Reinforcement Learning-Based Approach for Microservices-Based Application Placement in Edge Environment

Kaouther Gasmi
Communication System Laboratory
National Engineering School of Tunis
University of Tunis El Manar
Tunis, Tunisia
kaouther.gasmi@enit.utm.tn

Kamel Abassi
Communication System Laboratory
National Engineering School of Tunis
University of Tunis El Manar
Tunis, Tunisia
kamel.abassi@enit.utm.tn

Lamia Romdhani
Core Curriculum Program
Qatar University
Doha, Qatar
lamia.romdhani@qu.edu.qa

Olivier Debauche
Faculty of Engineering - ILIA / Infortech
University of Mons
Mons, Belgium
0000-0003-4711-2694

***Abstract*—Edge computing allows for the deployment of applications near end-users, resulting in low-latency real-time applications. The adoption of the microservices architecture in modern applications has made this possible. Microservices architecture describes an application as a collection of separate but interconnected entities that can be built, tested, and deployed individually. Each microservice runs in its own process and exchanges data with others. Instead, edge nodes can independently deploy microservices-based IoT applications. Consistently meeting application service level objectives while also optimizing service placement delay and resource utilization in an edge environment is non-trivial. The present paper introduces a dynamic placement strategy that aims to fulfill application constraints and minimize infrastructure resource usage while ensuring service availability to all end-users of the UE in the edge network.**

***Index Terms*—Dynamic service placement, Microservice-based IoT application, Reinforcement learning, Edge computing.**

I. Introduction

In recent years, the Internet of Things (IoT) has developed quickly, and the ever-increasing number and variety of connected devices generate massive amounts of data related to a wide range of smart applications including smart agriculture, smart power grid, smart cities, smart health, and smart transportation, which offer seamless services that enhance our daily lives. Recently, IoT application development is adapting microservices Architecture (MSA) to support the fast evolution of IoT application development toward growing an IoT environment. MSA builds applications as collections of modules known as microservices that are independently deployable and scalable [3].

Meanwhile, Edge computing is emerging as a powerful distributed computing paradigm for hosting latency-critical and real-time IoT applications (e.g. self-driving cars, smart manufacturing, driving assistant service) through its proximity characteristic. Edge computing extends cloud-like services toward the edge of the network by using computing, networking, and storage resources. Indeed, Cloud computing cannot satisfy the low latency requirement of such applications due to high end-to-end transmission latency [1].

While Edge computing offers numerous benefits to contemporary applications, it faces several challenges in delivering services, including limited processing and storage capabilities, as well as battery power in the edge network, geo-dispersed Edge nodes, heterogeneous application requirements, and dynamic service demands [1]. In this context, an important consideration is selecting the appropriate Edge nodes with sufficient hosting capacity to deploy and run multiple applications based on their respective demands, constraints, and performance criteria. As Edge nodes may not possess adequate resources to handle all end-users service requests, another significant challenge is optimizing the utilization of edge servers' resources while deploying the requested applications efficiently. Given that modern IoT applications have adopted the MSA architecture, the present paper proposes a dynamic MAS-based IoT application placement strategy across multiple edge nodes that aims to satisfy application requirements in terms of service latency while minimizing edge servers' resource utilization in hosting the requested application.

To solve the problem, the stochastic traffic and communication uncertainty in the Edge environment should be carefully addressed. Q-learning is a model-free reinforcement learning approach that works well in contexts with unknown and dynamic models. Such a feature makes the Q-learning method suitable for solving the problem of microservices-based application placement in edge network

environments, where network circumstances and workload patterns are very dynamic and unclear. our study contributes to the dynamic MSA-based service placement problem by employing a reinforcement learning-based dynamic strategy to provide services with a low delay while keeping the node's resource utilization low. Q-learning's most important component is estimating future benefits accurately and efficiently to optimize long-term decisions. Our aim is to find a placement approach that maximizes performance while considering latency, resource utilization, and user pleasure. Therefore, Q-Learning's reward prediction fits our optimization goals.

The present paper contributes in the following ways:

- Firstly, it addresses online MSA-based service placement problem in edge environment and proposes a learning-based approach that can efficiently solve the joint optimization of the requested service delay and resource utilization of the edge nodes that hosting the application with dynamic service requests arrivals.
- Secondly, the solution presented in this paper uses Q-learning in a novel way and implements an MSA-based service placement strategy that decouples the problem and recursively applies Q-learning to place a set of application services.

This paper is organized as follows. In the "Related work" section, we provide a comprehensive overview and discuss the most relevant studies conducted earlier on the same subject and their results. Details about the Edge environment used in our research study are presented in the "Edge Architecture" section. In addition, service problems, models, and network models are described in the "Service Statement Problem" section. Then, our proposed RL-Dynamic Service Placement approach is introduced in the next section. Finally, the paper concludes in the "Conclusion and Future Works" section, which presents a forward-looking vision and insight into future works.

II. Related works

Several studies in the literature that address service placement in the context of Edge computing to optimize network resource utilization [6], [9], reducing latency [7], [10], [13], [15], Computing resource utilization [13], deployment cost [8], [10], [11].

The Reinforcement and deep learning methods have gained considerable attention as an alternative approach to solving service placement problems in Edge computing such as Q-learning, deep Q-network (DQN) parameterized deep Q networks (P-DQN), and Deep Neural Networks (DNN). Authors in [8] applied Reinforcement Q-learning to minimize deployment costs while placing the requested services. In [6], authors propose a deep reinforcement Q-learning algorithm to place service in the edge network to optimize the overall network utility. They assume that the requested service could be modeled as a tree of sub-task with data flows to satisfy data analytics applications requirements. In [7], authors proposed an approach based on parameterized deep Q networks to make the joint service placement and computation resource allocation decisions, to minimize the total latency of tasks in terms of allocated memory. Also, authors in [9] proposed DQN based strategy to optimize the network system utility while placing service tasks.

Heuristic methods (e.g. genetic algorithms) have also been investigated to deal with service placement problems in the context of Edge computing. Authors of [10] proposed a genetic algorithm and RL-based method to address the imp12*act of service migration on request response time while considering resource capacity. Most of the existing works deal with the service placement problem focusing on monolithic application services.

Different from the existing works, our study contributes to the dynamic MSA-based service placement problem by employing a reinforcement learning-based dynamic strategy to provide services with a low delay while keeping the node's resource utilization low. The primary component of our proposed strategy is the integration of deep reinforcement Q-learning, which has been successfully utilized in addressing numerous network optimization problems, including scheduling [16], caching [18], and security in the edge network [17]. Our proposed approach involves a Q-learning agent that interacts with the system environment, selects placement decisions (i.e., actions), and enhances the decision-making of service placement based on experience, without previous information of the state transition stochastic in the environment. Moreover, the approach can adapt to the dynamic nature of environmental changes during online training.

Table I summarizes the recent existing work providing a brief overview of the used methodology, the application request model, and performance metrics.

TABLE I
SUMMARY OF THE RECENT RELATED WORK. COLUMN ABBREVIATIONS: DEPLOYMENT COST (D.), LATENCY (L.), NETWORK RESOURCE USAGE, (N.), COMPUTING RESOURCE UTILIZATION (C.).

Study	Methodology	Application	Metrics			
			D	L	N	C
[6]	DRL	Tree	×	×	✓	×
[7]	P-DQN	monolithic	×	✓	×	×
[8]	Q-learning, Heuristic	MSA-based	×	✓	×	×
[9]	DQN	monolithic	×	×	✓	×
[10]	Heuristic, RL	monolithic	×	✓	×	✓
[11]	Genetic algorithm	monolithic	✓	×	×	×
[12]	Genetic algorithm	monolithic	×	✓	×	✓
[13]	DNN	monolithic	×	✓	×	✓
[14]	DRL	monolithic	×	✓	×	✓
[15]	P-DPN	monolithic	×	✓	×	×
Proposed	Proposed approach	MSA-based	×	✓	×	✓

III. Edge architecture

Figure 1 presents the edge computing architecture, which uses four layers: the IoT layer, the edge network layer, the fog

layer, and the cloud layer and each layer in the architecture has distinct roles and responsibilities.

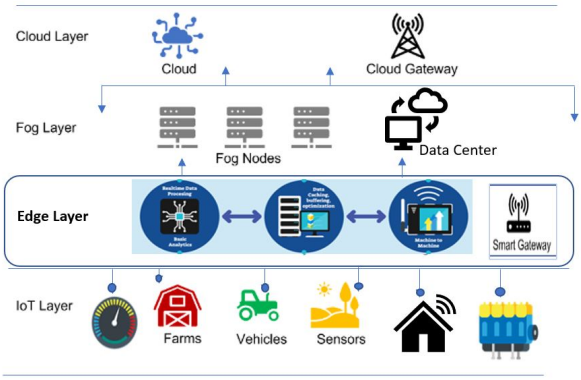

Fig. 1. IoT-to-Cloud architecture layers

A. IoT Layer

This tier consists of all the IoT devices connected to the internet including several types of devices such as sensor nodes, smart vehicles, smartphones, tablets, and others. These IoT devices are known as unintelligent devices since they only generate data and cannot process it. The primary purpose of this tier is to acquire, measure and collect data from these devices and send it to the following layer. In addition, the computation resources of IoT devices assigned to user devices are limited, they have low computational power, constrained by battery life and low processing capacity. Hence, devices can request the closest fog node to offload tasks that require many computation resources to run and accelerate the computation. This offloading request is placed in the output queue at the edge layer to be transferred to either the fog layer or the cloud layer.

B. Edge Layer

Edge computing refers to a new computing model which implements the computation of tasks at the periphery of the network known as the edge layer. Based on this concept, certain applications that may not require a significant amount of computing resources can be processed in the edge layer and not needed to be offloaded in the fog or the cloud. However, if resources at the edge layer are not available, the sensors will request to offload their tasks in the fog layer or the cloud layer for processing.

C. Fog Layer

The tier commonly known as the fog computing layer, serves to bring networking resources closer to the underlying networks. It acts as a mediator between the cloud and the edge layer, allowing for the deployment of latency-aware services and applications. This layer consists of several fog nodes (virtual or physical) with high processing capabilities. The fog nodes within this layer include network devices such as routers, switches, and Access Points (APs). The main objective of this layer is to handle the closest and most time-sensitive requests from nodes, while also enabling collaborative sharing of storage and computing facilities.

Both fog computing and edge computing aim to bring computing resources closer to end-user devices. However, fog computing uses intermediate fog nodes to collect and pre-process data before sending it to the cloud, while edge computing directly places computational capabilities close to the end devices, allowing real-time processing and decision-making without relying heavily on the cloud infrastructure.

D. Cloud Layer

The topmost layer of the edge-fog computing architecture is known as the Cloud Layer, it is the layer that includes various cloud servers and cloud Data centers with significantly more computation power, efficient storage, and processing capabilities than the fog layer. In addition, Cloud computing is an internet-based computing approach that offers a set of several resources such as services, processors, and storage to run the application. Although the cloud computing strategy can handle a substantial volume of offloaded tasks from the end devices, the cloud layer poses challenges due to limited bandwidth and the remote location of resources from end devices. As a result, it is essential to offload tasks close to the sources, and fog nodes can be used for service placement.

IV. Service Placement Problem

In the context of edge computing, we address the challenge of online service placement for micro-service-based applications. Specifically, the objective of the service placement problem is twofold: (i) identifying a suitable set of edge nodes that possess available computing resources to process the requested service and (ii) identifying a set of links to connect the sources with destinations for efficient data transfer. This problem presents a joint online optimization challenge due to the tight coupling of computing and network resources, which must be allocated for each incoming edge computing request in a challenging online environment.

A. Service request model

Let A be a microservices-based IoT applications where consists of a set of independently deployable and scalable microservices M_a. They communicate to create a set of services S_a provided to the application users. We use the term "service" to denote end-user-requested business functionalities, which can be either an atomic service (consisting of only a single microservice) or a composite service (composed of multiple microservices). An application A can be modeled using Directed Acyclic Graphs (DAG) [19] where the vertices represent microservices $m \in M_a$. Edges in DAG represent microservice invocations. Microservices are independently packaged and have heterogeneous resource requirements (RAM, CPU, storage, etc.) with performance requirements that can be defined at the service level. Each edge in the graph represents the required data flow $f \in F_a$ between microservices. Each

application can be denoted as a tuple of $(M_a, S_a, F_a, R_s, D_s)$ where each $s \in S_a$ is depicted by a tuple (r_m, d_m).

r_m is the required computing resource and d_m the required delay. For a micro-service to be assigned to an edge node, the computing resource capacity of the node must be sufficient. If the available computing resources on an edge node can accommodate multiple micro-services, then these services can be mapped to the same node.

B. Network model

We consider edge computing denoted as a graph, $G = (V, E)$, where each vertex $v \in V$ represents an edge node, and each edge $e \in E$ represents a physical link in the edge computing. For each edge node e, the residual computing resources (available resources) are denoted by C_e. The amount of resources

We group multiple resources into containers on individual edge nodes to optimize edge computing resources. These containers are then tracked using a vector Ct, which represents the number of available containers on each edge node for allocating new computing sub-tasks.

C. Computing model

The total service delay refers to the whole time from when an IoT device sends a service request s to when the corresponding response is received. This delay is estimated based on the propagation delay, the waiting time, and the processing time of a service s as follows:

$$d_e^s = T_{pro} + T_t + T_q + T_p \tag{1}$$

We define Q_E as the incoming traffic flow in the edge. The waiting time in Q_E is calculated based on the time between IoT services reaching the edge nodes, where the input rate is assumed by the Poisson distribution. Let λ be the input rate of requests and μ is the rate of distribution of IoT services. The waiting time for service s will be estimated as follows:

$$T_q^s = \frac{1}{\mu - \lambda} - \frac{1}{\mu} \tag{2}$$

The average propagation delay required to transfer data is calculated based on the ratio between the distance between the IoT device and the edge node e and the propagation speed c over the communication medium:

$$T_p = \frac{dist(s)}{c} \tag{3}$$

Where $dist(s)$ is the Euclidean distance between the IoT device and the receiving edge node v.

The edge resource utilization ϕ_e is defined as the ratio between the resources that the service s will consume and the remaining resources at the edge node hosting the service:

$$\phi_e = \frac{r_s}{C_e} \forall e \in E, \forall s \in S \tag{4}$$

V. Proposed RL-Dynamic service placement approach

This section introduces our proposed RL-Dynamic service placement method. A brief overview of the design of the state space, action space, policy, and reward function employed in our RL-based approach is provided.

A. State space

In the context of service graph placement, we adopt a discrete-time model where the time slots are designed to be sufficiently short. Specifically, each time slot accommodates at most one request arrival to enable efficient management of the service graph placement problem. The state space of the RL model represents the network environment at a specific time slot, denoted as t. The agent of the RL model observes the environment and constructs the set of data $\omega_t(g)$ from the service request model.

$$\begin{aligned}\omega_{t(g)} =&[(UE_1, g_1, R_{s,1}, D_{s,1}, t)], [(UE_2, g_2, R_{s,2}, D_{s,2}, t)] \\ &, ..., [(UE_n, g_n, R_{s,n}, D_{s,n}, t)]\end{aligned} \tag{5}$$

For the graph $g \in G$ describing micro-services in a given application to be deployed at time slot t, $UE_1, UE_2, ..., UE_n$ is the set of UE requesting for service g. The target micro-services graph is defined by its topology as well as resource and delay requirements associated with each vertex.

B. Action space

The action space in our proposed RL-based service placement approach represents the set of possible actions that can be taken by the policy module for the placement of micro-services on a set of available edge nodes. The action space is illustrated in Fig. 2. Let a denote the action space, then the action taken by the policy module for the placement of micro-services at time slot t can be defined as:

$$a = \pi(\gamma) = x_s, \forall s \in S, \tag{6}$$

where π is the policy called for generate an action over the observation set of γ at time slot t, and x_s refers to the decision matrix indicating the service placement s on the set of edge nodes V. A high Q-value means a high-quality decision.

C. Policy function

The policy function π is responsible for mapping the state space to an action space a. In our RL dynamic-based approach, the policy module aims to optimize the objective function while considering various constraints, as depicted in Fig. 2. The objective function of the proposed approach aims to balance the utilization of edge resources and the service delay. By minimizing the maximum edge resource utilization and service delay, the approach can efficiently utilize the limited edge resources and satisfy the delay requirements of users. The parameter α controls the relative importance of resource usage versus service delay. The policy function π is formulated as follows:

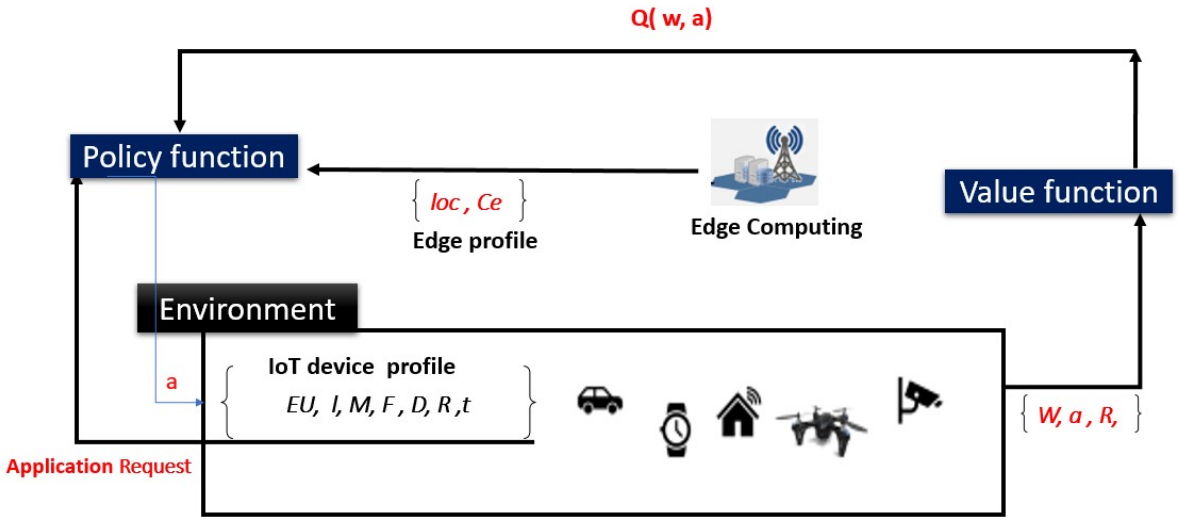

Fig. 2. The proposed DRL- Service placement approach system.

$$\phi = minmax_{s\in S, e\in E}[\alpha \sum_{e\in E} \phi_e + (1-\alpha) \sum_{e\in E} x_e^s d_e^s \leqslant D_s]; \quad \forall s \in S \tag{7}$$

where s_t is the state at time slot t, $\mathcal{A}(s_t)$ is the set of feasible actions in state s_t, u_i^{t-1} is the resource utilization of edge node i in the previous time slot $t-1$, C_i is the capacity of edge node i, $\omega_t(g_{ij})$ is the demand of micro-service j in service group g_{ij} at time slot t, a_{ij} is a binary decision variable indicating whether micro-service j is placed on edge node i or not, $\mathcal{N}$ is the set of all edge nodes, $\mathcal{M}$ is the set of all micro-services, D_{ij} is the maximum tolerable delay for service group g_{ij}, and α is a parameter that controls the relative importance of resource usage vs. service delay. The policy function selects an action that minimizes the objective function, which is a weighted sum of the maximum edge resource utilization and service delay.

The goal of this problem is to minimize the total service delay experienced by the UE while also minimizing the maximum utilization of edge resources.

the placement of a micro-service s at edge node e is given by x_e^s, where x_e^s is a binary variable. If the micro-service s is placed onto e, $x_e^s = 1$. Otherwise is 0. The placement of micro-service is subjective to the following constraints:

Mapping constraints: Ensures that each micro-service should be deployed onto exactly one edge node:

$$\sum_{e\in E} x_e^s = 1; \forall s \in G \tag{8}$$

Each data flow between two micro-services is mapped to exactly one network link:

$$\sum_{l\in L} x_l^d = 1; \forall d \in G \tag{9}$$

Where x_i^s is a binary variable used to indicate the placement of service s at node i. If edge node i deploys service s, x_i^s is 1. Otherwise, it is 0.

Delay constraint: this constraint ensures that the micro-service delay should be less than the micro-services maximum delay threshold D_s

$$\sum_{s\in G} x_e^s d_e^s \leqslant D_s; \forall e \in E \tag{10}$$

Where T_{pro} is processing time; T_t is transmission delay; T_q is queuing delay and T_p is propagation

Resource constraint: ensure that the available resource at the edge node is not exhausted while deploying micro-service s.

D. Reward function

At each time slot t, in response to the action taken by the RLD agent, an immediate reward $r(\omega, a)$ is sent by the environment. The primary goal of the DRL agent is to maximize the reward. However, the objective of our service placement problem is to minimize the service delay requested by the end user from from the corresponding edge node. Hence, the reward function is estimated as:

$$r(\omega, a) = d_s^s(t) \tag{11}$$

E. RLD agent

The RLD agent has direct interaction with the environment. From the environment, the request for service s is initiated by UE following the service request model. In return, considering the demand for service s at time slot t and location l of EU requesting for service s, the policy function module selects the edge servers for the placement services based on the action selection strategy π, as discussed in Section IV-C. The task of the value function module is to critique the performance of the policy module based on the action taken and rewards received. It is responsible for calculating the quality value $Q(w_t(s), a_t)$ of the decision taken by the policy module. A high $Q(w_t(s), a_t)$ means a high-quality decision. Therefore, the policy module has to select actions with the maximum quality value :

$$a = maxQ(w_t(s), a_t) \tag{12}$$

The Q-value of a state at a given time is calculated as:

$$\begin{aligned} Q'(w_t(s), a_t) =& r(w_t(s), a_t) \\ &+ \gamma(maxQ(w_{t+1})(s), a_t)) \\ &+ (\alpha - 1)(Q(w_t)(s), a_t)) \end{aligned} \tag{13}$$

Where $(w_t(s), a_t)$ is a state action pair at time t and $r(w_t(s), a_t)$ is the reward of applying action a_t at state $w_t(s)$. The value γ and α are the learning rate and discount factor, respectively.

VI. Discussion and Conclusion

In this study, we present a service placement strategy for a microservice-based IoT application in an edge environment where the user sends a service request, and the placement algorithm guarantees the availability of the requested service under constraints including service latency and resource utilization to satisfy the end user experiences. We use Machine learning techniques, such as Reinforcement Learning, that can adapt to the highly dynamic nature of Edge environments and microservices. To the best of our knowledge, no other related studies have focused on optimizing these two aspects while dealing with service placement problems in an edge environment. In future research, we plan to implement the proposed approach and compare it with several optimization methods with our proposed approach. We will also explore adding constraints to our study (e.g., deadlines).

References

[1] Yousefpour, Ashkan, et al. "All one needs to know about fog computing and related edge computing paradigms: A complete survey." Journal of Systems Architecture 98 (2019): 289-330.

[2] Cao, Keyan, et al. "An overview on edge computing research." IEEE Access 8 (2020): 85714-85728.

[3] Atitallah, Safa Ben, Maha Driss, and Henda Ben Ghzela. "Microservices for Data Analytics in IoT Applications: Current Solutions, Open Challenges, and Future Research Directions." Procedia Computer Science 207 (2022): 3938-3947.

[4] Maia, Adyson Magalhães, et al. "Dynamic service placement and load distribution in edge computing." 2020 16th International Conference on Network and Service Management (CNSM). IEEE, 2020.

[5] B. Gao, Z. Zhou, F. Liu, F. Xu and B. Li, "An Online Framework for Joint Network Selection and Service Placement in Mobile Edge Computing," in IEEE Transactions on Mobile Computing, vol. 21, no. 11, pp. 3836-3851, 1 Nov. 2022, doi: 10.1109/TMC.2021.3064847.

[6] Y. Wang, Y. Li, T. Lan and N. Choi, "A Reinforcement Learning Approach for Online Service Tree Placement in Edge Computing," 2019 IEEE 27th International Conference on Network Protocols (ICNP), Chicago, IL, USA, 2019, pp. 1-6, doi: 10.1109/ICNP.2019.8888150.

[7] T. Liu, S. Ni, X. Li, Y. Zhu, L. Kong and Y. Yang, "Deep Reinforcement Learning based Approach for Online Service Placement and Computation Resource Allocation in Edge Computing," in IEEE Transactions on Mobile Computing, doi: 10.1109/TMC.2022.3148254.

[8] Garg, Dhruv, Nanjangud C. Narendra, and Selome Tesfatsion. "Heuristic and reinforcement learning algorithms for dynamic service placement on mobile edge cloud." arXiv preprint arXiv:2111.00240 (2021).

[9] P. Zhou, G. Wu, B. Alzahrani, A. Barnawi, A. Alhindi and M. Chen, "Reinforcement Learning for Task Placement in Collaborative Cloud-Edge Computing," 2021 IEEE Global Communications Conference (GLOBECOM), Madrid, Spain, 2021, pp. 1-6, doi: 10.1109/GLOBECOM46510.2021.9685049.

[10] Farhadi, Vajiheh et al. "Service Placement and Request Scheduling for Data-intensive Applications in Edge Clouds." IEEE INFOCOM 2019 - IEEE Conference on Computer Communications (2019): 1279-1287.

[11] Maia, Adyson Magalhães, et al. "Dynamic service placement and load distribution in edge computing." 2020 16th International Conference on Network and Service Management (CNSM). IEEE, 2020.

[12] Gao, Yicheng, and Giuliano Casale. "JCSP: Joint caching and service placement for edge computing systems." 2022 IEEE/ACM 30th International Symposium on Quality of Service (IWQoS). IEEE, 2022.

[13] Premsankar, Gopika, and Bissan Ghaddar. "Energy-efficient service placement for latency-sensitive applications in edge computing." IEEE internet of things journal 9.18 (2022): 17926-17937.

[14] A. Talpur and M. Gurusamy, "DRLD-SP: A Deep-Reinforcement-Learning-Based Dynamic Service Placement in Edge-Enabled Internet of Vehicles," in IEEE Internet of Things Journal, vol. 9, no. 8, pp. 6239-6251, 15 April 15, 2022, doi: 10.1109/JIOT.2021.3110913.

[15] T. Liu, S. Ni, X. Li, Y. Zhu, L. Kong and Y. Yang, "Deep Reinforcement Learning based Approach for Online Service Placement and Computation Resource Allocation in Edge Computing," in IEEE Transactions on Mobile Computing, doi: 10.1109/TMC.2022.3148254.

[16] J. Zhu, Y. Song, D. Jiang, and H. Song. A new deep-q-learning-based transmission scheduling mechanism for the cognitive internet of things. IEEE Internet of Things Journal, 5(4):2375–2385, 2018.

[17] L. Xiao, X. Wan, C. Dai, X. Du, X. Chen, and M. Guizani. Security in mobile edge caching with reinforcement learning. IEEE Wireless Communications, 25(3):116–122, 2018.

[18] Y. He, N. Zhao, and H. Yin. Integrated networking, caching, and computing for connected vehicles: A deep reinforcement learning approach. IEEE Transactions on Vehicular Technology, 67(1):44–55, 2018.

[19] Pallewatta, Samodha, Vassilis Kostakos, and Rajkumar Buyya. "QoS-aware placement of microservices-based IoT applications in Fog computing environments." Future Generation Computer Systems 131 (2022): 121-136.