORCE-based approach and the WATQD DDPG approach can achieve higher performance rewards and accept more applications in most scenarios. The REINFORCE-based and the WATQD DDPG approaches can achieve approximate performances and are significantly higher than that obtained by other methods, demonstrating the advantage of proposed DRL-based JCCSP approaches for multiple sensing-data-driven IoT applications in edge-enabled IoT systems.

## VI. Extension and Future Works

### A. *Dynamic Refreshment of Service Placement*

This paper considers a long-term stable environment in which multiple sensing-data-driven IoT applications will be performed in the long term after their related SF and CFs have been placed on ESs. In some IoT systems, some applications may only be performed for a period of time. Besides, new applications may be triggered during system running. In such a dynamic IoT system, related sensors can be notified by the controller to stop caching data to CFs of a finished application, and the occupied resources can be freed and spared to support future applications. The system state defined in our work only consists of real-time resource consumption of ESs and an application to be placed, which is easily available to the edge service controller. Therefore, when there are new triggered applications, and some have been finished, the edge service controller can still perform the well-trained DRL model to conduct JCCSP operation for new applications without interfering with the executing applications. However, the JCCSP for some applications is implemented under the condition that some completed applications were still performing, which makes these JCCSP actions possibly not optimal in new system conditions, and may cause performance degradation after a period of time. Therefore, the refreshment of service placement policy dynamically is necessary to maintain performance in a dynamic IoT system. However, it requires a well-designed framework for implementing the refreshment of service placement and considers the cost introduced by the refreshment. This dynamic refreshment of service placement in dynamic IoT systems will be investigated in our future work.

### B. *Duplicated Sensing Data Requirement*

This paper sets to cache the sensing data from a sensor required by each application on its dedicated CF. However, in some conditions, some of the necessary sensing data may be simultaneously required by different applications, which makes them duplicated cached on ESs. Then, if we reduce the number of CFs caching the same sensing data while satisfying the requirement, the resources consumed by caching these duplicate data can be reduced and spared for accepting more applications. The sensing-data request relationship among different applications should be investigated in such a scenario. Besides, whether the required sensing data has been configured to be cached by a previously placed CF should be considered when conducting the JCCSP process of an application. This duplicated sensing data requirement condition will be studied in our future work.

## VII. Conclusion

This paper investigated the JCCSP problem for multi-sensing-data-driven IoT applications in an edge-enabled IoT system. The problem is first formulated as an MDP with finite steps to maximize the accumulated reward of JCCSP actions. Then, to deal with difficulty caused by system heterogeneity and limited prior knowledge, DRL is explored to address the considered JCCSP problem. The actor policy network is constructed based on the encoder-decoder model to satisfy the requirement of input system state and JCCSP action with variable sizes. An on-policy REINFORCE-based training policy is employed to train the policy network. After that, to improve the experience utilization and training efficiency, we also proposed an off-policy training method based on twin-delayed DDPG. A WATQD method is proposed in the DDPG-based training method to improve training stability. Finally, extensive simulation results show that the proposed DRL-based approaches outperform the benchmarks by comparing system performance in terms of the number of accepted applications and latency performance reward. Meanwhile, the proposed WATQD method can improve the training stability compared with the original TD method.

## Appendix A
## Details about the simulation environment

Table II provides the information of randomly created ESs, where the first row gives the indexes of 7 ESs, the second row lists the IDs of each RAN associated with the ES above, the third and fourth rows give the maximum computing and storage capacities of the ES, respectively.

TABLE II
Properties of ESs

| ES | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| RAN | 22 | 10 | 4 | 19 | 32 | 2 | 20 |
| Computing | 970 | 764 | 766 | 843 | 761 | 602 | 791 |
| Storage | 521 | 600 | 949 | 573 | 617 | 729 | 559 |

Table III lists the IDs of 50 sensors indexed from 0 to 49, each of which is associated with one randomly selected RAN from the 33 RANs indexed from 0 to 32 in the system.

TABLE III
Association relationship between sensors and RANs

| Sensor | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| RAN | 14 | 5 | 3 | 2 | 30 | 5 | 9 | 1 | 20 | 24 |
| Sensor | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
| RAN | 22 | 24 | 24 | 24 | 15 | 5 | 22 | 20 | 13 | 28 |
| Sensor | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 |
| RAN | 25 | 6 | 9 | 14 | 4 | 5 | 28 | 24 | 12 | 29 |
| Sensor | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 |
| RAN | 30 | 17 | 2 | 13 | 1 | 19 | 31 | 2 | 3 | 15 |
| Sensor | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 |
| RAN | 30 | 13 | 14 | 30 | 11 | 5 | 13 | 22 | 25 | 9 |

## References

[1] S. Guo, Y. Dai, S. Xu, X. Qiu, and F. Qi, "Trusted cloud-edge network resource management: Drl-driven service function chain orchestration for iot," *IEEE Internet of Things Journal*, vol. 7, no. 7, pp. 6010–6022, July 2020.

[2] P. Porambage, J. Okwuibe, M. Liyanage, M. Ylianttila, and T. Taleb, "Survey on multi-access edge computing for internet of things realization," *IEEE Communications Surveys Tutorials*, vol. 20, no. 4, pp. 2961–2991, 2018.

[3] S. Krishnendu, B. N. Bharath, and V. Bhatia, "Cache enabled cellular network: Algorithm for cache placement and guarantees," *IEEE Wireless Communications Letters*, vol. 8, no. 6, pp. 1550–1554, 2019.

[4] S. Yang, S. Fan, G. Deng, and H. Tian, "Local content cloud based cooperative caching placement for edge caching," in *2019 IEEE 30th Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, 2019, pp. 1–6.

[5] L. Chen, L. Song, J. Chakareski, and J. Xu, "Collaborative content placement among wireless edge caching stations with time-to-live cache," *IEEE Transactions on Multimedia*, vol. 22, no. 2, pp. 432–444, 2020.

[6] X. Xia, F. Chen, Q. He, J. Grundy, M. Abdelrazek, and H. Jin, "Online collaborative data caching in edge computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 2, pp. 281–294, 2021.

[7] L. Zhao and J. Liu, "Optimal placement of virtual machines for supporting multiple applications in mobile edge networks," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 7, pp. 6533–6545, 2018.

[8] Q. Fan and N. Ansari, "Application aware workload allocation for edge computing-based iot," *IEEE Internet of Things Journal*, vol. 5, no. 3, pp. 2146–2153, 2018.

[9] K. Poularakis, J. Llorca, A. M. Tulino, I. Taylor, and L. Tassiulas, "Joint service placement and request routing in multi-cell mobile edge computing networks," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, 2019, pp. 10–18.

[10] L. Chen, C. Shen, P. Zhou, and J. Xu, "Collaborative service placement for edge computing in dense small cell networks," *IEEE Transactions on Mobile Computing*, vol. 20, no. 2, pp. 377–390, 2021.

[11] S. Pasteris, S. Wang, M. Herbster, and T. He, "Service placement with provable guarantees in heterogeneous edge computing systems," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, 2019, pp. 514–522.

[12] H. Badri, T. Bahreini, D. Grosu, and K. Yang, "Energy-aware application placement in mobile edge computing: A stochastic optimization approach," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 4, pp. 909–922, 2020.

[13] W. Fan, Y. Liu, B. Tang, F. Wu, and H. Zhang, "Terminalbooster: Collaborative computation offloading and data caching via smart basestations," *IEEE Wireless Communications Letters*, vol. 5, no. 6, pp. 612–615, 2016.

[14] A. Ndikumana, N. H. Tran, T. M. Ho, Z. Han, W. Saad, D. Niyato, and C. S. Hong, "Joint communication, computation, caching, and control in big data multi-access edge computing," *IEEE Transactions on Mobile Computing*, vol. 19, no. 6, pp. 1359–1374, 2020.

[15] S. Yu, R. Langar, X. Fu, L. Wang, and Z. Han, "Computation offloading with data caching enhancement for mobile edge computing," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 11, pp. 11 098–11 112, 2018.

[16] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. Cambridge, MA, USA: MIT press, 2018.

[17] J. Pei, P. Hong, M. Pan, J. Liu, and J. Zhou, "Optimal vnf placement via deep reinforcement learning in sdn/nfv-enabled networks," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 2, pp. 263–278, 2020.

[18] I. Grondman, L. Busoniu, G. A. D. Lopes, and R. Babuska, "A survey of actor-critic reinforcement learning: Standard and natural policy gradients," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, no. 6, pp. 1291–1307, 2012.

[19] L. Lei, Y. Tan, K. Zheng, S. Liu, K. Zhang, and X. Shen, "Deep reinforcement learning for autonomous internet of things: Model, applications and challenges," *IEEE Communications Surveys Tutorials*, vol. 22, no. 3, pp. 1722–1760, 2020.

[20] Y. Wang, Y. Li, T. Lan, and N. Choi, "A reinforcement learning approach for online service tree placement in edge computing," in *2019 IEEE 27th International Conference on Network Protocols (ICNP)*, Chicago, IL, USA, USA, Oct 2019, pp. 1–6.

[21] D. Shi, H. Gao, L. Wang, M. Pan, Z. Han, and H. V. Poor, "Mean field game guided deep reinforcement learning for task placement in cooperative multiaccess edge computing," *IEEE Internet of Things Journal*, vol. 7, no. 10, pp. 9330–9340, 2020.

[22] S. Yu, X. Chen, Z. Zhou, X. Gong, and D. Wu, "When deep reinforcement learning meets federated learning: Intelligent multitimescale resource management for multiaccess edge computing in 5g ultradense network," *IEEE Internet of Things Journal*, vol. 8, no. 4, pp. 2238–2251, 2021.

[23] A. Laghrissi and T. Taleb, "A survey on the placement of virtual resources and virtual network functions," *IEEE Communications Surveys Tutorials*, vol. 21, no. 2, pp. 1409–1434, 2019.

[24] P. T. Anh Quang, Y. Hadjadj-Aoul, and A. Outtagarts, "Evolutionary actor-multi-critic model for vnf-fg embedding," in *2020 IEEE 17th Annual Consumer Communications Networking Conference (CCNC)*, 2020, pp. 1–6.

[25] P. T. A. Quang, Y. Hadjadj-Aoul, and A. Outtagarts, "A Deep Reinforcement Learning Approach for VNF Forwarding Graph Embedding," *IEEE Transactions on Network and Service Management*, vol. 16, no. 4, pp. 1318–1331, 2019.

[26] S. Wang and T. Lv, "Deep reinforcement learning for demand-aware joint vnf placement-and-routing," in *2019 IEEE Globecom Workshops (GC Wkshps)*, 2019, pp. 1–6.

[27] G. Li, H. Zhou, B. Feng, Y. Zhang, and S. Yu, "Efficient provision of service function chains in overlay networks using reinforcement learning," *IEEE Transactions on Cloud Computing*, vol. 10, no. 1, pp. 383–395, 2022.

[28] R. Solozabal, J. Ceberio, A. Sanchoyerto, L. Zabala, B. Blanco, and F. Liberal, "Virtual network function placement optimization with deep reinforcement learning," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 2, pp. 292–303, 2020.

[29] H. Zhu, Y. Cao, W. Wang, T. Jiang, and S. Jin, "Deep reinforcement learning for mobile edge caching: Review, new features, and open issues," *IEEE Network*, vol. 32, no. 6, pp. 50–57, 2018.

[30] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," in *Advances in Neural Information Processing Systems*, vol. 28. Curran Associates, Inc., 2015, pp. 2692–2700.

[31] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, "Neural combinatorial optimization with reinforcement learning," *arXiv preprint arXiv:1611.09940*, 2016. [Online]. Available: https://arxiv.org/abs/1611.09940

[32] A. Mirhoseini, H. Pham, Q. V. Le, B. Steiner, R. Larsen, Y. Zhou, N. Kumar, M. Norouzi, S. Bengio, and J. Dean, "Device placement optimization with reinforcement learning," in *Proceedings of the 34th International Conference on Machine Learning*. International Convention Centre, Sydney, Australia: PMLR, 06–11 Aug 2017, pp. 2430–2439.

[33] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in *3rd International Conference on Learning Representations, ICLR 2015*, San Diego, United States, May 2015. [Online]. Available: https://arxiv.org/abs/1409.0473

[34] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, no. 3, p. 229–256, 1992.

[35] S. Prem, W. Gordon, P. Bryan, and L. R. Jonathan, "Autoclip: Adaptive gradient clipping for source separation networks," in *2020 IEEE 30th International Workshop on Machine Learning for Signal Processing (MLSP)*, Espoo, Finland, 2020, pp. 1–6.

[36] S. Fujimoto, H. Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *International Conference on Machine Learning*. PMLR, 2018, pp. 1587–1596.

[37] O. Anschel, N. Baram, and N. Shimkin, "Averaged-dqn: Variance reduction and stabilization for deep reinforcement learning," in *International conference on machine learning*. PMLR, 2017, pp. 176–185.

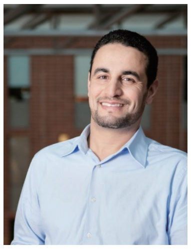
[38] Q. He and X. Hou, "Reducing estimation bias via weighted delayed deep deterministic policy gradient," *arXiv preprint arXiv:2006.12622*, 2020. [Online]. Available: https://arxiv.org/abs/2006.12622

[39] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," *Neural Information Processing Systems (NIPS)*, p. 6382–6393, 2017.

**Yan Chen** received his B.S. degree in Information Engineering from China University of Mining and Technology in 2016. He is currently pursuing his Ph.D. degree in Information and Communication Engineering at the School of Information and Control Engineering, China University of Mining and Technology. He is also a visiting Ph.D. student at the Department of Communications and Networking, School of Electrical Engineering, Aalto University (Dec. 2020∼Dec. 2021) under the support of the Chinese government scholarship awarded by the China Scholarship Council (CSC). His research interests include Edge Computing, Internet of Things, and wireless networks.

**Yanjing Sun** is a Professor in School of Information and Control Engineering, China University of Mining and Technology since July 2012. He received the Ph.D. degree in Information and Communication Engineering from China University of Mining and Technology in 2008. He is also a council member of the Jiangsu Institute of Electronics, a member of the Information Technology Working Committee of the China Safety Production Association. His current research interests include wirless communication, Embedded real-time system, Wireless sensor networks, Cyberphysical system and so on.

**Bin Yang** received his Ph.D. degree in Systems Information Science from Future University Hakodate, Japan in 2015. He is a Professor with the School of Computer and Information Engineering, Chuzhou University, China, and is also a Senior Researcher with MOSAIC LAB, Finland. His research interests include unmanned aerial vehicle networks, cyber security and Internet of Things.



**Tarik Taleb** received the B.E. degree (with distinction) in information engineering and the M.Sc. and Ph.D. degrees in information sciences from Tohoku University, Sendai, Japan, in 2001, 2003, and 2005, respectively. He is the founder and the Director of the MOSA!C Lab, Espoo, Finland. Since Oct. 2018, he has been a Full Professor with the Center of Wireless Communications, University of Oulu, Oulu, Finland. Between Oct. 2014 and Dec. 2021, he was a Professor with the School of Electrical Engineering, Aalto University, Espoo, Finland. Prior to that, he was a Senior Researcher and a 3GPP Standards Expert with NEC Europe Ltd., Heidelberg, Germany. He was then leading the NEC Europe Labs Team, involved with research and development projects on carrier cloud platforms, an important vision of 5G systems. Between 2006 and 2009, he was an Assistant Professor with the Graduate School of Information Sciences, Tohoku University, in a laboratory fully funded by KDDI. From 2005 to 2006, he was a Research Fellow with the Intelligent Cosmos Research Institute, Sendai. He has also been directly engaged in the development and standardization of the Evolved Packet System as a member of the 3GPP System Architecture Working Group. His current research interests include architectural enhancements to mobile core networks (particularly 3GPP's), network softwarization and slicing, mobile cloud networking, network function virtualization, software defined networking, mobile multimedia streaming, and unmanned vehicular communications. Prof. Taleb was a recipient of the 2017 IEEE ComSoc Communications Software Technical Achievement Award in 2017 for his outstanding contributions to network softwarization and the Best Paper Awards at prestigious IEEE-flagged conferences for some of his research work. He was a corecipient of the 2017 IEEE Communications Society Fred W. Ellersick Prize in 2017, the 2009 IEEE ComSoc Asia–Pacific Best Young Researcher Award in 2009, the 2008 TELECOM System Technology Award from the Telecommunications Advancement Foundation in 2008, the 2007 Funai Foundation Science Promotion Award in 2007, the 2006 IEEE Computer Society Japan Chapter Young Author Award in 2006, the Niwa Yasujirou Memorial Award in 2005, and the Young Researcher's Encouragement Award from the Japan Chapter of the IEEE Vehicular Technology Society in 2003. He is a member of the IEEE Communications Society Standardization Program Development Board.