Dependency-Aware Microservice Deployment for Edge Computing: A Deep Reinforcement Learning Approach with Network Representation

Chenyang Wang, Hao Yu*, Xiuhua Li, Fei Ma, *Member, IEEE,* Xiaofei Wang, *Senior Member, IEEE,* Tarik Taleb, *Senior Member, IEEE,* Victor C. M. Leung, *Life Fellow, IEEE*

***Abstract*—The popularity of microservices in industry has sparked much attention in the research community. Despite significant progress in microservice deployment for resource-intensive services and applications at the network edge, the intricate dependencies among microservices are often overlooked, and some studies underestimate the importance of system context extraction in deployment strategies. This paper addresses these issues by formulating the microservice deployment problem as a max-min problem, considering system cost and quality of service (QoS) jointly. We first study the attention-based microservice representation (AMR) method to achieve effective system context extraction. In this way, the contributions of different computing power providers (users, edge servers, or cloud servers) in the network can be effectively paid attention to. Subsequently, we propose the attention-modified soft actor-critic (ASAC) algorithm to tackle the microservice deployment problem. ASAC leverages attention mechanisms to enhance decision-making and adapt to changing system dynamics. Our simulation results demonstrate ASAC's effectiveness, prioritizing average system cost and reward compared to other state-of-the-art algorithms.**

***Index Terms*—Dependency-aware, microservice deployment, edge computing, attention mechanism, deep reinforcement learning, network representation**

An earlier version of this work is published in [1]. This work was supported in part by Science and Technology Innovation Key R&D Program of Chongqing (Grant No.CSTB2023TIAD-STX0035), National NSFC (Grants No. 62372072 and 62072332), Chongqing Research Program of Basic Research and Frontier Technology (Grant No. cstc2022ycjhbgzxm0058), and Regional Science and Technology Innovation Cooperation Project of Chengdu City (Grant No. 2023-YF11-00023-HZ), and the Tianjin Xinchuang Haihe Lab (Grant 22HHXCJC00002). It was supported in part by the European Union's under the 6G-Path project (Grant No. 101139172) and by the AerOS project (Grant No. 101069732). The paper reflects only the authors' views, and the European Commission bears no responsibility for any utilization of the information contained herein.

C. Wang is with the College of Computer Science and Software Engineering, Shenzhen University, and Guangdong Laboratory of Artificial Intelligence and Digital Economy (SZ), Shenzhen, Guangdong, 518132, China. (e-mail: chenyangwang@ieee.org); H. Yu is with the ICTFicial Oy, Espoo, 02130, Finland. (e-mail: hao.yu@ictficial.com); F. Ma is with Tsinghua-Berkeley Shenzhen Institute, Tsinghua University, Shenzhen, China (e-mail: mf.2017@tsinghua.org.cn); X. Li is with the School of Big Data & Software Engineering, Chongqing University, Chongqing, 401331 China (e-mail: lixiuhua1988@gmail.com); X. Wang is with the College of Intelligence and Computing, Tianjin University, Tianjin, 300072 China (e-mail: xiaofeiwang@tju.edu.cn); T. Taleb is with the Faculty of Electrical Engineering and Information Technology, Ruhr University Bochum, Bochum, 44780, Germany (e-mail: tarik.taleb@rub.de); V. C. M. Leung is with the College of Computer Science & Software Engineering, Shenzhen University, Shenzhen, China 518060, and with the Department of Electrical and Computer Engineering, The University of British Columbia, Vancouver, BC, Canada V6T 1Z4 (e-mail: vleung@ieee.org).

Hao Yu* is the corresponding author (e-mail: hao.yu@ictficial.com).

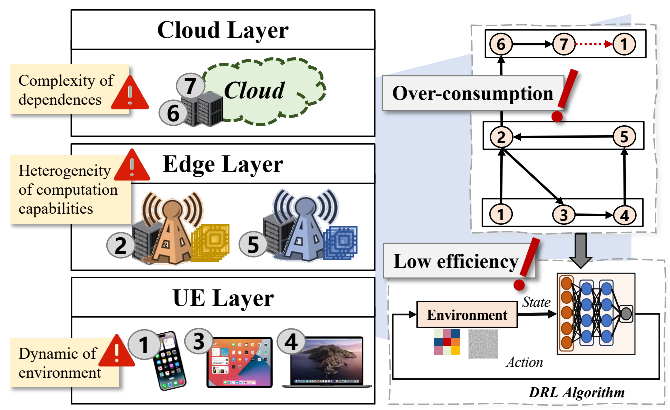


Fig. 1. Architecture overview, traditional solutions and challenges.

I. Introduction

The unprecedented expansion of intelligent devices, coupled with the imperative to facilitate the emergence of 5G and subsequent communication technologies, heralds a proliferation of services and applications of considerable demand, such as face recognition, human-robot interactions, and 3D gaming [2]–[4], *et al*. These applications, characterized by their extensive computational demands or sensitivity to latency, voracious demand for resources, and the necessity for swift responses, present formidable challenges. Certain studies [5]–[7] advocate for the service's deployment on cloud servers, leveraging advantages such as rapid elasticity, on-demand resource pooling, and autonomous configuration. Nonetheless, transmitting data from devices to distant cloud centers often incurs unpredictable latency and excessive consumption of network resources. To address these challenges, it is crucial to design revolutionary network architecture and service deployment technologies for the next generation of mobile networks.

Edge Computing (EC) potentially leverages computation resources in proximity to data sources, *e.g.,* the Base Station (BS) and User Equipment (UE), reducing backhaul usage and elevating data access efficiency [8]. Some studies [9]–[11] investigate the optimization strategies of service deployment under cloud-edge architecture by achieving the minimum service completion time. However, conditions like dynamic system environments, heterogeneity of computation capabilities and complexity of dependencies are not always fully investigated, resulting in consequences related to **over-consumption of**

network resource and **low efficiency of decision-making on deployment strategy**, as shown in Fig. 1.

(i) Over-consumption of network resources arises from the redundant system information in decision-making. Typically, the hierarchy of the EC networks is sophisticated, including the UE layer, edge layer and cloud layer, and the multiple microservices of an application with a dependent sequence of the execution can be deployed in the same layer or across the layers. To model the inter-dependency between different microservices, traditional methods leverage the directed acyclic graph (DAG) to encode the dependency information. However, when solving the optimal microservice deployment problem in terms of the task completion time, features such as the topology, size, and type of microservices [12], [13] are usually overlooked. For instance, task completion time changes with different sizes of microservices, even for the same microservice deployed in several locations, the task execution time will vary. On the other hand, the infrastructure conditions (*e.g.,* the waiting queue state of the edge server) are always changing under the dynamic environment, leading to the heterogeneity of the computation capabilities of the system. Some studies employ the deep reinforcement learning (DRL) method [14], [15] to obtain the microservice deployment strategy by learning the dynamics of the network resources. However, most existing methods require the transfer of a huge amount of data, e.g., dependency information between microservices, changing edge server state, for training, resulting in the over-consumption of network resources.

(ii) Low efficiency of the decision-making is a consequence of complex and high-dimensional states and action spaces. On one hand, the structure of the microservice DAG and the microservice profiles and dynamic scenarios in hierarchical EC networks play an paramount role in the overall deployment of microservices, and the decision-making of the deployment strategy needs to adapt to various conditions [16]–[18] (*e.g.,* the size/type of a microservice, edge server state). On the other hand, the system context abstracting, e.g., and the microservice deployment are coupled and will affect each other, making it challenging to find the optimal solution for both decisions. For the studies using DRL methods, current decisions of system information capturing will make the microservice deployment to be solved in a higher time complexity. In this way, the system faces difficulties in handling environments characterized by complex and high-dimensional states and action spaces, resulting in lower efficiency.

In this paper, we consider a three-layer *UE-Edge-Cloud* network architecture and propose the Attention-based Microservice Representation (*AMR*) mechanism and Attention-modified Soft Actor-Critic (*ASAC*) algorithms to address issues of over-consumption and low efficiency by the following two solutions, respectively, as shown in Fig. 2.

- ***System information representation:*** We first study the delay-sensitive and energy-efficient microservice deployment (MSD) problem, which is then formulated as a joint optimization of the overall system cost and the quality of service (QoS) of UEs. To address the problem issued above, the DAG is first used to model the internal execution sequence and the dependency of the microservices, and the embedded information of the infrastructure-service pair (structure) is further injected into the service-chain attention for better learning of the importance of service. Furthermore, an attention mechanism-based microservice representation is carried out, it focuses on the important system semantic information selection within the entire network and avoids the interference of redundant information on the system decision-making.
- ***DRL-based microservice deployment strategy:*** Distinguished from the state-of-the-art methods, we first model the MSD as a Markov Decision Process (MDP) with continuous state space, and the attention-modified soft actor-critic [19] algorithm is utilized to derive optimal decision-making. After the AMR algorithm filters system information, the state and action space can be effectively reduced. At the same time, Soft Actor-Critic (SAC) encourages exploration and enhances the stability and robustness of learning. Additionally, it outputs a probability distribution of actions, suitable for high-dimensional state and action spaces, thereby mitigating the risk of converging to local optima during the training process.

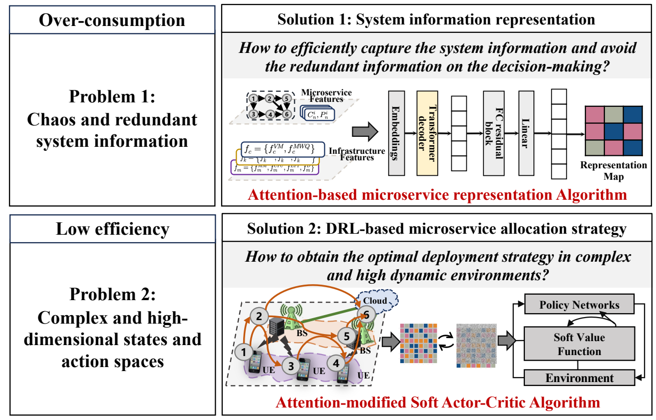


Fig. 2. The issues and solutions of *AMR* and *ASAC* algorithms.

Finally, we evaluate the proposed solution compared to the original SAC (*w/o* attention), Double DQN, Random Allocation, and the ablation algorithms. Our results demonstrate significant performance improvements with the proposed ASAC, increasing the average system reward by 20% to 140% with the episode rising, and around 30% under different MS, UE and ES numbers.

The remainder of this article is organized as follows: Section II surveys the related studies of microservice deployment. In Section III we introduce the system model, including service-DAG and execution models in each layer. Then the optimization problem is formulated in Section IV, and the experimental simulation is conducted in Section VII. Finally, we conclude the article in Section VIII.

II. Related Work

Microservice deployment has been drawing great attention from academics in terms of the benefits of flexibility, loose coupling, and scalability. Especially for computation-intensive and latency-sensitive applications, each microservice can work independently on distributed computing architecture with required computation resources and communication bandwidth.

Traditionally, the microservices were successfully deployed and executed in the cloud server by using virtual machines (VMs) with virtualization technology. To increase the reliability and parallelism of the system, many efforts have been devoted to designing container-aware auto-scaling deployment schemes in cloud servers [20]–[23]. For example, Shihong *et al.* [20] addressed the complex coupling relationship between request scheduling and container retention decisions, and proposed *Onco*, which incorporated the two aspects mentioned above for multiple vehicle services. Literature [24] designed a container-aware strategy with auto-scaling for microservices deployment in the cloud environment. The requested applications were deployed on best-fit lightweight containers, to achieve the minimum deployment time. Other studies investigated the service deployment problem from the view of the virtual network functions (VNFs), *e.g.,* Xiaojun *et al.* [25] examined the challenge of highly available and cost-effective service function chains (SFCs) under edge resource limitations and time-varying VNF failures, they proposed *RAD*, a reliability-aware adaptive deployment scheme to efficiently place and back up SFCs. Similarly, A. Jindal *et al.* [26] addressed the challenge of identifying the capacity for each microservice and evaluated the implementation performance in tool Terminus with four different applications. However, most of the previous studies on the design of microservice deployment strategies mainly focused on cloud-centric solutions, resulting in excessive bandwidth or spectrum consumption when transmitting massive data from the cloud center to the edge server or UE.

Some other studies employed the flexible multi-layer computing architecture to deploy the microservice, *i.e.,* by migrating a certain execution of microservices to the edge of the network, meeting the requirement of fast-response [27]–[30]. M. Alam *et al.* [31] proposed a highly dynamic microservice deployment system with the aid of docker technology and edge computing, the fault tolerance and system availability are dispersed across different layers, to achieve the minimum impact on the overall system performance. Some previous works focused on optimizing the communications delay through dynamic service placement but ignored the effect of access network selection [32]–[34], *e.g.,* Bin *et al.* studied the problem of jointly optimizing the access network selection and service placement for MEC, intending to improve the QoS by trade-off the access delay, communication delay and service switching cost.

Meanwhile, to tackle the challenges raised by the dynamic environment, DRL was an emerging and promising solution where the MDP was employed to model the interactions between users and the environment. For example, the literature [12] introduced an edge-cloud collaborative architecture and proposed a microservice deployment strategy by utilizing a deep deterministic policy gradient to minimize the service access delay. Qiying *et al.* [35] introduced a novel client selection mechanism for leveraging the correlations across local datasets to accelerate the training process, and proposed a neural contextual combinatorial bandit algorithm to establish relationships between client features and rewards, to enable the adaptive selection of client. To extract the relationship among microservices, W. Lv *et al.* [36] employed an undirected weighted interaction graph, *i.e.,* DAG, to reflect the internal dependencies of the EC networks, and proposed a multi-objective microservice deployment strategy based on reward-sharing deep Q-learning, to achieve the minimum communications overhead while balancing the load trade-off. However, the above studies construct the strategy by merely considering a single factor of the system (*e.g.,* energy consumption), ignoring the wealth system context information derived from both the infrastructure and microservice conditions.

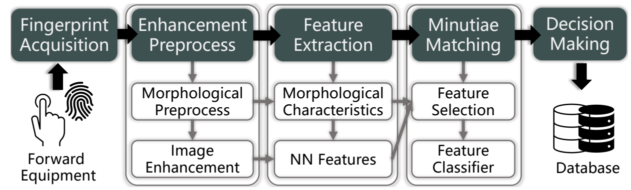


Fig. 3. Illustration of service DAG of fingerprint identification.

III. System Model

In this section, we first introduce the overall three-layer hierarchical architecture with an illustration of service DAG. The execution models on each layer are further demonstrated. Some main notations are summarized in Table I.

A. Architecture Overview

The overall three-layer architecture is shown in Fig. 1 which consists of a cloud data center C in the *Cloud Layer*, edge servers $K = \{1, 2, ..., k\}$ in the *Edge Layer*, and mobile users $M = \{1, 2, ..., m\}$ in the *UE Layer*. Given $M = \{1, 2, ..., m\}$ UE, $S = \{S_1, S_2, ..., S_n\}$ services, each service elaborates a directed acyclic graph (DAG). Denote the service DAG $S_n^D = \{MS_n, \mathcal{E}_n\}$, where $MS_n = \{ms_n^i | i = 1, 2, \cdots, I, \cdots\}$ is the set of microservice of S_n and $\mathcal{E}_n = \{e_n^{ij} | i, j \in \{1, 2, ..., I\}, i < j\}$ represents the set of the internal dependencies of microservices, *i.e.,* the precedence relation such that microservice ms_n^i should be completed before ms_n^j starts. Taking fingerprint identification as an example illustrated in Fig. 3, it contains five sequential processes: fingerprint image acquisition, enhancement preprocessing, feature extraction, minutiae matching, and decision making [37]. The whole process is initiated from the forward equipment (FE), and the dependency exists in the following way: the system is required to complete the ***feature extraction*** before ***minutiae matching*** starts, and finally, the database (DB) receives the service when all the microservices are accomplished. Specifically, we define FE and DB as two virtual microservices ms_n^o and ms_n^{I+1} indicating the entry and exit of service S_1, respectively.

Furthermore, we define each microservice ms_n^i in $\mathbb{S}_n$ is associated with the two-tuples (c_n^i, d_n^i), where c_n^i is the required CPU cycles to finish the microservice ms_n^i and d_n^i denotes the size of the input microservice. Thus, we have $c_n^0 = c_n^{I+1} = 0$, which ensures the place where service S_n starts and ends. We define a deployment variable $\alpha_n^{i,l} = \{0, 1\}$,

$l \in \mathcal{I}$, which indicates that the deploy location is on which device in which layer, and $\mathcal{I} = \{M, K, C\}$ denotes the set of all the infrastructure, *e.g.,* $\alpha_n^{i,K(3)} = 1$ indicating the microservice ms_n^i is deployed on the number 3 edge server of *Edge Layer*, and 0 is otherwise. Recall that $\alpha_n^{0,l} = \alpha_n^{\mathcal{I}+1,l} = 1$, which ensures the beginning and ending deployment of service S_n at FE and DB, respectively.

We have the following definitions to introduce the time and energy consumption of service execution:

Definition 1 (*Ready Time*): The time that the microservice ms_n^i has all the prerequisites (*e.g.,* input data and computation resources) for execution. Let $RT_n^{i,l}$, $l \in \mathcal{I}$ denote the ready time of microservice ms_n^i executed at *UE Layer*, *Edge Layer* and *Cloud Layer*, respectively.

Definition 2 (*Finish Time*): The time that the microservice ms_n^i accomplishes all the workload c_n^i. We define $FT_n^{i,l}$, $l \in \mathcal{I}$ as the finish time of microservice ms_n^i executed at *UE Layer*, *Edge Layer* and *Cloud Layer*, respectively.

Definition 3 (*Wireless Receiving Time*): Accordingly, we define $RT_n^{i,wt,l}$, $FT_n^{i,wt,l}$, $l \in \mathcal{I}$ as the ready time and finish time of the microservice ms_n^i when receiving the wireless channel indicated by the superscript wt from the *Edge Layer* and *Cloud Layer*, respectively.

B. UE Layer Execution Model

In the UE layer execution model, each microservice ms_n^i is deployed on the user equipment. The latency of local execution consists of two parts: 1) The microservice processing time for computing workload on user equipment m; 2) The receiving time of the pre-microservice ms_n^{i-1} if it is deployed at *Edge Layer* or *Cloud Layer*.

Assume that each UE has σ^m cores with the c^m CPU frequency, *i.e.,* the maximal number of a UE microservice processing ability is σ^m. Denote $FT^{\sigma,m}$ the minimum finish time for all microservices in UE m. Furthermore, we have $FT^{\sigma,m} = 0$ when an idle core is assumed in UE m. Then the ready time is calculated as follows:

$$RT_n^{i,m} = \max_{m\prime \in pre\{m\}} g_n^{i,m}, \tag{1}$$

and

$$g_n^{i,m} = max\{FT_n^{i,m\prime}, FT_n^{i,wr,m\prime}, FT_n^{i,wr,k}, FT_n^{i,wr,c}, FT^{\sigma,m\prime}\}, \tag{2}$$

where $pre\{m\}$ is the set of immediate predecessors of microservice ms_n^i, note that the execution of ms_n^i will not start unless all the predecessors have been accomplished due to the internal dependencies.

Accordingly, the local execution time depends on the actual operating frequency c_m^{UE} by $T_n^{i,m} = \frac{d_n^i}{c_m^{UE}}$. Thus, the finish time of microservice ms_n^i at *UE Layer* is $FT_n^{i,m} = RT_n^{i,m} + T_n^{i,m}$. The corresponding energy consumption of microservice ms_n^i at local execution can be obtained as $\epsilon_n^{i,m} = \kappa_m d_n^i (c_m^{UE})^2$ [38], where κ_m is the coefficient related to chip types. Note that we have $\epsilon_n^{0,m} = \epsilon_n^{I+1,m} = 0$ for the FE and DB, respectively.

TABLE I
NOTATION DESCRIPTION

Notation	Description
$RT_n^{i,l}$	Ready time of microservice
$FT_n^{i,l}$	Finish time of microservice
$RT_n^{i,wt,l}$	Ready time receiving from wireless channel
$FT_n^{i,wt,l}$	Finish time receiving from wireless channel
c_n^i	Required CPU for microservice
d_n^i	Size of input microservice
$\alpha_n^{i,l}$	Deployment variable
$q_n^{i,x}$	Indicator function of UE's satisfaction
$\mathbf{w}_x$	Feature embeddings of infrastructure
$\mathbf{w}_y$	Feature embeddings of microservice
$\tau_{xy}^{\mathbf{O}}$	Structure importance on meta-chain $\mathbf{O}$
$\mu_{xy}^{\mathbf{O}}$	Normalized structure attention score function
$\mathbf{w}_x^{\mathbf{O}}$	Infrastructure representation
$\Lambda^{\mathbf{O}_u}$	Final score function of a service-chain $\mathbf{O}$
$\omega^{\mathbf{O}_u}$	Normalized score function of $\Lambda^{\mathbf{O}_u}$
$\mathbf{W}_x$	Final infrastructure embedding
$\mathbf{W}_y$	Final service embedding
$\mathbf{S}^t$	System state space
$\mathbf{A}^t$	System action space
$\mathbf{R}^t(\mathbf{S}^t, \mathbf{A}^t)$	System reward
$\mathcal{H}$	Policy entropy
$\Gamma^\pi Q\left(\mathbf{s}^t, \mathbf{a}^t\right)$	Modified bellman backup operator
D_{KL}	KL divergence operation
$J_Q\left(\varrho\right)$	Soft Q-function
$J_V(\varpi)$	Soft value function
$J_\pi(\varkappa)$	Policy function

C. Edge Layer Execution Model

The microservice ms_n^i is deployed at an edge server k on *Edge Layer*, assume that the user m sends the microservice ms_n^i directly to edge server k via cellular links, denoted by $T_n^{i,w,k} = d_n^i / v_{m,k}$, $v_{m,k}$ is the uplink transmission rate between edge server k and UE m [39]. In this paper, we set the channel gain as $g^{m,k} = -4\ db$ power of the distance between UE m and the edge server k. In this case, the energy consumption of UE m is $\epsilon_n^{i,k} = g^{m,k} \times T_n^{i,w,k}$.

Thus, the ready time on *Edge Layer* can be obtained as follows:

$$RT_n^{i,k} = \max_{m\prime \in pre\{m\}} g_n^{i,k} + T_n^{i,w,k}, \tag{3}$$

and

$$g_n^{i,k} = max\{FT_n^{i,m\prime}, FT_n^{i,wr,m\prime}, FT_n^{i,wr,k}, FT_n^{i,wr,c}\}. \tag{4}$$

Suppose that each edge server equips σ_m^k cores with the c_m^k CPU frequency and the minimum accomplishment time for all the microservices is denoted as $FT_m^{\sigma^k}$. Note that we consider both edge servers and cloud server can satisfy the demand to perform concurrent microservices potentially, thus we have $\sigma_m^k = \infty$. Therefore, the execution time of microservice ms_n^i on *Edge Layer* can be calculated as

$$T_n^{i,k} = \frac{d_n^i}{c_m^k}. \tag{5}$$

Consequently, the finish time of microservice ms_n^i on *Edge Layer* is $FT_n^{i,k} = RT_n^{i,k} + T_n^{i,k}$. The energy of microservice execution in edge server k is $\epsilon_n^{i,k} = \kappa_k d_n^i (c_m^k)^2$.

D. Cloud Layer Execution Model

If the microservice ms_n^i is deployed in the *Cloud Layer*, similar to the execution on *Edge Layer*. We consider that the microservice ms_n^i is first sent to the edge server and then delivered to the cloud server directly via fibre connections, the data transmission delay can be ignored in this way. Thus, the ready time of microservice ms_n^i on *Cloud Layer* can be regraded as

$$RT_n^{i,C} = FT_n^{i,k}. \tag{6}$$

The CPU capable of the cloud is denoted as c^C, and the execution time of ms_n^i is $T_{n,i}^C = d_n^i / c^C$. Accordingly, the energy consumption in *Cloud Layer* can be obtained as $\epsilon_n^{i,C} = \kappa_C d_n^i (c^C)^2$. Therefore, the finish time of microservice ms_n^i is presented as $FT_n^{i,C} = RT_n^{i,C} + T_n^{i,C}$.

IV. Problem Formulation

Recall that the deployment variable $\alpha_n^{i,l} = \{0,1\}$, and $l \in \mathcal{I}$, here we define the average cost of time-varying energy (CTE) ξ^m as follows:

$$\xi^m = \omega_t \times (FT_n^{I+1,l} - FT_n^{0,l}) + \omega_e \times E, \tag{7}$$

where E is the total energy consumption of all devices, ω_t and ω_e holding $\omega_t + \omega_e = 1$, are the coefficients of execution time and energy consumption, respectively.

Besides, the microservices deployment fee (MDF) is ζ^l, $l \in \mathcal{I}$, where $\zeta^m \gg \zeta^k \gg \zeta^C$. Therefore, the overall system cost can be expressed as follows:

$$\mathcal{A} = \sum_{i \in ms_n^i} \sum_{x \in X} \sum_{n \in S_n^D} \left(\alpha_n^{i,l} \zeta^l + \frac{1}{X} \sum_{x=1}^{X} \xi^x \right), \tag{8}$$

where $X \subseteq \mathcal{I}$ is the number of devices executing the microservices, and one of our objectives is to minimize the overall system cost, shown as follows:

$$\min_{\alpha} \quad \mathcal{A} \tag{9}$$
$$s.t. \quad \omega_t + \omega_e = 1 \tag{9a}$$
$$l \in \mathcal{I}, x \in X, \tag{9b}$$
$$\alpha = \{0,1\} \tag{9c}$$
$$\zeta^m \gg \zeta^k \gg \zeta^C \tag{9d}$$

Define $\delta = t_{UE-FE} + t_{ms_n^i}$ is the whole time consumption, where t_{UE-FE} is the communication latency between UE and FE, and $t_{ms_n^i}$ is the serving time, which is regarded as ms_n^i processing time related on the hardware conditions of devices. Furthermore, assume that δ^{max} is the maximal tolerant time of the UE with the demand for the delay-sensitive services, and we denote $q_n^{i,x}$ as the indicator function that satisfying the demand of a UE when executing the microservices.

$$q_n^{i,x} = \begin{cases} 1, & \delta_n^{i,x} < \delta_{max} \\ 0, & otherwise \end{cases}, \tag{10}$$

and we define the quality of service (QoS) of the UE as:

$$\mathcal{B} = \sum_{i \in ms_n^i} \sum_{x \in X} \sum_{n \in S_n^D} \alpha_n^{i,l} q_n^{i,x} \tag{11}$$

The other objective of this work is to maximize the overall QoS for UE in the system,

$$\begin{aligned} \max_{q} \quad & \mathcal{B} \\ s.t. \quad & \alpha = \{0,1\} \\ & q_n^{i,x} \in \{0,1\} \end{aligned} \tag{12}$$

By comprehensively considering both the system cost and the QoS, we can balance different performance metrics crucial for the system's overall efficiency. Finally, the problem of microservices deployment (MSD) can be modelled as the $max-min$ joint optimization problem $\mathcal{Z}(\mathcal{A},\mathcal{B}) = \eta_1 \mathcal{A} + \eta_2 \mathcal{B}$, which ensures that we first minimize the overall system cost, thereby achieving a balanced solution that maximizes the consistent QoS across all UE in the system, shown as follows:

$$\begin{aligned} \max_{q} \min_{\alpha} \quad & \mathcal{Z}(\mathcal{A},\mathcal{B}) \\ s.t. \quad & \eta_1 + \eta_2 = 1, \\ & (9a) - (9d), (10) \end{aligned} \tag{13}$$

where the coefficients η_1 and η_2 balance the trade-offs between the two subproblems $\mathcal{A}$ and $\mathcal{B}$. The constraints $(9a) - (9d)$ ensure the feasibility and adherence to predefined criteria, such as resource limits and binary decision variables. Note that the quadratic terms $\alpha_n^{i,l} q_n^{i,m}$ in the objective (11), the problem is a binary integer linearly constrained quadratic programming (BILCQP) problem. Meanwhile, the terms $\alpha_n^{i,l} \zeta^l$ in the objective (8) is a mixed binary integer linearly constrained programming (MBILP). It has been proved as a *NP-hard* [40] problem, thereby it is not feasible to solve the problem by heuristic algorithm or dynamic programming because of its high computational and spatial complexity and large scale. Thus, a deep reinforcement learning (DRL) method is derived. Before introducing the proposed DRL framework, it is necessary to embed the representation of nodes and microservices in the network to cope with the excessive state and action space.

V. Attention-based Microservice Representation

This section first introduces the attention mechanism-based microservice representation (AMR) layer to extract the system features using a multi-head attention mechanism, shown in Fig.4. First, the features of microservice and infrastructure are extracted into the structure attention space. Then, the service-chain attention space is derived by feeding with the obtained weighted representation. The final infrastructure and service embedding are concluded.

A. System Infrastructure Embedding

The infrastructure features f_x, $x \in \mathcal{I}$ is based on the calculation of service and energy overhead, related to the features of user devices, edge servers, and the cloud server.

- For *UE Layer*, we abstract the features f_m, $m \in M$ from the following information: microservice makespan f_m^{MM}, CPU computation ability f_m^{CPU}, microservice expectation finish time f_m^{EFT}, distance from the local edge server f_m^D;
- For *Edge Layer*, we have the features f_k, $k \in K$ by considering the information of CPU computation ability

f_k^{CPU}, channel state information f_k^{CSI}, microservice waiting queue f_k^{MWQ};
- For *Cloud Layer*, we extract the features f_c, according to the information about computation ability on each virtual machine f_c^{VM}, microservice waiting for queue f_c^{MWQ}.

Aiming to learn the embedding of the infrastructure status, the features extracted from the infrastructures are fed into the MLP (multilayer perceptron) with 2 hidden layers and 1 output layer, containing 512 neurons on each layer. Furthermore, we use the transformation matrix $\mathbf{H}$ to map the features of the infrastructure status f_x, the process of obtaining the embedding $\mathbf{w}_x$ can be expressed as $\mathbf{w}_x = \mathbf{H} \cdot f_x$.

B. Microservice Embedding

In order to capture the overall structure of the deployment of service and make the optimal re-deployment policy for each microservice $ms_n^i \in MS_n$. Considering the dependences of microservices, we employ the same transformation matrix $\mathbf{H}$ to map the microservice's features $f_y, y \in ms_n^i$ (*e.g.,* the size/type of microservice). Thus, the corresponding embedding $\mathbf{w}_y$ is expressed as $\mathbf{w}_y = \mathbf{H} \cdot f_y$.

C. Structure Attention

We utilize the self-attention mechanism to learn the importance of the infrastructure-service (structure) neighbours. Given a meta-chain of the service DAG $\mathbf{O}$, the structure self-attention τ_{xy} indicates the importance of structure (service embedding) y to the structure (infrastructure embedding) x on the meta-chain. The expression is shown as follows:

$$\tau_{xy}^{\mathbf{O}} = \beta_{SS}([\mathbf{w}_x, \mathbf{w}_y], \mathbf{O}), \tag{14}$$

where $\beta_{SS}(\cdot)$ is the deep neural network-based self-attention. It is applied to learn the cross-dependencies, thereby identifying the importance of each service embedding in relation to each infrastructure embedding. Note that it can be shared by the infrastructure-service pairs when they are in the same meta-chain, since the mapping patterns are similar to each other under a certain meta-chain.

This is achieved by computing the attention scores between each pair of infrastructure and service embedding using the masked attention mechanism deployed to inject the graph structure information. We have the normalized structure attention score expressed by the softmax function as follows:

$$\mu_{xy}^{\mathbf{O}} = softmax(\tau_{xy}^{\mathbf{O}}) = \frac{exp(\psi(\mathbf{w}^{\mathbf{O}} \cdot [\mathbf{w}_x || \mathbf{w}_y]))}{\sum_{h \in N_h^{\mathbf{O}}} exp(\psi(\mathbf{w}^{\mathbf{O}} \cdot [\mathbf{w}_x || \mathbf{w}_h]))}, \tag{15}$$

where $\mathbf{w}^{\mathbf{O}}$ indicates the chain-attention vector for the service meta-chain $\mathbf{O}$, $N_h^{\mathbf{O}}$ is the set that the neighbors h of microservices deployed in infrastructure x on the service meta-chain $\mathbf{O}$[1], and $\psi(\cdot)$ denotes the LeakyReLU (with negative input slop $\alpha = 0.2$) function with the concatenation operation $[\cdot||\cdot]$.

[1]Say if ms_1^1 in Fig. 3 is allocated on infrastructure x, the neighbors ms_1^3 and ms_1^2 are in the set $N_h^{\mathbf{O}}$.

Thus, we can obtain the representation of the infrastructure x by integrating the learnable weighted sum of the neighbouring service y as:

$$\mathbf{w}_x^{\mathbf{O}} = \psi(\sum_{y \in N_y^{\mathbf{O}}} \mu_{xy}^{\mathbf{O}} \cdot \mathbf{w}_y), \tag{16}$$

Furthermore, the multi-head attention mechanism is utilized to stabilize the learning process [41] by applying U independent heads to compute the hidden states. Specifically, we calculate the sum of the Eq. (16) for U times and obtain the average, resulting in the following output representation:

$$\mathbf{w}_x^{\mathbf{O}} = \psi(\frac{1}{U} \sum_{u=1}^{U} \sum_{y \in N_y^{\mathbf{O}}} \mu_{xy}^{\mathbf{O}} \cdot \mathbf{w}_y). \tag{17}$$

Finally, we respectively have U groups of chain-specific representations of the infrastructure embedding space $\mathbf{W}_x^{\mathbf{O}_u}$, and the service embedding space $\mathbf{W}_y^{\mathbf{O}_u}$, where $u = \{1, 2, \ldots, U\}$. These attention scores quantify how much each service embedding should influence each infrastructure embedding, thereby identifying the importance of each microservice embedding concerning each infrastructure embedding.

D. Service-chain Attention

Generally, the infrastructure can be associated with microservice through multiple meta-chains with different semantic information in a DAG. Therefore, from the infrastructure perspective, one infrastructure (*e.g.,* edge server) could obtain multiple semantic-specific service embeddings generated based on different meta-chains. To obtain a comprehensive representation of the infrastructure, a novel service-chain attention mechanism is proposed to learn the importance of different meta-chains for semantics selection.

For each meta-chain, semantic-specific service embeddings are generated by considering the unique semantic information conveyed by the meta-chain. The attention mechanism then computes attention scores for each meta-chain to determine its relevance. The service-chain attention takes U chains of infrastructure embeddings $\{\mathbf{W}_x^{\mathbf{O}_1}, \mathbf{W}_x^{\mathbf{O}_2}, \mathbf{W}_x^{\mathbf{O}_3}, \ldots, \mathbf{W}_x^{\mathbf{O}_U}\}$ learned from structure attention as input, and learn the attention values of service-chain as follows:

$$(\omega^{\mathbf{O}_1}, \omega^{\mathbf{O}_2}, \ldots, \omega^{\mathbf{O}_U}) = \Omega^{\mathbf{O}}(\mathbf{W}_x^{\mathbf{O}_1}, \mathbf{W}_x^{\mathbf{O}_2}, \ldots, \mathbf{W}_x^{\mathbf{O}_U}), \tag{18}$$

where $\Omega^{\mathbf{O}}$ indicates the deep neural network promised service-chain attention. Thus, the chain attention can capture multi-semantic information revealed by meta-chains in the system.

We can obtain the score function $\Lambda^{\mathbf{O}_u}$ for the service-chain $\mathbf{O}_u$ as follows, indicating the importance of different meta-chains: First, the learned service-chain embedding is derived from the structure attention $\mathbf{W}_x^{\mathbf{O}_u}$ by applying a nonlinear transformation function, which captures the complex dependencies and semantic information within the service-chain. Next, the score function of the service-chain is calculated by measuring the similarity between this transformed embedding and a predefined service-chain attention vector $\mathbf{g}$, using methods such as dot product or cosine similarity to quantify their relevance. Finally, to obtain the overall score function $\Lambda^{\mathbf{O}_u}$ in

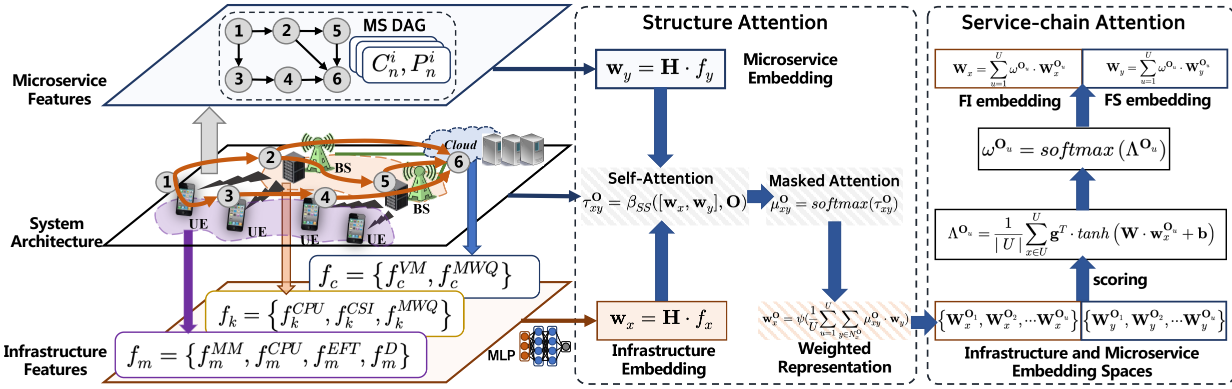


Fig. 4. Implementation of AMR scheme.

the following, we average the importance scores of all service-chain embeddings, providing a comprehensive assessment of the significance of each meta-chain and enabling effective prioritization and optimization in the deployment of services on edge computing resources.

$$\Lambda^{\mathbf{O}_u} = \frac{1}{\mid U \mid} \sum_{x \in U}^{U} \mathbf{g}^T \cdot tanh\left(\mathbf{W} \cdot \mathbf{w}_x^{\mathbf{O}_u} + \mathbf{b}\right), \tag{19}$$

where $\mathbf{W}$ is the weight matrix and $\mathbf{b}$ denotes the bias vector.

Similarly, the final service-chain attention can be calculated by the normalized softmax function of score function $\Lambda^{\mathbf{O}_u}$:

$$\omega^{\mathbf{O}_u} = softmax\left(\Lambda^{\mathbf{O}_u}\right) = \frac{exp\left(\Lambda^{\mathbf{O}_u}\right)}{\sum_{x=1}^{U} exp\left(\Lambda^{\mathbf{O}_x}\right)}. \tag{20}$$

In this way, the weight $\omega^{\mathbf{O}_u}$ can be considered as the importance of the service-chain $\mathbf{O}_u$ for the microservice deployment rating prediction, wherein the higher $\mathbf{O}_u$ scored, the more important the service-chain is.

Taking the learned meta-chain attention as coefficients in the system, the final infrastructure (FI) embedding $\mathbf{W}_x$ and final service (FS) embedding $\mathbf{W}_y$ can be derived by the aggregation of the service-chain embeddings as:

$$\begin{cases} \mathbf{W}_x = \sum_{u=1}^{U} \omega^{\mathbf{O}_u} \cdot \mathbf{W}_x^{\mathbf{O}_u}, & x \in \mathcal{I} \\ \mathbf{W}_y = \sum_{u=1}^{U} \omega^{\mathbf{O}_u} \cdot \mathbf{W}_y^{\mathbf{O}_u}, & y \in ms_n^i \end{cases}. \tag{21}$$

This service-chain attention mechanism allows the model to dynamically select and emphasize the most relevant semantic information from multiple meta-chains, thus providing a richer and more accurate representation of the infrastructure. By capturing the importance of different meta-chains, the model can effectively integrate diverse semantic insights.

The overall process of the AMR algorithm is shown in Algorithm 1. In the initialization phase, we obtain the microservice DAG s_n^D and input the infrastructure f_x and microservice features f_y as well as the service-chain set (Line 1). For the infrastructure and microservice on each service-chain $\mathbf{O}_u$, we first select the neighbours of microservices which are deployed in infrastructure from the set $N_h^{\mathbf{O}}$, then calculate the structure attention weight $\mu_{xy}^{\mathbf{O}}$ (Line 4-11). After getting the learnable infrastructure representation $\mathbf{w}_x^{\mathbf{O}}$ by (17), we have the infrastructure and microservice embedding, then the final service-chain attention $\omega^{\mathbf{O}_u}$ is obtained by normalizing the score function $\Lambda^{\mathbf{O}_u}$, we get the final infrastructure and microservice embeddings lastly (Line 12-17). The iterative processes of the AMR algorithm, as well as the calculations involved in attention mechanisms and embedding updates, are what primarily determines its computational complexity. It iterates over U service chains, T time steps, and I infrastructure features, resulting in an external complexity of $\mathcal{O}(U \cdot T \cdot I)$.

Algorithm 1 AMR Algorithm

1: **Initialize:** Microservice DAG $S_n^D = \{MS_n, \mathcal{E}_n\}$;
Infrastructure features $f_x, x \in \mathcal{I}$;
Microservice features $f_y, y \in ms_n^i$;
The service-chain set $\mathbf{O}_1, \mathbf{O}_2, \ldots \mathbf{O}_U$.
2: **Output:** The learnable structure attention weight $\mu_{xy}^{\mathbf{O}}$;
The service-chain attention weight $\omega^{\mathbf{O}_u}$;
The final infrastructure and microservice representation $\mathbf{W}_x$ and $\mathbf{W}_y$;
3: **for** $\mathbf{O}_u \in \{\mathbf{O}_1, \mathbf{O}_2, \ldots, \mathbf{O}_U\}$ **do**
4: **for** $t = 0, 1, 2, ..., T$ **do**
5: Compute the infrastructure and microservice embeddings $\mathbf{w}_x \leftarrow \mathbf{H} \cdot f_x$, $\mathbf{w}_y \leftarrow \mathbf{H} \cdot f_y$.
6: **for** $x \in \mathcal{I}$ **do**
7: Select the neighbourhood microservices deployed in infrastructure from the set $N_h^{\mathbf{O}}$;
8: **for** $y \in ms_n^i$ **do**
9: Calculate the structure attention weight $\mu_{xy}^{\mathbf{O}}$;
10: **end for**
11: **end for**
12: Calculate the learnable representation of the infrastructure $\mathbf{w}_x^{\mathbf{O}}$ by (17);
13: Obtain the infrastructure and microservice embeddings $\mathbf{W}_x^{\mathbf{O}_u}$ and $\mathbf{W}_y^{\mathbf{O}_u}$.
14: Compute score function $\Lambda^{\mathbf{O}_u}$ by (19).
15: Calculate the weight of final service-chain attention $\omega^{\mathbf{O}_u}$ by (20).
16: Obtain the final infrastructure and microservice embeddings $\mathbf{W}_x$ and $\mathbf{W}_y$ by (21), respectively.
17: **end for**
18: **Return** $\mu_{xy}^{\mathbf{O}}$, $\omega^{\mathbf{O}_u}$, $\mathbf{W}_x$ and $\mathbf{W}_y$.
19: **end for**

For each infrastructure feature, the algorithm processes N_h neighbouring microservices, contributing an additional factor of $\mathcal{O}(N_h)$. Matrix multiplications and non-linear transformations drive the core operations, which involve computing embedding and attention weights. Specifically, embedding computations and attention weight calculations (Eq. (14)-(19)) have a complexity of $\mathcal{O}(d^2)$ due to matrix operations, while normalized softmax operations (Eq. (20)) and final embedding aggregations (Eq. (21)) contribute $\mathcal{O}(U \cdot d)$ for each feature and microservice. In this way, we have the overall computational complexity of the AMR algorithm as $\mathcal{O}(U \cdot T \cdot I \cdot N_h \cdot d^2)$.

VI. Attention-based Deep Reinforcement Learning

The system learns the deployment strategy by considering the obtained representation context of FI and FS. In this paper, we adopt the attention-aided soft actor-critic (ASAC) method to solve the aforementioned joint optimization problem. The implementation of the proposed ASAC algorithm is shown in Fig. 5.

A. DRL-based Microservice Deployment Strategy Design

We consider the deployment policy learning to be in the continuous action spaces, and a finite-horizon Markov Decision Process (MDP) is deployed as follows:

1) System State: Recall that we obtain the FI embedding $\mathbf{W}_x$ and FS embedding $\mathbf{W}_y$ from the AMR algorithm. Here we define the system state vector space in a manner of two-tuple $\mathbf{S}^t = \{\mathbf{s}^t\}_{\times(x,y)} = \{\mathbf{s}_x^t, \mathbf{s}_y^t\}$, where $\mathbf{s}_x^t = (\mathbf{W}_x^t)_{x \in \mathcal{I}}$ and $\mathbf{s}_y^t = (\mathbf{W}_y^t)_{y \in ms_n^i}$ indicate the system infrastructure state and microservice placement state, respectively.

2) System Action: For the service deployment, the system decides which microservice is executed in which infrastructure. Formally, we define the system action space as $\mathbf{A}^t = \{\mathbf{a}_n^t\}$, where $\mathbf{a}_n^t = \{\alpha_n^{i,l}\}$. To this end, the system action can be expressed as follows:

$$\mathbf{A}^t = \{\mathbf{a}_n^t\}_{\times \mathbb{S}_n} = \{\alpha_n^{i,l}\}, \forall n, i \in ms_n^i, \forall l \in \mathcal{I}. \tag{22}$$

3) System Reward: Considering system reward is usually calculated as the weighted sum of current reward r, in this manner, we formulate the system reward according to the joint optimization objective, expressed as:

$$\mathbf{R}^t(\mathbf{S}^t, \mathbf{A}^t) = \eta_1 \mathcal{A}^t + \eta_2 \mathcal{B}^t, \tag{23}$$

where $\mathcal{A}^t = (\alpha_n^{i,l}\zeta^l + \frac{1}{X}\sum_{x=1}^{X} \xi^x)^t$ denotes the overall system cost and $\mathcal{B}^t = (\alpha_n^{i,l} q_n^{i,x})^t$ is the QoS, respectively[2]. The coefficients hold $\eta_1 + \eta_2 = 1$, and $\mathcal{A}$, $\mathcal{B}$ are both non-negative, the system reward meets $\mathbf{R}^t > 0$.

In this paper, we focus on minimizing the system cost while improving the QoS, the deployment policy $\pi(\mathbf{a}^t|\mathbf{s}^t) : \mathbf{S}^t \to \mathbf{A}^t$ designed for mapping from the system state under the dynamic environment to the action. Let $r(\mathbf{a}^t, \mathbf{s}^t) \sim \mathbf{R}^t(\mathbf{S}^t, \mathbf{A}^t)$ denote the current reward at t-th transition. By utilizing the SAC mechanism, we consider the stochastic policy by augmenting the cumulative system reward with the expected entropy of the policy over $\rho_\pi(\mathbf{s}^t)$ in the finite-horizon scenario. Our objective is to find the optimal policy π^*, shown as follows:

$$\pi^\star = arg\ \underset{\pi}{max} \sum_{t=0}^{T} \mathbb{E}_{(\mathbf{s}^t, \mathbf{a}^t) \sim \rho_\pi} \left[r\left(\mathbf{s}^t, \mathbf{a}^t\right) + \phi \mathcal{H}\left(\pi\left(\cdot \mid \mathbf{s}^t\right)\right) \right], \tag{24}$$

where $\phi > 0$ is the trade-off coefficient indicating the relative importance of the entropy term against reward, so as to control the stochastic of the optimal policy. $\mathcal{H}\left(\pi\left(\cdot \mid \mathbf{s}^t\right)\right) = \mathbb{E}_{\mathbf{a}^t \sim \rho_\pi}\left[-log_\pi\left(\mathbf{a}^t \mid \mathbf{s}^t\right)\right]$ is the policy entropy which measures the uncertainly of the random variable.

B. Soft Policy Model

In the maximum entropy paradigm, policy evaluation and policy improvement alternate in order to learn the optimal maximum entropy policies. For a deterministic policy, we can obtain the soft Q-value starting from any function $Q : \mathbf{S} \times \mathbf{A} \to \mathcal{R}$ iteratively with a modified Bellman backup operator Γ^π given by:

$$\Gamma^\pi Q\left(\mathbf{s}^t, \mathbf{a}^t\right) \triangleq r\left(\mathbf{s}^t, \mathbf{a}^t\right) + \gamma \mathbb{E}_{\mathbf{s}^{t+1} \sim p}\left[V\left(\mathbf{s}^{t+1}\right)\right], \tag{25}$$

where $V\left(\mathbf{s}^t\right) = \mathbb{E}_{\mathbf{a}^t \sim \pi}\left[Q\left(\mathbf{s}^t, \mathbf{a}^t\right) - log\ \pi\left(\mathbf{a}^t | \mathbf{s}^t\right)\right]$ is the soft state-value function.

Accordingly, we have the entropy-augmented soft returns as follows:

$$r_{soft}(\mathbf{s}^t, \mathbf{a}^t) \triangleq r\left(\mathbf{s}^t, \mathbf{a}^t\right) + \gamma \mathbb{E}_{\mathbf{s}^{t+1} \sim \rho_\pi}\left[\theta \mathcal{H}\left(\pi\left(\cdot \mid \mathbf{s}^{t+1}\right)\right)\right], \tag{26}$$

denotes that the accumulated returns under the system state $\mathbf{s}^t$ obtained by the current policy π.

To this end, we can calculate it for policy π by repeatedly computing the modified operator Γ^π according to the following *lemma*.

lemma 1: Given the entropy-augmented return, $r_{soft}(\mathbf{s}^t, \mathbf{a}^t)$, the soft Q-value function $Q_\varrho(\mathbf{s}^t, \mathbf{a}^t)$ converges to the optimal soft Q-value function $Q_\varrho^*(\mathbf{s}^t, \mathbf{a}^t)$ under the policy π_κ in the finite action space $\mathcal{A}$.

Proof: To show the convergence of the soft Q-value function, we start by considering the Bellman equation for the entropy-augmented return:

$$r_{\text{soft}}(\mathbf{s}^t, \mathbf{a}^t) = r(\mathbf{s}^t, \mathbf{a}^t) + \gamma \mathbb{E}_{\mathbf{s}^{t+1} \sim \rho}(\mathbf{s}^{t+1} | \mathbf{s}^t, \mathbf{a}^t)\left[V_\varpi(\mathbf{s}^{t+1})\right], \tag{27}$$

where

$$V_\varpi(\mathbf{s}^{t+1}) = \mathbb{E}_{\mathbf{s}^{t+1} \sim \pi_\kappa}(\cdot | \mathbf{s}^{t+1})\left[Q_\varrho^{t+1} - \theta log\ \pi_\kappa\left(\mathbf{a}^{t+1} | \mathbf{s}^{t+1}\right)\right]. \tag{28}$$

The soft Q-value update is given by:

$$Q_\varrho(\mathbf{s}^t, \mathbf{a}^t) = r_{soft}^t + \gamma \mathbb{E}_{(\mathbf{s}^{t+1}, \mathbf{a}^{t+1}) \sim \rho_\pi}[V_\varpi(\mathbf{s}^{t+1}, \mathbf{a}^{t+1})]. \tag{29}$$

Substituting $V_\varpi(\mathbf{s}^{t+1})$ into the soft Q-value update equation, we have:

$$\begin{aligned} Q_\varrho\left(\mathbf{s}^t, \mathbf{a}^t\right) = & r^t + \gamma \mathbb{E}_{\mathbf{s}^{t+1} \sim \rho_\pi}\left(\mathbf{s}^{t+1} | \mathbf{s}^t, \mathbf{a}^t\right)\left[\mathbb{E}_{\mathbf{a}^{t+1} \sim \pi_\kappa}\left(\cdot | \mathbf{s}^{t+1}\right)\right. \\ & \left.\left[Q_\varrho\left(\mathbf{s}^{t+1}, \mathbf{a}^{t+1}\right) - \theta log \pi_\kappa\left(\mathbf{a}^{t+1} | \mathbf{s}^{t+1}\right)\right]\right]. \end{aligned} \tag{30}$$

[2]Note that in order to calculate the long term reward of the learning process, here we take the immediate system cost and QoS as the reward input at each round t, rather than the long term system objective defined as (13).

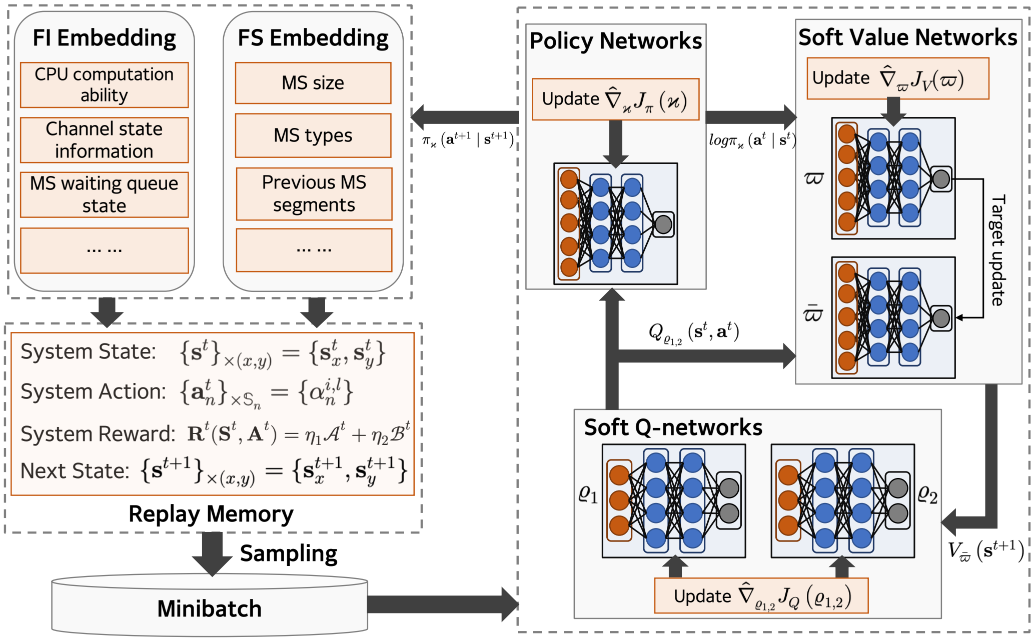


Fig. 5. Implementation of ASAC scheme.

Rewriting the update as a recursive relation, we get:

$$Q_\varrho\left(\mathbf{s}^t, \mathbf{a}^t\right) = r^t + \gamma \mathbb{E}_{\mathbf{s}^{t+1} \sim \rho_\pi, \mathbf{a}^{t+1} \sim \pi_\kappa}[Q_\varrho\left(\mathbf{s}^{t+1}, \mathbf{a}^{t+1}\right) - \theta log\pi_\kappa\left(\mathbf{a}^{t+1}|\mathbf{s}^{t+1}\right). \tag{31}$$

To establish convergence, we analyze the fixed point of this recursive update. Let $Q_\varrho^{fixed}(\mathbf{s}^t, \mathbf{a}^t)$ be the fixed point of the above equation, satisfying:

$$Q_\varrho^{fixed}\left(\mathbf{s}^t, \mathbf{a}^t\right) = r^t + \gamma \mathbb{E}_{\mathbf{s}^{t+1} \sim \rho_\pi, \mathbf{a}^{t+1} \sim \pi_\kappa}[Q_\varrho^*\left(\mathbf{s}^{t+1}, \mathbf{a}^{t+1}\right) - \theta log\pi_\kappa\left(\mathbf{a}^{t+1}|\mathbf{s}^{t+1}\right). \tag{32}$$

Recall the modified Bellman backup operator Γ in Eq. (25), we seek to show that Γ is a contraction mapping, which ensures the convergence to a unique fixed point. Thus, for any two Q-value functions Q_1 and Q_2, we have:

$$\Gamma \parallel Q_1 - Q_2 \parallel \leq \gamma \parallel Q_1 - Q_2 \parallel. \tag{33}$$

This follows from the properties of expectation and the boundedness of the entropy term.

Since $\gamma < 1$ and Γ is a contraction mapping. By the Banach fixed-point theorem [42], the sequence of Q-value functions generated by the Bellman update converges to a unique fixed point Q_ϱ^{fixed}. Thus, the soft Q-value function $Q_\varrho(\mathbf{s}^t, \mathbf{a}^t)$ converges to the optimal soft Q-value function $Q_\varrho^*(\mathbf{s}^t, \mathbf{a}^t)$ under the policy π_κ.

Proof completes. ∎

The agent can concurrently learn the entropy-augmented policy evaluation and soft policy improvement by alternately updating the policies towards the exponential of a new Q-function. The particular choice can be guaranteed to result in an improved policy in terms of its soft value. The main purpose is to find the new deployment policy π_{new} which is better than the current policy π_{old}. Denote the Π as the set of policies, and we have the constraint $\pi \in \Pi$. To obtain the guaranteed deployment policy improvement, we update the policy by using Kullback-Leibler (KL) divergence, and output the Gaussian distribution as follows:

$$\pi_{new}(\cdot|\mathbf{s}^t) = arg \min_{\pi' \in \Pi} D_{KL}\left(\pi'\left(\cdot|\mathbf{s}^t\right) \parallel \frac{exp\left(Q^{\pi_{old}}\left(\mathbf{s}^t \cdot\right)\right)}{Z^{\pi_{old}}\left(\mathbf{s}^t\right)}\right), \tag{34}$$

where $D_{KL}(\cdot)$ is the KL divergence operation, and the partition function $Z^{\pi_{old}}\left(\mathbf{s}^t\right)$ is used to normalize the distribution. Note that $Z^{\pi_{old}}\left(\mathbf{s}^t\right)$ is always intractable and usually can be ignored due to it contributes nothing to the gradient concerning the new policy. In this way, it is noticed that the new policy π_{new} can achieve a higher value than the old one in terms of the objective of Eq.(24). To this end, we have the following lemma to formalize the improvement process.

lemma 2: Let $\pi_{old} \in \Pi$ and let π_{new} be the optimizer of the minimization problem defined in Eq.(34). Then we have $Q_{\pi_{new}}(\mathbf{s}^t, \mathbf{a}^t) \geq Q_{\pi_{old}}(\mathbf{s}^t, \mathbf{a}^t)$, $\forall(\mathbf{s}^t, \mathbf{a}^t) \in \mathbf{S} \times \mathbf{A}$, where $| A |< \infty$.

Proof: First, let $J^{\pi_{old}}(\pi'(\cdot|\mathbf{s}^t)) = D_{KL}(\pi'(\cdot|\mathbf{s}^t)||\cdot)$. Recall that expression of soft value function $V(\mathbf{s}^t)$, we can expand the KL-divergence and obtain the expectation as

$$\begin{aligned}
& J_{\pi_{old}}\left(\pi'\left(\cdot \mid \mathbf{s}^t\right)\right) \\
& \triangleq D_{KL}\left(\pi'\left(\cdot \mid \mathbf{s}^t\right) \parallel exp\left(Q_{\pi_{old}}\left(\mathbf{s}^t, \cdot\right) - logZ_{\pi_{old}}\left(\mathbf{s}^t\right)\right)\right) \\
& = \int \pi'\left(\mathbf{a}^t \mid \mathbf{s}^t\right)\left(log\pi'\left(\mathbf{a}^t \mid \mathbf{s}^t\right)\right. \\
& \left. + logZ_{\pi_{old}}\left(\mathbf{s}^t\right) - Q_{\pi_{old}}\left(\mathbf{s}^t, \mathbf{a}^t\right)\right) d\mathbf{a}^t \\
& = \mathbb{E}_{\mathbf{a}^t \sim \pi'}\left[log\pi'\left(\mathbf{a}^t \mid \mathbf{s}^t\right) + logZ_{\pi_{old}}\left(\mathbf{s}^t\right) - Q_{\pi_{old}}\left(\mathbf{s}^t, \mathbf{a}^t\right)\right]
\end{aligned} \tag{35}$$

In this way, there always exists a policy $\pi_{new} = \pi_{old} \in \Pi$, holding that

$$J_{\pi_{old}}(\pi_{new}(\cdot \mid \mathbf{s}^t)) \leq J_{\pi_{old}}(\pi_{old}(\cdot \mid \mathbf{s}^t)) \tag{36}$$

then apply the standard convergence proof from literature [19], and lemma 1 leads to the convergence to $Q_{\pi_{new}}$.

Proof completes. ∎

C. ASAC Learning Process

In order to deal with the large continuous domains, the function approximators for both Q-function and policy are employed by alternating between optimizing both networks with stochastic gradient descent (SGD). We employ a parameterized soft Q-function $Q_\varrho(\mathbf{s},\mathbf{a})$, value function $V_\varpi(\mathbf{s}^t)$ and the deploy policy $\pi_\varkappa(\mathbf{a}\mid\mathbf{s})$, where ϱ and $\varkappa$ are the coefficients of networks, and the policy can be given as a Gaussian with mean and covariance by the neural networks.

The soft Q-function parameters can be trained by minimizing the soft Bellman residual, shown as follows:

$$J_Q(\varrho)=\mathbb{E}_{(\mathbf{s}^t,\mathbf{a}^t)\sim\mathcal{D}}\left[\frac{1}{2}\left(Q_\varrho\left(\mathbf{s}^t,\mathbf{a}^t\right)-\hat{Q}\left(\mathbf{s}^t,\mathbf{a}^t\right)\right)^2\right], \quad (37)$$

and

$$\begin{aligned}\hat{Q}\left(\mathbf{s}^t,\mathbf{a}^t\right)=&r\left(\mathbf{s}^t,\mathbf{a}^t\right)+\gamma\mathbb{E}_{(\mathbf{s}^{t+1},\mathbf{a}^{t+1})\sim\rho_\pi}[Q_{\bar{\varrho}}\left(\mathbf{s}^{t+1},\mathbf{a}^{t+1}\right)\\&-\theta log_{\pi_\varkappa}\left(\mathbf{a}^{t+1}\mid\mathbf{s}^{t+1}\right)].\end{aligned}$$

Then (37) can be optimized via computing its stochastic gradients as:

$$\begin{aligned}\hat{\nabla}_\varrho J_Q(\varrho)=&\nabla Q_\varrho\left(\mathbf{s}^t,\mathbf{a}^t\right)\left(Q_\varrho\left(\mathbf{s}^t,\mathbf{a}^t\right)-r^t\right.\\&-\gamma[Q_{\bar{\varrho}}\left(\mathbf{s}^{t+1},\mathbf{a}^{t+1}\right)-\theta log_{\pi_\varkappa}\left(\mathbf{a}^{t+1}\mid\mathbf{s}^{t+1}\right)]\\=&\nabla Q_\varrho\left(\mathbf{s}^t,\mathbf{a}^t\right)\left(Q_\varrho\left(\mathbf{s}^t,\mathbf{a}^t\right)-r^t-\gamma V_{\bar{\varpi}}\left(\mathbf{s}^{t+1}\right)\right)\end{aligned} \quad (38)$$

where the $\bar{\varpi}$, from a target value network $V_{\bar{\varpi}}$, is used to stabilize the training process by exponentially moving the average of the soft Q-function weights.

The soft value function $V_\varpi(\mathbf{s}^t)$ is to be trained by minimizing the mean squared error (MSE) as follows:

$$\begin{aligned}J_V(\varpi)=&\mathbb{E}_{\mathbf{s}^t\sim\mathcal{D}}[\frac{1}{2}((V_\varpi(\mathbf{s}^t))-\mathbb{E}_{\mathbf{a}^t\sim\pi_\varkappa}[Q_\varrho(\mathbf{s}^t,\mathbf{a}^t)\\&-log\pi_\varpi(\mathbf{a}^t\mid\mathbf{s}^t)])^2],\end{aligned} \quad (39)$$

where $\mathcal{D}$ is the replay buffer storing the historical experiences, then we can estimate the (39) by using the unbiased estimator:

$$\begin{aligned}\hat{\nabla}_\varpi J_V(\varpi)=&\nabla_\varpi V_\varpi(\mathbf{s}^t)(V_\varpi\left(\mathbf{s}^t\right)-Q_\varrho(\mathbf{s}^t,\mathbf{a}^t)\\&+log\pi_\varpi(\mathbf{a}^t\mid\mathbf{s}^t)])^2].\end{aligned} \quad (40)$$

In this way, it is observed that the actions are selected according to the current policy, rather than from the replay buffer $\mathcal{D}$.

Similarly, the policy parameter can be trained by directly minimizing the KL-divergence from (34), shown as follows:

$$\begin{aligned}J_\pi(\varkappa)=&\mathbb{E}_{\mathbf{s}^t\sim\mathcal{D},\mathring{a}^t\sim\mathring{A}}[log\pi_\varpi\left(f_\varpi\left(\mathring{a}^t;\mathbf{s}^t\right)\mid\mathbf{s}^t\right)\\&-Q_\varrho\left(\mathbf{s}^t,f_\varpi\left(\mathring{a}^t;\mathbf{s}^t\right)\right)],\end{aligned} \quad (41)$$

where $\mathring{a}$ is an input noise vector, sampled from a fixed spherical Gaussian distribution, holding $f_\varpi(\mathring{a}^t;\mathbf{s}^t)=\mathbf{a}^t$. In this way, we can approximate the gradient as follows:

$$\begin{aligned}\hat{\nabla}_\varkappa J_\pi(\varkappa)=&\nabla_\varkappa log\pi_\varkappa\left(\mathbf{a}^t\mid\mathbf{s}^t\right)+\nabla_\varkappa f_\varkappa\left(\mathring{a}^t;\mathbf{s}^t\right)\\&\left(\nabla_{\mathbf{a}^t}log\pi_\varkappa\left(\mathbf{a}^t\mid\mathbf{s}^t\right)-\nabla_{\mathbf{a}^t}Q\left(\mathbf{s}^t,\mathbf{a}^t\right)\right)\end{aligned} \quad (42)$$

The process of the proposed ASAC is shown in Algorithm 2. Input the system state $\{\mathbf{s}^t\}_{\times(x,y)}$ and action $\{\mathbf{a}^t\}_{\times\mathbb{S}_n}$ obtained from AMR layer (Line 3), compute the current reward and next system state at each episode (Lines 4-9), Then, the soft Q-function, value function, and soft policy parameters will be updated until convergence (Lines 10-14).

Algorithm 2 ASAC Algorithm

Initialize: Infrastructure features $f_x, x\in\mathcal{I}$;
Soft Q-function parameters ϱ_1, ϱ_2;
Value function parameters ϖ and target value network parameters $\bar{\varpi}$. Soft policy parameters $\varkappa$.
2: **for** $t=0,1,2,...,T$ **do**
  Obtain the system state $\mathbf{s}^t$ from *Algorithm.1*;
4: **for** each episode **do**
    Get the system action $\mathbf{a}^t$ according to the input system state $\mathbf{s}^t$ in actor-network, holding $\mathbf{a}^t\sim\pi_\varkappa(\mathbf{a}^t\mid\mathbf{s}^t)$.
6:    Obtain the current reward r^t and the next system state $\mathbf{s}^{t+1}$.
    Store the quadruplet into replay memory $\mathcal{D}\leftarrow\{(\mathbf{s}^t,\mathbf{a}^t,r^t,\mathbf{s}^{t+1})\}\bigcup\mathcal{D}$.
8:    Randomly sample a batch of $\mathcal{N}$ samples from $\mathcal{D}$.
  end for
10: **for** each gradient step in batch $\mathcal{N}$ **do**
    Update the soft Q-function parameters ϱ and $\bar{\varrho}$ according to (38).
12:    Update the soft value function parameters $\bar{\varpi}\leftarrow\tau\varpi+(1-\tau)\bar{\varpi}$ according to (40).
    Update the soft policy parameters $\varkappa$ according to (42).
14: **end for**
end for

The computational complexity of the proposed ASAC algorithm can be comprehensively analyzed by considering the key processes, *e.g.,* MDP process, gradient updates, and parameter optimisation. The algorithm begins with initializing infrastructure features and Q-function parameters, which has a complexity of $\mathcal{O}(1)$. It then proceeds with an outer loop that runs for T times. Within each time step, the proposed ASAC selects actions based on the policy network π_κ with a complexity of $\mathcal{O}(d)$ and computes the current reward and next state, both operations being $\mathcal{O}(1)$. Storing transitions in the replay memory and sampling batches involve $\mathcal{O}(1)$ and $\mathcal{O}(N)$ operations, respectively. Furthermore, the computation-intensive part of the proposed ASAC involves updating the soft Q-function parameters ϱ and $\bar{\varrho}$, value function ϖ, and policy parameters $\varkappa$ via gradient descent, each with a complexity of $\mathcal{O}(P)$, where P denotes the number of parameters in the neural networks. The soft Q-function is updated using the soft Bellman Equation (Eq. (30) and (29)), which includes expected value computations over future states, adding $\mathcal{O}(P)$ complexity for each update. Additionally, the policy update through KL-divergence minimization (Eq. (33)-(34)) and entropy-augmented returns further contribute to the overall complexity. In conclusion, we have the total computational complexity of the proposed ASAC as $\mathcal{O}(T\cdot E\cdot(N+G\cdot P))$, where E is the number of episodes, N is the batch size, G represents the number of gradient steps, and P accounts for the parameter updates in the networks.

VII. Experiment

A. Experimental Settings

In this section, we conduct the experimental simulations in the Python environment, the main parameter values of computation ability of UE layer c^m, edge layer c_m^k, and cloud layer c^C are uniformly set at 1.0–1.2 GHz, 2.4–2.5 GHz, and 3.0 GHz, respectively. The number of microservices ranges from 5 to 20, with a uniform size of [300 kb, 500 kb]. The number of UE and edge servers are set in the range of [5, 10] and $\{3, 5, 10\}$, respectively. The discount rate γ is 0.85, and the batch size is 128. We set the learning rate as $3e-4$, the system runs 1000 episodes, and all the experimental results are the average values for 20 runs. Other simulation parameters in this work are similar to [39] and we summarize some important parameter settings in Table. II in the following.

TABLE II
Main Simulation Parameters Settings

Parameter	Value
c^m	1.0-1.2 GHz
c_m^k	2.4-2.5 GHz
c^C	3.0 GHz
Microservice number	[5, 20]
Size of microservice	[300 kb, 500 kb]
UE number	[5, 10]
Edge server number	{3, 5, 10}
Discount rate γ	0.85
Batch size	128
Learning rate	$3e-4$
Coefficient of execution time ω_t	0.5
Coefficient energy consumption ω_e	0.5

The data is generated in the form of the service DAGs by a random graph generator [43], where the depth of a DAG is limited from 2 to 5, and the parameters of fat (affects the height and width of the DAG[3]), $desity$ (determines the number of edges between two levels of the DAG), and $regular$ (determines the uniformity of the number of tasks in each level) are set as [0.3, 0.5, 0.7], [0.6, 0.7, 0.8], and [0.5, 0.7, 0.9], respectively.

B. Performance Metrics and Baselines

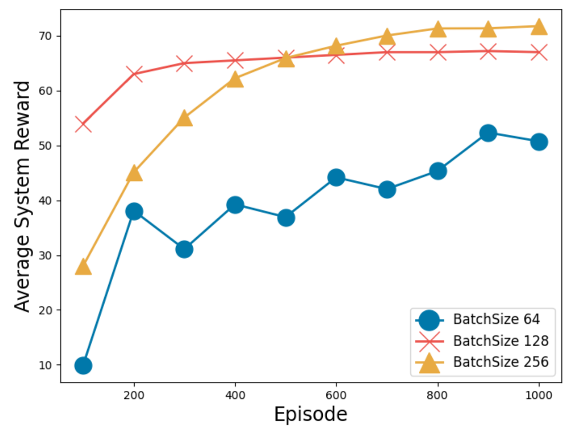
1) Performance Metrics: To evaluate the performance of the proposed algorithm, we first demonstrate the measurements of the proposed ASAC in terms of average system cost, QoS, and reward under different numbers of microservices (MS), user equipment (UE), and edge servers (ES).

Recall that the average system cost function is derived from (9) which considers the time-varying energy consumption and deployment fee. Moreover, we design the average system QoS based on (12) that outputs the performance of the proposed algorithm from the user part. Furthermore, the performance of average system reward is integrated with the aforementioned two metrics from the DRL model-design perspective. In this way, we carry out the system cost, QoS, and reward at each time slot, and then obtain the average outputs.

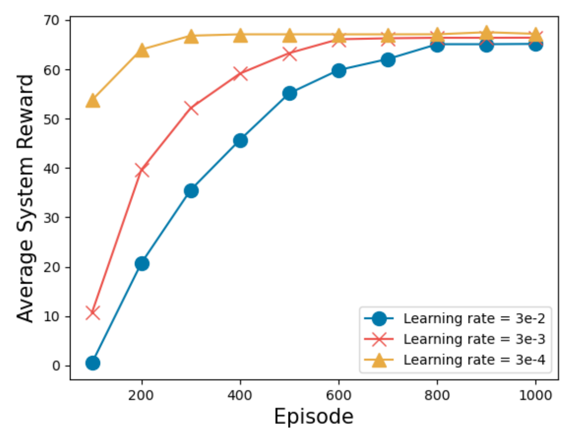
[3]The width in each level is defined by a uniform distribution with a mean equal to fat. The height is created until MSs are defined in the DAG. The width of the DAG is the maximum number of tasks that can be executed concurrently.

2) Baselines: The comparisons of the following three algorithms are derived:

- **Soft Actor-Critic (SAC) [19]:** The original SAC algorithm without considering the attention-aided mechanism.
- **Double Q-Learning (DDQN) [44]:** A classical deep reinforcement learning method is proposed by using experience replay to learn in small batches.
- **Random allocation:** The microservices are randomly allocated in the system.
- **All Local:** The microservices are only deployed and executed on local user equipment.
- **All Edge:** The microservices are only deployed and executed on the edge servers.
- **Cloud First:** The microservices tend to be executed on the cloud server.



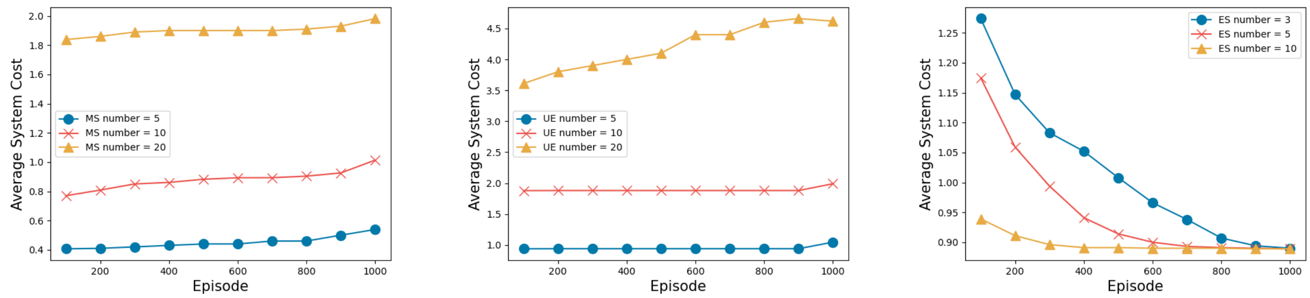
(a) System reward with different batch sizes.



(b) System reward with different learning rates.

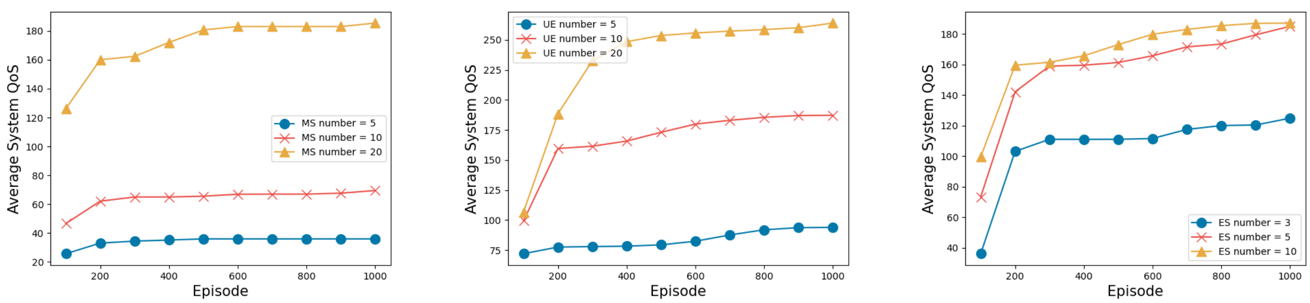
Fig. 6. Performance of average system reward with different batch sizes and learning rates.

To validate the effectiveness of the proposed ASAC algorithm, recall that the optimization objective (13) comprises two sub-problems: minimizing the system cost and maximizing the system QoS. Besides, the system reward function represents the long-term cumulative gains of these two sub-problems. Hence, we first compare the immediate reward performance by evaluating the average system cost and system QoS. This part illustrates the network performance comparison in terms



(a) Average system cost with different MS number. (b) Average system cost with different UE number. (c) Average system cost with different ES number.

Fig. 7. Performance of average system cost with different MS, UE, and ES numbers.



(a) Average system QoS with different MS number. (b) Average system QoS with different UE number. (c) Average system QoS with ES number.

Fig. 8. Performance of average system QoS with different MS, UE, and ES numbers.

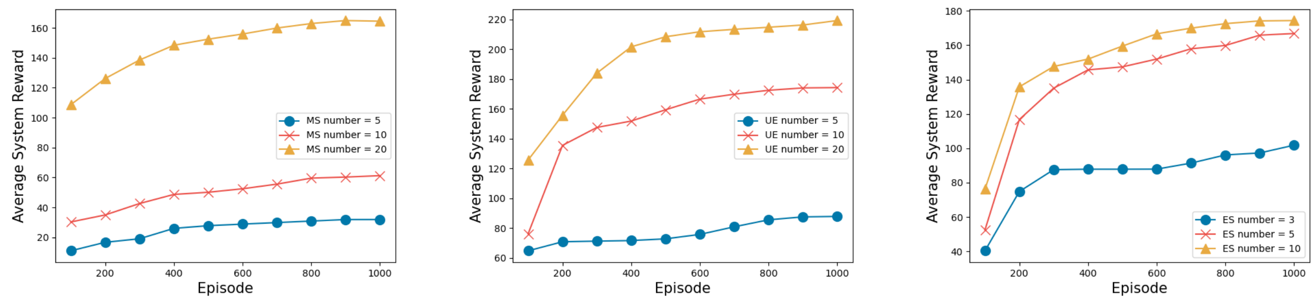
of the cost of CTE, *i.e.,* finish time (FT) jointly energy consumption, and user QoS, respectively. Furthermore, we compare the performance in terms of average system reward, which effectively demonstrates the advantages of the proposed learning algorithm architecture. Additionally, we also present performance under different batch sizes and learning rates.

C. Simulation Results

We first conduct the convergence performance of the average system reward with different learning rates and batch sizes, respectively. As shown in Fig. 6(a), we observe that varying the batch sizes has distinct effects on the reward curves. A batch size of 64 demonstrates fast initial convergence but exhibits significant fluctuations, indicating instability in training. When the batch size is 128, the curve shows smoother behaviour with moderate initial convergence and improved stability. Nevertheless, a batch size of 256 provides the most stable performance, though it has a slower initial convergence. These results suggest that larger batch sizes enhance training stability and final performance but at the expense of slower initial convergence, highlighting the need to balance convergence speed and stability when selecting batch sizes. Fig. 6(b) shows the performance of system reward under different learning rates. However, there is a significant difference in the convergence rate among the three rates. The performance reaches stability when $lr = 3e-4$ after the 300 episodes, while $lr = 3e-2$ and $lr = 3e-3$, the curves start to level off after 800 and 600 episodes, respectively. It shows that a higher learning rate gives a quicker convergence speed. Accordingly, the batch size and learning rate are respectively set as 128 and $3e-4$ for the hereafter simulations.

1) ***Scalability and Efficiency Performance:*** In considering the equal importance of both latency and energy consumption to user experience in equation (7), the weights ω_t and ω_e are set to 0.5. Additionally, the structures of MS are randomly generated with the DAG depth limited to 5. This section evaluates the performance of average system cost with varying numbers of MSs, UEs, and ESs. From Fig.7(a) and Fig.7(b), it is observed that the average system cost increases with the number of MSs and UEs. Conversely, Fig. 7(c) shows that the average system cost decreases as the number of ESs increases. The increase in system cost with more MSs and UEs can be attributed to the higher network resource occupancy, which results in increased costs. On the other hand, an increased number of ESs enhances network computation capabilities, thereby reducing the overall system cost.

From Fig.8, it is evident that the average system QoS improves with the increasing number of MSs, UEs, and ESs. The performance significantly increases in the scenario with 10 UEs and 5 ESs, achieving nearly an 8-fold improvement when the number of MSs is increased from 5 to 20, as depicted in Fig.8(a). Fig. 8(b) illustrates the case with varying numbers of UEs while maintaining 20 MSs and 10 ESs. It can be observed that with 20 UEs, the system QoS performance peaks after 400 episodes. Conversely, the system exhibits a faster convergence rate with 10 UEs. These results are somewhat counterintuitive, considering that the QoS function is designed



(a) Average system reward with different MS number.

(b) Average system reward with different UE number.

(c) Average system reward with different ES number.

Fig. 9. Performance of average system reward with different MS, UE, and ES numbers.

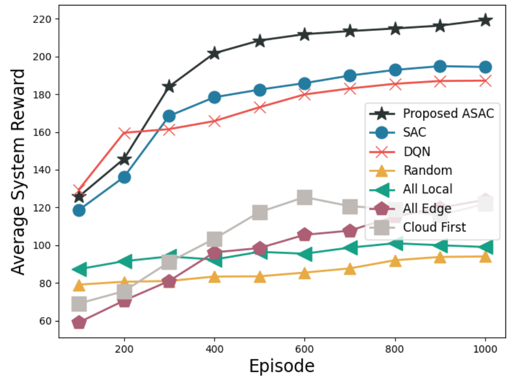
to reflect the accumulated satisfaction of UEs. Accordingly, the average system QoS performance enhances as the number of UEs increases over time. In contrast to the previous two cases, Fig. 8(c) shows that the system achieves almost identical performance with 5 ESs compared to 10 ESs. This indicates that the performance benefits saturate at a certain number of ESs, highlighting the diminishing returns on system QoS with further increases in ESs beyond a certain threshold.

We demonstrate the performance of the average system reward in Fig. 9, which exhibits trends similar to the previously discussed evaluation cases. In Fig. 9(a), the average system reward significantly improves as the number of MSs increases. Specifically, the reward is approximately 7 times higher when the number of MSs is increased from 5 to 20, and 1.67 times higher when the number of MSs is increased from 10 to 20. This substantial increase is due to the efficient utilization of network resources, as more MSs can handle a greater volume of tasks, leading to higher cumulative rewards. In Fig. 9(b), the system reward improves by 1.75 times and 37.5% when the number of UEs is increased to 20 compared to 10 and 5, respectively. This enhancement is attributed to the increased number of UEs, which bring more network resources into the system. With more UEs, there is a higher demand and utilization of resources, resulting in better overall system performance and reward, even though the number of MSs remains constant. Fig. 9(c) displays the average system reward with varying numbers of ESs. The performance trend here is similar to that observed in Fig. 8(c). The system reward is relatively stable when the number of ESs increases from 5 to 10. This stability indicates that the system reaches a point of diminishing returns with additional ESs. Beyond a certain number of ESs, the improvement in system performance and reward is marginal. This behaviour suggests that there is an optimal number of ESs for the given system configuration, beyond which additional ESs do not significantly enhance performance. The figures highlight that the average system reward increases significantly with the number of MSs and UEs due to better resource utilization and higher task-handling capacity.

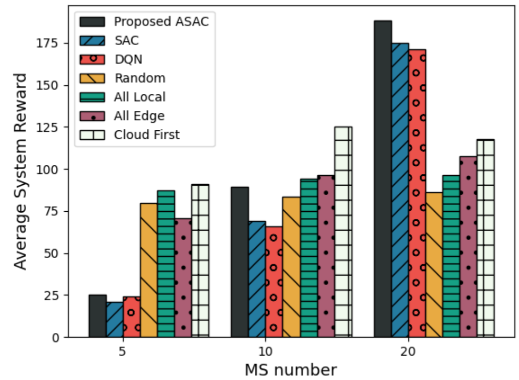
*2) **Convergence Performance**:* Fig. 10(a) demonstrates that the proposed ASAC algorithm significantly outperforms the SAC, DQN, and Random algorithms, achieving improvements of up to 15.8%, 22.2%, and 120%, respectively, in an evaluation setup with 20 MSs, 20 UEs, and 10 ESs. This superior performance of ASAC is attributed to several key factors. First, ASAC's advanced feature extraction capabilities enable it to effectively capture the system's state, leading to more informed decision-making. Second, it optimizes resource allocation and utilization, minimizing latency and energy consumption. Third, the decision-making process and reinforcement learning techniques allow it to learn optimal policies for task offloading and resource management, resulting in faster convergence, especially noticeable after 300 episodes. Additionally, the ability to handle dynamic environments and scalability ensures high performance even as system configurations change. Furthermore, its balanced consideration of latency and energy consumption optimizes both aspects simultaneously, ensuring system efficiency and sustainability. In contrast, while SAC and DQN perform better than the Random algorithm, they lack the advanced capabilities of ASAC, resulting in similar but lower performance levels. The Random algorithm's poor performance highlights the importance of intelligent algorithm design in optimizing system performance.

Moreover, from Fig. 10(b), it is observed that the Random scheme achieves the best performance when the number of MSs is 5, due to the sufficient availability of UEs and ESs allowing for efficient random allocation of a small number of MSs. However, as the number of MSs increases, the performance of all schemes improves significantly, except for the Random scheme. The proposed ASAC algorithm consistently outperforms the other schemes, reaching the highest performance levels due to its advanced adaptive learning and predictive capabilities, which allow it to efficiently allocate resources and manage tasks based on real-time feedback. The proposed ASAC provides the ability of feature extraction and targeted task prioritization to ensure optimal resource utilization, while its balanced optimization of latency, energy consumption, and computational efficiency further enhances performance. Additionally, ASAC's scalability ensures that it maintains high performance even as the system grows, making it an ideal choice for dynamic and large-scale network environments where efficient resource management is crucial for maximizing system rewards.

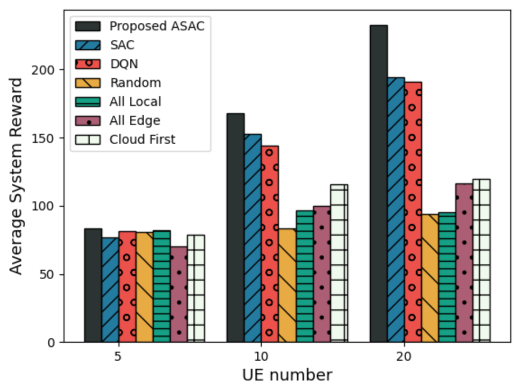
Fig. 10(c) illustrates the reward performance with varying



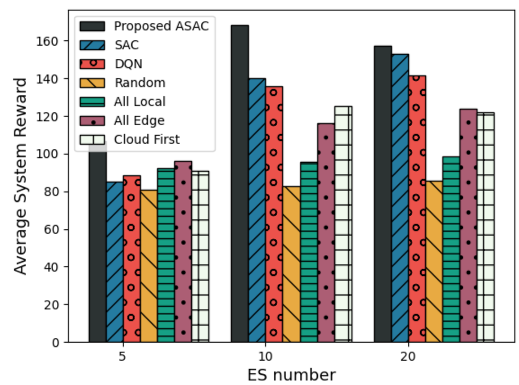
(a) Average system reward with different episodes.



(b) Average system reward with different MS number.



(c) Average system reward with different UE number.



(d) Average system reward with different ES number.

Fig. 10. Comparisons of average system reward under different episodes, MS, UE, and ES number.

numbers of UEs, with the numbers of MS and ES set to 20 and 10, respectively. The Random algorithm maintains a stable trend as the number of UEs increases since most of the MSs are stochastically allocated to UEs initially. However, the proposed ASAC algorithm significantly outperforms the other schemes, showing much better performance as the number of UEs increases. The performance of the proposed ASAC is attributed to its ability to fully extract critical features from the system and optimize deployment at a lower cost. This capability allows ASAC to efficiently manage resources and enhance system reward, demonstrating its effectiveness in handling increased UEs compared to SAC, DQN, and other baseline schemes.

From Fig. 10(d), the performance of the proposed ASAC algorithm shows an initial improvement followed by a decrease as the number of ESs increases, achieving the highest reward performance when the number of ESs is 10, and then declining at 20 ESs. In contrast, other schemes like SAC and DQN exhibit a stable, increasing trend. This behaviour is due to the ASAC algorithm's efficient utilization of system resources, which are fully occupied and optimized at 10 ESs. However, when the number of ESs increases to 20, the system becomes overloaded with higher computational demands for network representation, leading to a decrease in performance. The ASAC algorithm's ability to balance resource allocation and computational efficiency allows it to outperform other schemes under optimal conditions, highlighting its advantage in managing system resources effectively.

VIII. Conclusion

In this paper, we have investigated the microservice deployment (MSD) problem in hierarchical edge computing networks, to jointly minimize the overall system cost and maximize the quality of service (QoS) of users. To achieve this goal, the internal dependency of the microservices is revealed by employing the directed acyclic graph (DAG) to model the execution sequence of the microservice. We focus on the crucial information in the networks that impacts the decision-making of microservice deployment strategies and have proposed attention-based microservice representation (AMR) to extract the system context better. Specifically, we have first embedded the information of the network infrastructure (*e.g.,* the makespan, CPU computation capabilities, channel state) and microservice itself (*e.g.,* the input size/type) respectively. Then, the service-chain attention is constructed to learn the importance level of the different service DAGs by coupling the two information above embedding. The final infrastructure and microservice embedding will be utilized in the forthcoming strategy design to tackle the NP-hard MSD optimization problem, which is formally defined as a BILCQP problem. To effectively adapt to the continuous deployment problem, we modelled the MSD problem as an MDP and proposed the attention-modified soft actor-critic algorithm (ASAC) to solve it. The experimental simulation results have shown the superiority of the proposed algorithm.

REFERENCES

[1] C. Wang, B. Jia, H. Yu, and et al., "Deep reinforcement learning for dependency-aware microservice deployment in edge computing," in *GLOBECOM 2022-2022*. IEEE, 2022, pp. 5141–5146.

[2] Q. Meng, S. Zhao, Z. Huang, and F. Zhou, "Magface: A universal representation for face recognition and quality assessment," in *Proceedings of the CVPR*, 2021, pp. 14 225–14 234.

[3] E. Rosen, T. Groechel, M. E. Walker, C. T. Chang, and J. Z. Forde, "Virtual, augmented, and mixed reality for human-robot interaction (vamhri)," in *Companion of the 2021 ACM/IEEE International Conference on Human-Robot Interaction*, 2021, pp. 721–723.

[4] G. Gomes, C. A. Vidal, J. B. Cavalcante-Neto, and Y. L. Nogueira, "Two level control of non-player characters for navigation in 3d games scenes: A deep reinforcement learning approach," in *2021 20th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*. IEEE, 2021, pp. 182–190.

[5] H. D. Kabir, A. Khosravi, S. K. Mondal, M. Rahman, S. Nahavandi, and R. Buyya, "Uncertainty-aware decisions in cloud computing: Foundations and future directions," *ACM Computing Surveys (CSUR)*, vol. 54, no. 4, pp. 1–30, 2021.

[6] H. Sun, L. Wang, and M. Liu, "Dynamic deployment and scheduling strategy for dual-service pooling based hierarchical cloud service system in intelligent buildings," *IEEE Transactions on Cloud Computing*, 2021.

[7] N. Chen, T. Qiu, and et al., "An adaptive robustness evolution algorithm with self-competition and its 3d deployment for internet of things," *IEEE/ACM Transactions on Networking*, vol. 30, no. 1, pp. 368–381, 2021.

[8] X. Wang, C. Wang, X. Li, V. C. Leung, and T. Taleb, "Federated deep reinforcement learning for internet of things with decentralized cooperative edge caching," *IEEE Internet of Things Journal*, vol. 7, no. 10, pp. 9441–9455, 2020.

[9] J. Islam, T. Kumar, I. Kovacevic, and E. Harjula, "Resource-aware dynamic service deployment for local iot edge computing: Healthcare use case," *IEEE Access*, vol. 9, pp. 115 868–115 884, 2021.

[10] N. Chen, T. Qiu, X. Zhou, S. Zhang, W. Si, and D. O. Wu, "A distributed co-evolutionary optimization method with motif for large-scale iot robustness," *IEEE/ACM Transactions on Networking*, 2024.

[11] Q. Tang, L. Chang, and et al., "Task number maximization offloading strategy seamlessly adapted to uav scenario," *Computer Communications*, vol. 151, pp. 19–30, 2020.

[12] F. Guo, B. Tang, and et al., "Microservice selection in edge-cloud collaborative environment: A deep reinforcement learning approach," in *EdgeCom*. IEEE, 2021, pp. 24–29.

[13] H. Tian, X. Xu, and et al., "Dima: Distributed cooperative microservice caching for internet of things in edge computing by deep reinforcement learning," *World Wide Web*, pp. 1–24, 2021.

[14] Y. Zhai, T. Bao, L. Zhu, M. Shen, X. Du, and M. Guizani, "Toward reinforcement-learning-based service deployment of 5g mobile edge computing with request-aware scheduling," *IEEE Wireless Communications*, vol. 27, no. 1, pp. 84–91, 2020.

[15] C. Pham, D. T. Nguyen, N. H. Tran, K. K. Nguyen, and M. Cheriet, "Optimized iot service chain implementation in edge cloud platform: A deep learning framework," *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, pp. 538–551, 2021.

[16] C. Li, L. Zhu, W. Li, and Y. Luo, "Joint edge caching and dynamic service migration in sdn based mobile edge computing," *Journal of Network and Computer Applications*, vol. 177, p. 102966, 2021.

[17] X. Zhu, J. Liu, L. Lu, T. Zhang, T. Qiu, C. Wang, and Y. Liu, "Enabling intelligent connectivity: A survey of secure isac in 6g networks," *IEEE Communications Surveys & Tutorials*, 2024.

[18] Z. Fan and et al., "Serving at the edge: An edge computing service architecture based on icn," *ACM TOIT*, vol. 22, no. 1, pp. 1–27, 2021.

[19] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *ICML*. PMLR, 2018, pp. 1861–1870.

[20] S. Hu, Z. Qu, and et al., "Joint service request scheduling and container retention in serverless edge computing for vehicle-infrastructure collaboration," *IEEE Transactions on Mobile Computing*, 2023.

[21] F. Zhang, X. Tang, X. Li, S. U. Khan, and Z. Li, "Quantifying cloud elasticity with container-based autoscaling," *Future Generation Computer Systems*, vol. 98, pp. 672–681, 2019.

[22] H. Zhang, H. Ma, G. Fu, X. Yang, Z. Jiang, and Y. Gao, "Container based video surveillance cloud service with fine-grained resource provisioning," in *2016 IEEE 9th International Conference on Cloud Computing (CLOUD)*. IEEE, 2016, pp. 758–765.

[23] X. Wan, X. Guan, and et al., "Application deployment using microservice and docker containers: Framework and optimization," *Journal of Network and Computer Applications*, vol. 119, pp. 97–109, 2018.

[24] S. N. Srirama and et al., "Application deployment using containers with auto-scaling for microservices in cloud environment," *Journal of Network and Computer Applications*, vol. 160, p. 102629, 2020.

[25] X. Shang, Y. Huang, and *et al.*, "Reducing the service function chain backup cost over the edge and cloud by a self-adapting scheme," *IEEE Transactions on Mobile Computing*, vol. 21, no. 8, pp. 2994–3008, 2021.

[26] A. Jindal, V. Podolskiy, and M. Gerndt, "Performance modeling for cloud microservice applications," in *Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering*, 2019, pp. 25–32.

[27] B. Gao, Z. Zhou, and *et al.*, "An online framework for joint network selection and service placement in mobile edge computing," *IEEE Transactions on Mobile Computing*, vol. 21, no. 11, pp. 3836–3851, 2021.

[28] S. Y. Nikouei, Y. Chen, S. Song, B.-Y. Choi, and T. R. Faughnan, "Toward intelligent surveillance as an edge network service (isense) using lightweight detection and tracking algorithms," *IEEE Transactions on Services Computing*, vol. 14, no. 6, pp. 1624–1637, 2019.

[29] Q. Qu, R. Xu, and et al., "An experimental study on microservices based edge computing platforms," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications Workshops*. IEEE, 2020, pp. 836–841.

[30] X. Zhu, T. Qiu, and *et al.*, "Bls-location: A wireless fingerprint localization algorithm based on broad learning," *IEEE Transactions on Mobile Computing*, vol. 22, no. 1, pp. 115–128, 2021.

[31] M. Alam, J. Rufino, and et al., "Orchestration of microservices for iot using docker and edge computing," *IEEE Communications Magazine*, vol. 56, no. 9, pp. 118–123, 2018.

[32] Z. Gao, L. Yang, and Y. Dai, "Large-scale computation offloading using a multi-agent reinforcement learning in heterogeneous multi-access edge computing," *IEEE Transactions on Mobile Computing*, 2022.

[33] Z. Bai, Y. Lin, Y. Cao, and W. Wang, "Delay-aware cooperative task offloading for multi-uav enabled edge-cloud computing," *IEEE Transactions on Mobile Computing*, 2022.

[34] 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, 2019.

[35] Q. Pan, H. Cao, Y. Zhu, J. Liu, and B. Li, "Contextual client selection for efficient federated learning over edge devices," *IEEE Transactions on Mobile Computing*, 2023.

[36] W. Lv, Q. Wang, P. Yang, Y. Ding, B. Yi, Z. Wang, and C. Lin, "Microservice deployment in edge computing based on deep q learning," *IEEE Transactions on Parallel and Distributed Systems*, 2022.

[37] G. Shen, J. Zhang, A. Marshall, L. Peng, and X. Wang, "Radio frequency fingerprint identification for lora using deep learning," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 8, pp. 2604–2616, 2021.

[38] Y. Mao, C. You, and et al., "A survey on mobile edge computing: The communication perspective," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017.

[39] J. Chen, Y. Yang, C. Wang, H. Zhang, C. Qiu, and X. Wang, "Multi-task offloading strategy optimization based on directed acyclic graphs for edge computing," *IEEE Internet of Things Journal*, 2021.

[40] X. Li, X. Wang, P.-J. Wan, Z. Han, and V. C. Leung, "Hierarchical edge caching in device-to-device aided mobile networks: Modeling, optimization, and design," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 8, pp. 1768–1785, 2018.

[41] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[42] S. Oltra and O. Valero, "Banach's fixed point theorem for partial metric spaces," 2004.

[43] H. Arabnejad and J. G. Barbosa, "List scheduling algorithm for heterogeneous systems by an optimistic cost table," *IEEE transactions on parallel and distributed systems*, vol. 25, no. 3, pp. 682–694, 2013.

[44] F. Dai, G. Liu, and et al., "Task offloading for vehicular edge computing with edge-cloud cooperation," *World Wide Web*, pp. 1–19, 2022.

Chenyang Wang (Member, IEEE) received B.S. and M.S. degrees in computer science and technology from Henan Normal University, Xinxiang, China, in 2013 and 2017, respectively. He received his Ph.D. degree in 2023 from the School of Computer Science and Technology, College of Intelligence and Computing, Tianjin University, Tianjin, China. He is also a visiting PhD student under the support of the China Scholarship Council (CSC) at the School of Electrical Engineering, Aalto University in 2021. He is currently a postal researcher at the College of Computer Science and Software Engineering, Shenzhen University. His current research interests include edge intelligence, large models, active inference, and deep learning. He received the Best Student Paper Award of the 24th International Conference on Parallel and Distributed Systems by IEEE Computer Society in 2018. He also received the Best Paper Award from the IEEE International Conference on Communications in 2021. In 2022, he received the “IEEE ComSoc Asia-Pacific Outstanding Paper Award”.

Hao Yu (Member, IEEE) received the B.S. and Ph.D degree in communication engineering from the Beijing University of Posts and Telecommunications (BUPT), Beijing, China, in 2015 and 2020. He was also a Joint-Supervised Ph.D. Student with the Politecnico di Milano, Milano, Italy. He was a Postdoctoral Researcher with the school of Electrical Engineering, Aalto University, Espoo, Finland, in 2021, and with the Center of Wireless Communications (CWC), Oulu University, Oulu, Finland, from 2022 to 2023. He is currently a senior researcher in ICTFicial Oy, Finland. His research interests include intelligent edge network, time-sensitive networks, 6G deterministic networking.

Xiuhua Li (S’12, M’19) received his both B.S. degree and M.S. degree from Harbin Institute of Technology, China, in 2011 and 2013, respectively, and his Ph.D. degree from the Department of Electrical and Computer Engineering, University of British Columbia, Vancouver, Canada, in 2018. He is currently a professor with the School of Big Data & Software Engineering, Chongqing University, Chongqing, China, and also a member of the Haihe Laboratory of ITAI. He is the Head of the Institute of Intelligent Software and Services Computing associated with the Key Laboratory of Dependable Service Computing in Cyber Physical Society, Chongqing University, Education Ministry, China. His current research interests include edge computing and caching, edge intelligence, IoT and so on.

Fei Ma (Member, IEEE) received the B.S. degree in Communication Engineering from University of Electronic Science and Technology of China in 2017 and the Ph.D. degree in Information and Communication Engineering from Tsinghua University in 2022. His research interests include generative AI, multimodal learning, and emotional intelligence. During the PhD period, he was selected for the 2021 Tencent Rhino-Bird Elite Training Program.

Xiaofei Wang (Senior Member, IEEE) received the B.S. degree from Huazhong University of Science and Technology, China, and received M.S. and Ph.D. degrees from Seoul National University, Seoul, South Korea. He was a Postdoctoral Fellow with The University of British Columbia, Vancouver, Canada, from 2014 to 2016. He is currently a Professor with the College of Intelligence and Computing, Tianjin University, Tianjin, China. Focusing on the research of edge computing, edge intelligence, and edge systems, he has published more than 150 technical papers in IEEE JSAC, TCC, ToN, TWC, IoTJ, COMST, TMM, INFOCOM, ICDCS and so on. In 2017, he was the recipient of the “IEEE ComSoc Fred W. Ellersick Prize”, and in 2022, he received the “IEEE ComSoc Asia-Pacific Outstanding Paper Award”.

Tarik Taleb (Senior Member, IEEE) 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 currently a Professor at the Center of Wireless Communications, the University of Oulu, Finland. He is the founder and the Director of the MOSA!C Lab, Espoo, Finland. He was an Assistant Professor with the Graduate School of Information Sciences, Tohoku University, in a laboratory fully funded by KDDI until 2009. 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. 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, intervehicular communications, and social media networking.

Victor C. M. Leung (Life Fellow, IEEE) is a Distinguished Professor of Computer Science and Software Engineering at Shenzhen University, China, and an Emeritus Professor of Electrical and Computer Engineering at the University of British Columbia, Canada. He has co-authored numerous refereed articles in the broad areas of wireless networks and mobile systems, and is a Clarivate Analytics “Highly Cited Researcher”. Dr. Leung is an editor of the IEEE TGCN, IEEE TCSS, and several other journals. He received the 2017 IEEE ComSoc Fred W. Ellersick Prize, 2017 IEEE Systems Journal Best Paper Award, 2017 Canadian Award for Telecommunications Research, 2018 IEEE TCGCC Distinguished Technical Achievement Recognition Award, 2018 IEEE CSIM Best Journal Paper Award, and 2019 IEEE TCGCC Best Journal Paper Award, as well as several other awards. He is a Life Fellow of IEEE, and a Fellow of the Royal Society of Canada, Canadian Academy of Engineering, and Engineering Institute of Canada.