Deep Reinforcement Learning for Adaptive Network Slicing in 5G for Intelligent Vehicular Systems and Smart Cities

Almuthanna Nassar and Yasin Yilmaz, *Senior Member, IEEE*

***Abstract*—Intelligent vehicular systems and smart city applications are the fastest growing Internet-of-Things (IoT) implementations at a compound annual growth rate of 30%. In view of the recent advances in IoT devices and the emerging new breed of IoT applications driven by artificial intelligence (AI), the fog radio access network (F-RAN) has been recently introduced for the fifth-generation (5G) wireless communications to overcome the latency limitations of cloud-RAN (C-RAN). We consider the network slicing problem of allocating the limited resources at the network edge (fog nodes) to vehicular and smart city users with heterogeneous latency and computing demands in dynamic environments. We develop a network slicing model based on a cluster of fog nodes (FNs) coordinated with an edge controller (EC) to efficiently utilize the limited resources at the network edge. For each service request in a cluster, the EC decides which FN to execute the task, i.e., locally serve the request at the edge, or to reject the task and refer it to the cloud. We formulate the problem as infinite-horizon Markov decision process (MDP) and propose a deep reinforcement learning (DRL) solution to adaptively learn the optimal slicing policy. The performance of the proposed DRL-based slicing method is evaluated by comparing it with other slicing approaches in dynamic environments and for different scenarios of design objectives. Comprehensive simulation results corroborate that the proposed DRL-based EC quickly learns the optimal policy through interaction with the environment, which enables adaptive and automated network slicing for efficient resource allocation in dynamic vehicular and smart city environments.**

***Index Terms*—Deep reinforcement learning (DRL), edge computing, fog RAN, intelligent vehicular systems, network slicing.**

I. Introduction

THE FIFTH-GENERATION (5G) wireless communication systems will enable massive Internet of Things (IoT) with deeper coverage, very high data rates of multi giga-bit-per-second (Gbps), ultralow latency, and extremely reliable mobile connectivity [1], [2]. It is anticipated that the IoT devices will constitute the 50% of the 29.3 billion connected devices globally by 2023, where Internet of Vehicles (IoV) and smart city applications are the fastest growing IoT implementations at annual growth rates of 30% and 26%, respectively [3]. The emerging new breed of IoT applications which involve video analytics, augmented reality (AR), virtual reality (VR), and artificial intelligence (AI) will produce an annual worldwide data volume of 4.8 zettabyte by 2022, which is more than 180 times the data traffic in 2005 [4]. Equipped with variety of sensors, radars, lidars, ultrahigh definition (UHD) video cameras, GPS, navigation system, and infotainment facilities, a connected and autonomous vehicle (CAV) will generate 4.0 terabyte of data in a single day, of which 1.0 gigabyte need to be processed every second [5].

A. Cloud and Fog RAN

Through centralization of network functionalities via virtualization, cloud radio access network (C-RAN) architecture is proposed to address the big data challenges of massive IoT. In C-RAN, densely deployed disseminated remote radio units (RRUs) are connected through high capacity fronthaul trunks to a powerful cloud controller (CC) where they share a vast pooling of storage and baseband units (BBUs) [6]. The centralized computing, processing, and collaborative radio in C-RAN improves network security, flexibility, availability, and spectral efficiency. It also simplifies network operations and management, enhances capacity, and reduces energy usage [7]. However, considering the fast growing demands of IoT deployments, C-RAN lays overwhelming onus on cloud computing and fronthaul links, and dictates unacceptable delay caused mainly by the large return transmission times, finite-capacity fronthaul trunks, and flooded cloud processors [8]. The latency limitation in C-RAN makes it challenging to meet the desired Quality-of-Service (QoS) requirements, especially for the delay-sensitive IoV and smart city applications [9]. Hence, an evolved architecture, fog RAN (F-RAN), is introduced to extend the inherent operations and services of cloud to the edge [10]. In F-RAN, the fog nodes (FNs) are not only restricted to perform the regular radio frequency (RF) functionalities of RRUs but they are also equipped with computing, storage, and processing resources to afford the low latency demand by delivering network functionalities directly at the edge and independently from the cloud [11]. However, due to their limited resources compared to the cloud, FNs are unable to serve all requests from IoV and smart city applications, and hence, they should utilize their limited resources

Manuscript received December 20, 2020; revised March 16, 2021 and June 5, 2021; accepted June 7, 2021. Date of publication June 23, 2021; date of current version December 23, 2021. This work was supported in part by the U.S. National Science Foundation (NSF) under Grant ECCS-2029875. *(Corresponding author: Almuthanna Nassar.)*

The authors are with the Electrical Engineering Department, University of South Florida, Tampa, FL 33620 USA (e-mail: atnassar@usf.edu; yasiny@usf.edu).

Digital Object Identifier 10.1109/JIOT.2021.3091674

intelligently to satisfy the QoS requirements in synergy and complementarity with the cloud [12].

B. Network Slicing for Heterogeneous IoV and Smart City Demands

IoV and smart city applications demand various computing, throughput, latency, availability, and reliability requirements to satisfy a desired level of QoS. For instance, in-vehicle audio, news, and video infotainment services are satisfied by the traditional mobile broadband (MBB) services of high throughput and capacity with latency greater than 100 ms [13]. Cloud computing plays an essential role for such delay-tolerant applications. Other examples of delay-tolerant applications include smart parking [14], intelligent waste management [15], infrastructure (e.g., bridges, railways, etc.) monitoring [16], air quality management [17], noise monitoring [18], smart city lighting [19], smart management of city energy consumption [20], and automation of public buildings, such as schools, museums, and administration offices to automatically and remotely control lighting and air condition [21].

On the other hand, latency and reliability are more critical for other IoV and smart city applications. For instance, deployment scenarios based on enhanced MBB (eMBB) require latency of 4.0 ms. Enhanced vehicle-to-everything (eV2X) applications demand 3–10 ms latency with a packet loss rate of 10^{-5}. Ultrareliable and low-latency communications (URLLCs) seek a latency level of 0.5–1.0 ms and 99.999% reliability [22], [23], e.g., autonomous driving [24]. AI-driven and video analytics services are considered both latency-critical and compute-intensive applications [25]. For instance, real-time video streaming for traffic management in intelligent transportation system (ITS) [26] requires a frame rate of 100 Hz, which corresponds to a latency of 10 ms between frames [13]. Future electric vehicles (EVs) and CAVs are viewed as computers on wheels (COWs) rather than cars because they are equipped with super computers to execute extremely intensive computing tasks, including video analytics and AI-driven functionalities. However, with the high power consumption associated with such intense computing, COWs capabilities are still bounded in terms of computing power, storage, and battery life. Hence, computing offloading to fog and cloud networks is inevitable [27]. Especially in a dynamic traffic and load profiles of dense IoV and smart city service requests with heterogeneous latency and computing needs, partitioning RAN resources virtually, i.e., network slicing [28], assures service customization.

Network slicing is introduced for the evolving 5G and beyond communication technologies as a cost-effective solution for mobile operators and service providers to satisfy various user QoS [29]. In network slicing, a heterogeneous network of various access technologies and QoS demands that share a common physical infrastructure is logically divided into virtual network slices to improve network flexibility. Each network slice acts as an independent end-to-end network and supports various service requirements and a variety of business cases and applications. In this work, we consider the network slicing problem of adaptively allocating the limited edge computing and processing resources in F-RAN to dynamic IoV and smart city applications with heterogeneous latency demands and diverse computing loads.

II. Related Work

There is an increasing number of works in the literature focusing on network slicing as an emerging network architecture for 5G and future technologies. Issues and challenges of network slicing as well as the key techniques and solutions for resource management are considered in [28]. The work in [30] provides an overview of various use cases and network requirements of network slicing. Network slicing for resource allocation in F-RAN is considered in [31]–[33], where network is logically partitioned into two slices: 1) a high downlink-transmission-rate slice for MBB applications and 2) a low-latency slice to support URLLC services. While [31] focuses on efficiently allocating radio resources and satisfying various QoS requirements, [32] investigates a joint radio and caching resource allocation problem. For massive IoT environment, Sun *et al.* [33] proposed a hierarchical architecture in which a global resource manager allocates the radio resources to local resource managers in slices, which assign resources afterward to their users. Two-level resource management in C-RAN is explored in [34] and [35]: 1) an upper level for allocating fronthaul capacity and computing resources of C-RAN among multiple wireless operators and 2) a lower level for controlling the allocation of C-RAN radio resources to individual operators.

Reinforcement learning (RL) is embraced as a powerful tool to deal with dynamic network slicing for adaptive resource allocation in F-RAN. In [4], [29], and [36], the RL methods of Q-learning (QL), Monte Carlo, SARSA, expected SARSA, and dynamic programming are utilized to learn the optimal resource allocation policy for a single FN. The work [37] follows the problem formulation in [4] with an extension to spectrum sharing between 5G users and incumbent users. As the complexity of the control problem increases with more FNs, deep RL (DRL), which integrates deep neural networks (DNNs) with RL, is more advantageous to cope with the large state and action spaces [38].

A. Deep RL for Network Slicing

Applying DRL as a solution for core network slicing is investigated in [39] and [40]. In [39], a particular scenario with three service types (VoIP, video, and URLLC) and hundred users is considered. Resource reconfiguration of core network slices is studied in [40] with the aim of minimizing long-term resource consumption. DRL-based centralized agent for C-RAN slicing is investigated in [41]–[45]. In [41], the deep Q-network (DQN) is utilized by a cloud server to optimally manage centralized caching and radio resources and support two transmission-mode network slices: 1) hotspot slice, which supports high-transmission-rate users for MBB applications and 2) vehicle-to-infrastructure slice for delay-guaranteed transmission. For better radio resource management, a DRL agent is used as a slice manager in [43] to schedule users into three slice types: 1) best effort rate; 2) constant bit rate;

and 3) minimum bit rate slices. In [44], a DRL agent at a centralized controller manages the allocation of shared radio resources (bandwidth) among multiple base stations and different network slices (VoLTE, eMBB, and URLLC) to maximize spectrum efficiency and service-level agreements. A general network slicing model is proposed by [45] in which an owner of a network provides physical resources to tenants (service providers) to meet the service demand of their end users. With the aim of maintaining high quality of service and maximizing the long-term revenue of service providers by minimizing the reconfiguration cost, a centralized DRL agent reconfigures the allocated resources for two network slices: 1) eMBB and 2) URLLC.

The single base station slicing model is considered in [42] and [46]–[49]. C-RAN with single base station is investigated in [42], where generative-adversarial-network distributed DQN (GAN-DDQN) is examined for dynamic bandwidth slicing among network slices, each of which supports users of a particular service type. The dependency of radio resource allocation and the number of slices supported by a single BS is studied by [46], in which distributed DRL is utilized to achieve optimal and flexible radio resource allocation regardless of the number of slices. The work in [47] follows [39] where a single base station provides three service types (video, VoLTE, and URLLC) and decides the bandwidth to allocate for each user request. In a vehicular network, [48] employs a DRL agent at a single base station to allocate radio resources for users that belong to four slices: cellular high-definition television slice, cellular ultralow latency slice, and two device-to-device slices, one from each cellular slice. In [49], a DRL agent at a single base station is exploited to allocate uplink bandwidth to mobile users from various slices with the aim of maximizing uplink throughput and minimizing energy cost. De Bast *et al.* [50] utilized DQN to dynamically select the best slicing configuration in WiFi networks. DRL slicing in a hierarchical network architecture for dynamic resource reservation is studied in [51], in which the infrastructure provider assigns unutilized resources to network slices maintaining a minimum resource requirement and demand in each slice, where DRL is then employed to efficiently manage reserved resources and maximize QoS.

B. Research Gaps and Proposed Improvements

Despite the growing literature, still there exists a significant research gap: adaptively satisfying the QoS requirements of the URLLC applications while efficiently utilizing the local resources in F-RAN. Motivated by this problem, we provide a novel network slicing technique for sequentially allocating the FNs' limited computing and processing resources at the network edge to various vehicular and smart city applications with heterogeneous latency needs. The proposed technique ensures the efficient utilization of the edge resources in dynamic traffic profiles and task loads. Specifically, this article contributes to resolving the following limitations of the existing works.

1) First, an uncoordinated DRL-based network slicing and a single FN slicing model as in [4], [29], [37], and [46], [47], [49] are not an ideal network slicing approach for 5G and future technologies. Especially in dynamic environments, coordination among FNs is needed for more efficient utilization of edge resources while satisfying the QoS needs of users. In this article, we present a coordinated network slicing model based on multiple FNs cooperating through an edge controller (EC).
2) Second, a centralized DRL-based CC for an entire network to manage resource allocation among various network slices as in [34], [35], and [40] will have latency limitations, especially for URLLC-based IoV, V2X, and smart city applications, such as autonomous driving. Whereas, distributed and independent ECs, which are FNs that serve as cluster heads, as proposed in this article can avoid large transmission delays and satisfy the desired level of QoS at FNs by making local real-time decisions for the received service requests in a cluster.
3) Third, a fixed DRL-based network slicing approach as in [31]–[33] with dedicated fog access point resources for the URLLC slice and dedicated remote radio heads for the MBB slice can cause inefficient utilization of edge resources, especially in a dynamic environment. The nature of many smart city and IoV applications (e.g., autonomous vehicles) requires continuous edge capabilities everywhere in the service area; hence, radio, caching, and computing resources need to be available at the edge. In practice, the population of delay-sensitive and high-data-rate services dynamically varies over time, and as a result, a fixed URLLC or MBB slice will be underutilized during light demand for low-latency or high-speed services, respectively. A more flexible network slicing method as proposed in this article would smartly adapt to the environment. We provide an infinite-horizon Markov decision process (MDP) formulation and a DRL algorithm to adaptively learn the optimal network slicing policy by closely interacting with the IoV and smart city environment.
4) Finally, a hard DRL-based network slicing and hierarchical slicing architecture as in [33], [39], [40], [45], and [51] requires frequent physical shifting of resources, and hence, cannot address dynamic environments with changing demands in a cost-efficient manner. With hard slicing, it will be costly and impractical for cellular operators and service providers to keep adding and transferring further infrastructural assets, i.e., capital expenditure, which includes transceivers (TRX) and other radio resources, computing and signal processing resources, such as BBUs, CPUs, and GPUs, as well as caching resources and storage data centers. Such major network changes could be considered as part of network expansion plans over time. In this work, we propose a cost-efficient, soft (i.e., virtual) network slicing method in F-RAN. We present extensive simulation results to examine the performance and adaptivity of the proposed DRL-based network slicing method in diverse intelligent vehicular systems and smart city environments and different performance objectives.

TABLE I
SUMMARY OF NOTATIONS

Notation	Description
k	cluster size
f_i	fog node (FN) in cluster, $i \in \{1, \ldots, k\}$
$\mathcal{N}_i$	list of neighboring FNs to f_i
$\mathcal{C}$	an edge cluster
$\mathrm{f}_i^\star$	the FN which serves as edge controller
$\hat{\mathrm{f}}$	primary FN which receives the request
$\bar{\mathrm{f}}$	serving FN
u_t	utility of a service request at time t
u_h	threshold for high utility
U	set of utilities
c_t	computing resources for a task at time t
C	set of required computing resources
h_t	holding time for a computing task at time t
H	set of holding times
b_{it}	number of occupied resources for f_i at time t
l_{it}	computing load of f_i at time t
L_t	task load defined as $c_t \times h_t$
N_i	resource capacity of f_i
s_t	state at time t
s'	successor state
$\mathcal{S}$	set of states
a_t	action taken at s_t
a'	successor action at s'
a^*	optimal action
$\mathcal{A}$	set of actions
$\tilde{\mathcal{A}}$	set of possible $serve$ actions
r_t	reward collected for taking a_t at s_t
r_L	reward portion for handling a task load L
R	rewarding system
m_h	number of served high-utility tasks
m_l	number of served low-utility tasks
m	total number of served tasks
M_h	total number of received high-utility tasks
M_l	total number of received low-utility tasks
M	total number of received tasks
ω_g	weight for GoS
ω_u	weight for resource utilization
π	policy for taking actions
π^*	optimal policy
G_t	return from time t onward
$V(s)$	state-value function of s
$V^*(s)$	optimal state-value function
$Q(s,a)$	action-value function
$Q^*(s,a)$	optimal action-value function
w	DNN model weights
$\hat{\mathrm{w}}$	target DNN model weights
$\hat{Q}$	output of target DNN model for an input state s
$\tilde{Q}$	output of DNN model for an input state s
$\mathcal{E}$	IoV and smart city environment
D	capacity of DNN replay memory
γ	reward discount factor
α	learning rate
ϵ	probability of random action
n	batch size
τ	$\hat{\mathrm{w}}$ update interval
ρ	target update rate

The remainder of this article is organized as follows. Section III introduces the network slicing model. The proposed MDP formulation for the network slicing problem is provided in Section IV. Optimal policies and the proposed DRL algorithm are discussed in Section V. Simulation results are presented in Section VI, and the article is concluded in Section VII. A list of notations used throughout this article is provided in Table I.

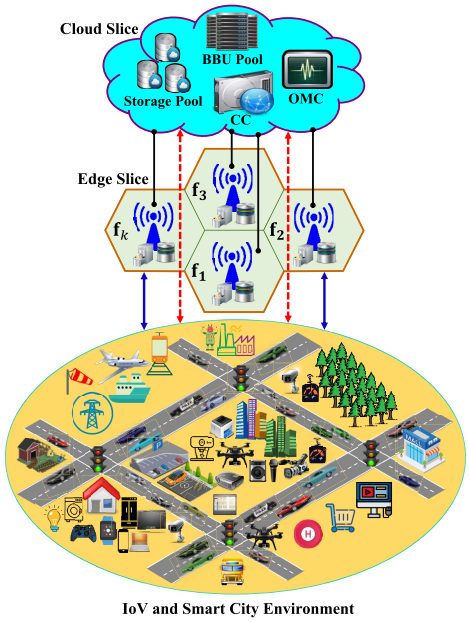

Fig. 1. Network slicing model. Edge slice is connected to cloud slice through high-capacity fronthaul links represented by solid lines. Solid arrows represent edge service to satisfy QoS, and dashed arrows represent task referral to the cloud-slice to save the limited resources of edge slice.

III. NETWORK SLICING MODEL

We consider the F-RAN network slicing model for IoV and smart city shown in Fig. 1. The two logical network slices, cloud slice and edge slice, support multiple radio access technologies and serve heterogeneous latency needs and resource requirements in dynamic IoV and smart city environments. The hexagonal structure represents the coverage area of FNs in the edge slice, where each hexagon exemplifies an FN's footprint, i.e., its serving zone. An FN in an edge cluster is connected through extremely fast and reliable optical links with its adjacent FNs whose hexagons have a common side with it. FNs in the edge slice are also connected via high-capacity fronthaul links to the cloud slice, which includes a powerful CC of massive computing capabilities, a pool of huge storage capacity, centralized BBUs, and an operations and maintenance center (OMC), which monitors the key performance indicators (KPIs) and generates network reports. To ensure the best QoS for the massive smart city and IoV service requests, especially the URLLC applications, and to mitigate the onus on the fronthaul and the cloud, FNs are equipped with computing and processing capabilities to independently deliver network functionalities at the edge of network. However, the edge resources at FNs are limited and therefore, need to be used efficiently.

In an environment densely populated with low-latency service requests, it is rational for the FNs to route delay-tolerant applications to the cloud and save the limited edge resources for delay-sensitive applications. However, in practice, smart city and IoV environments are dynamic, i.e., a typical environment will not be always densely populated with delay-sensitive applications. A rule-based network slicing policy cannot ensure efficient use of edge resources in dynamic

environments as it will underutilize the edge resources when delay-sensitive applications are rare. On the other hand, a statistical learning policy can adapt its decisions to the changing environment characteristics. Moreover, it can learn to prioritize low-load delay-sensitive applications over high-load delay-sensitive ones.

We propose using ECs to efficiently manage the edge resources by enabling cooperation among FNs. In this approach, FNs are grouped in clusters, each of which covers a particular geographical area, and manages the edge resources in that area through a cluster head called EC. The cluster size k is a network design parameter, which represents the number of coordinated FNs in an edge cluster. An FN in each cluster is appointed as EC to manage and coordinate edge resources at FNs in the cluster. The EC is nominated by the network designer mainly based on its central geo location among the FNs in the cluster, like f_1 and f_3 in Fig. 1. Note that unlike the CC, the EC is close to the end users as it is basically one of the FNs in a cluster. Also, the cluster size k is constrained by the neighboring FNs that cover a limited service area, such as a downtown, industrial area, university campus, etc.

All FNs in an edge cluster are connected together and with the EC through super speedy reliable optical links. The EC monitors all individual FN internal states, including resource availability and received service requests, and decides for each service request received by an FN in the cluster. For each received request, the EC chooses one of the three options: 1) serve at the receiving FN (primary FN); 2) serve at a neighboring FN; or 3) serve at the cloud. Each FN in the cluster has a predefined list $\mathcal{N}_i$ of neighboring FNs, which can help serving a received service request. For instance, $\mathcal{C} = \{\mathrm{f}_1, \mathrm{f}_2, \ldots, \overset{\star}{\mathrm{f}}_\mathrm{i}, \ldots, \mathrm{f}_\mathrm{k}\}$ is an edge cluster of size k, where $\overset{\star}{\mathrm{f}}_\mathrm{i}$ denotes the EC which can be any FN in the cluster $\mathcal{C}$. The network designer needs to define a neighboring list $\mathcal{N}_i \subseteq \{\mathcal{C} - \mathrm{f_i}\}$ for each FN in the cluster. An FN can handover service tasks only to its neighbors. Dealing with IoV and smart city service requests, we call the FN that receives a request the primary FN $\hat{\mathrm{f}}$, and call the FN that actually serves the request utilizing its resources the serving FN $\bar{\mathrm{f}}$. Depending on the EC decision, the primary FN or one of its neighbors can be the serving FN, or there can be no serving FN (for the decision to serve at the cloud).

An IoV or smart city application attempts to access the network by sending a service request to the primary FN, which is usually the closest FN to the user. The primary FN checks the utility $u \in U = \{1, 2, \ldots, u_{\max}\}$, i.e., the priority level of executing the service task at the edge, analyzes the task load by figuring the required amount of resources $c \in C = \{1, 2, \ldots, c_{\max}\}$ and holding time of resources $h \in H = \{1, 2, \ldots, h_{\max}\}$, and sends the EC the task input (u_t, c_t, h_t) at time t. Since the service requests are handled sequentially, the changing number of vehicles and smart city applications does not cause any problem in decision making. We consider the resource capacity of the ith FN $\mathrm{f_i} \in \mathcal{C}$ is limited to $\mathrm{N_i}$ resource blocks. Hence, the maximum number of resource blocks to be allocated for a task is constrained by the FN resource capacity, i.e., $c \leq c_{\max} \leq \mathrm{N_i}$. We partition the time into very small time steps $t = 1, 2, \ldots,$ and assume a high-rate sequential arrival of IoV and smart city service requests, one task at a time step. ECs should be intelligent to learn how to decide (which FN to *serve* or *reject*) for each service request, i.e., how to sequentially allocate limited edge resources, to achieve the objective of efficiently utilizing the edge resources while maximizing the Grade of Service (GoS) defined as the proportion of served high-utility requests to the total number of high-utility requests received.

A straightforward approach to deal with this network slicing problem is to filter the received service requests by comparing their utility values with a predefined threshold. For instance, consider ten different utilities $u \in \{1, 2, 3, \ldots, 10\}$ for all received tasks in terms of the latency requirement, where $u = 10$ represents the highest priority and lowest latency task such as the emergency requests from the driver distraction alerting system, and $u = 1$ is for the lowest priority task with highest level of latency such as a service task from smart waste management system. Then, a straightforward non-adaptive solution for network slicing is to dedicate the edge resources to high-utility tasks, such as $u \geq u_h$, and refer to the cloud the tasks with $u < u_h$, where the threshold u_h is a predefined network design parameter. However, such a policy is strictly suboptimum since the EC will execute any task, which satisfies the threshold regardless of how demanding the task load is. For instance, while FNs are busy with serving a few high-utility requests of high load, i.e., low-latency tasks that require large amount of resources c and long holding times h, many high-utility requests with low load demand may be missed. In addition, this straightforward policy increases the burden on the cloud unnecessarily, especially when the environment is dominated by low-utility tasks with $u < u_h$. A better network slicing policy would consider the current resource utilization and expected reward of each possible action while deciding, and also adapt to changing utility and load distributions in the environment. To this end, we next propose an MDP formulation for the considered network slicing problem.

IV. MDP Formulation at EC

MDP formulation enables the EC to consider expected rewards of all possible actions in its network slicing decision. Since closed-form expressions typically do not exist for the expected reward of each possible action at each system state in a real-world problem, RL is commonly used to empirically learn the optimum policy for the MDP formulation. The RL agent (the EC in our problem) learns to maximize the expected reward by trial and error. That means, the RL agent sometimes exploits the best known actions, and sometimes, especially in the beginning of learning, explores other actions to statistically strengthen its knowledge of best actions at different system states. Once, the RL agents learns an optimum policy (i.e., the RL algorithm converges) by managing this exploitation–exploration tradeoff, the learned policy can be exploited as long as the environment (i.e., the probability distribution of system state) remains the same. In dynamic IoV and smart city environments,

an RL agent can adapt its decision policy to the changing distributions.

As illustrated in Fig. 2, for each service request in an edge cluster at time t from an IoV or smart city application with utility u_t, the primary FN computes the number of resource blocks c_t and the holding time h_t, which are required to serve the task locally at the edge. Then, the primary FN shares (u_t, c_t, h_t) with the EC, which keeps track of the available resources at all FNs in the cluster. If neither the primary FN nor its neighbors has c_t available resource blocks for a duration of h_t, the EC inevitably rejects serving the task at the edge and refers it to the cloud. Note that in the proposed cooperative structure enabled by the EC, such an automatic rejection will be much less frequent compared to the noncooperative structure considered in [4], [29], [37], and [46], where each FN decides for its resources on its own. If the requested resource blocks c_t for the requested duration h_t are available at the primary FN or at least one of the neighbors, then the EC uses the RL algorithm given in the next section to decide either to serve or reject. In any case, as a result of the taken action a_t, the EC will observe a reward r_t and the system state s_t will transition to s_{t+1}. We next explain the KPIs in an F-RAN to guide the design of the proposed MDP formulation.

A. Key Performance Indicators

Considering the main motivation behind F-RAN, we define the GoS as a KPI. GoS is the proportion of the number of served high-utility service tasks to the total number of high-utility requests in the cluster, and given by

$$\text{GoS} = \frac{m_h}{M_h} = \frac{\sum_{t=0}^{T-1} \mathbb{1}_{\{u_t \geq u_h\}} \mathbb{1}_{\{a_t \in \{1,2,\ldots,k\}\}}}{\sum_{t=0}^{T-1} \mathbb{1}_{\{u_t \geq u_h\}}} \tag{1}$$

where u_h is a utility threshold, which differentiates the low-latency (i.e., high-utility) tasks such as URLLC from other tasks, $a_t \in \{1, 2, \ldots, k\}$ means *serve* the requested task at the ith FN in the cluster, $\mathrm{f_i} \in \mathcal{C} = \{\mathrm{f_1}, \mathrm{f_2}, \ldots, \mathrm{f_k}\}$, and $\mathbb{1}_{\{\cdot\}}$ is the indicator function taking the value 1 when its argument is true, and 0 otherwise.

Naturally, a network operator would want the edge resources to be efficiently utilized. Hence, the average utilization of edge resources over a time period T gives another KPI

$$\text{Utilization} = \frac{1}{T} \sum_{t=0}^{T-1} \frac{\sum_{i=1}^{k} b_{it}}{\sum_{i=1}^{k} \mathrm{N_i}} \tag{2}$$

where b_{it} and $\mathrm{N_i}$ are the number of occupied resources at time t, and the resource capacity of the FN $\mathrm{f_i}$ in the cluster, respectively. Another KPI to examine the EC performance is cloud avoidance, which is given by the proportion of all IoV and smart city service requests that are served by FNs in the edge cluster to all requests received. Cloud avoidance is reported over a period of time T, and it is given by

$$\text{Cloud Avoidance} = \frac{m}{M} = \frac{\sum_{t=0}^{T-1} \mathbb{1}_{\{a_t \in \{1,2,\ldots,k\}\}}}{M} \tag{3}$$

where $m = m_h + m_l$ is the number of high-utility and low-utility served requests at the edge cluster, and $M = M_h + M_l$ is the total number of high-utility and low-utility received requests. Note that $M - m$ is the portion of IoV and smart city service tasks, which is served by the cloud, and one of the objectives of F-RAN is to lessen this burden especially during busy hours. Cloud avoidance shows a general overview about the contribution of edge slice to share the load. It gives a similar metric as resource utilization, which is more focused on resource occupancy rather than dealing with service requests in general. While we use the resource utilization together with GoS to define an overall performance metric below, cloud avoidance is still used as a performance evaluation metric in Section VI.

To evaluate the performance of an EC over a particular period of time T, we consider the weighted sum of the main two KPIs, the GoS and edge-slice average resource utilization as

$$\text{Performance} = \omega_g \text{GoS} + \omega_u \text{Utilization}. \tag{4}$$

In Section VI, different performance scenarios are considered to evaluate the proposed DRL scheme with respect to other slicing approaches in various vehicular and smart city environments.

B. MDP Formulation

An MDP is defined by the tuple $(\mathcal{S}, \mathcal{A}, P^a, R^a, \gamma)$, where $\mathcal{S}$ is the set of states, i.e., $s_t \in \mathcal{S}$, $\mathcal{A}$ is the set of actions, i.e., $a_t \in \mathcal{A} = \{1, 2, \ldots, k, k+1\}$, $P^a(s, s') = P(s_{t+1} = s' | s_t = s, a_t = a)$ is the transition probability from state s to s' when action a is taken, $R^a(s, s')$ is the reward received by taking action a in state s, which ends up in state s', i.e., $r_t \in R^a(s, s')$, and $\gamma \in [0, 1]$ is the discount factor in computing the return, which is the cumulative reward

$$G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots = \sum_{j=0}^{\infty} \gamma^j r_{t+j}. \tag{5}$$

γ represents how much weight is given to the future rewards compared to the immediate reward. For $\gamma = 1$, future rewards are of equal importance as the immediate reward, whereas $\gamma = 0$ completely ignores future rewards. The objective in MDP is to maximize the expected cumulative reward starting from $t = 0$, i.e., $\max_{\{a_t\}} \mathbb{E}[G_0 | s_0]$, where G_t is given by (5), by choosing the actions $\{a_t\}$.

Before explaining the (state, action, reward) structure in the proposed MDP, let us first define the task load $L_t = c_t \times h_t$ as the number of resource blocks required to execute a task completely, and similarly, the existing load l_{it} of FN i as the number of already allocated resource blocks (see Fig. 2).

1) *State:* We define the system state in a cluster of size k at any time t as

$$s_t = \left(b_{1t}, l_{1t}, b_{2t}, l_{2t}, \ldots, b_{kt}, l_{kt}, \hat{\mathrm{f}}_t, u_t, c_t, h_t\right) \tag{6}$$

where b_{it} denotes the number of resource blocks in use at FN i at time t. Note that $b_{i(t+1)}$, $l_{i(t+1)}$, and, in turn, the next state s_{t+1} are independent of the past values given the current state s_t, satisfying the Markov property $P(s_{t+1} | s_0, s_1, s_2, \ldots, s_t, a_t) = P(s_{t+1} | s_t, a_t)$.

2) *Action:* The EC decides, as shown in Fig. 2, for each service request by taking an action $a_t \in \mathcal{A} = \{1, 2, \ldots, k, k+1\}$, where $a_t = i \in \{1, 2, \ldots, k\}$ means

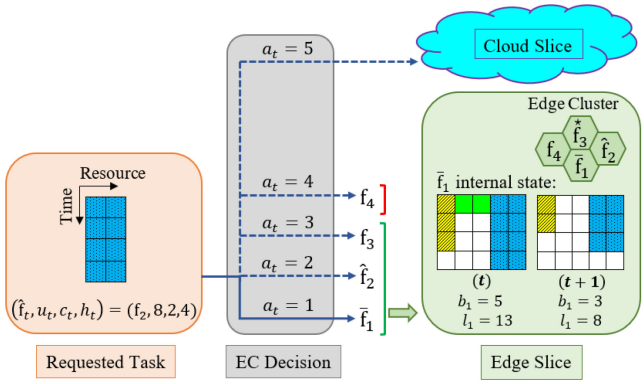

Fig. 2. EC decision for a sample service request received by f_2, and the internal state of the serving FN f_1 with $N_1 = 5$ and $h_{\max} = 4$ [see (6)]. The edge cluster size is $k = 4$ and f_3 is the EC.

serve the requested task at the ith FN in the cluster, $f_i \in \mathcal{C} = \{f_1, f_2, \ldots, f_k\}$, whereas $a_t = k + 1$ means to *reject* the job and refer it to the cloud. Note that for a request received by f_i, the feasible action set is a subset of $\mathcal{A}$ consisting of f_i, its neighbors $\mathcal{N}_i$, and the cloud. Fig. 2 illustrates the decision of the EC for a sample service request received by f_2 at time t in an edge cluster with $k = 4$ FNs. Note that in this example, the action $a_t = 4$ is not feasible as $f_4 \notin \mathcal{N}_2$, and the EC took the action $a_t = 1$, which means serve the task by f_1. Hence, f_1 started executing the task at t while another two tasks (striped yellow and green) are in progress. At $t+1$, two resource blocks are released as the job in clear-green is completed. Note that resource utilization of f_1 decreased from 100% at t, i.e., internal busy state with $b_{1t} = 5$, to 60% at $t + 1$.

3) *Reward:* In general, a proper rewarding system is crucial for an RL agent to learn the optimum policy of actions that maximizes the KPIs. The RL agent at the EC collects an immediate reward $r_t \in R^a(s, s')$ for taking action a at time t from state s, which ends in the successor state s' in the next time step $t + 1$. We define the immediate reward

$$r_t = r_{(a_t,u_t)} \pm r_{L_t} \tag{7}$$

using two components. The first term $r_{(a_t,u_t)} \in \{r_{sh}, r_{sl}, r_{rh}, r_{rl}, r_{bh}, r_{bl}\}$ corresponds to the reward portion for taking an action $a \in \{1, 2, \ldots, k, k+1\}$ when a request of specific u is received, and the second term

$$r_{L_t} = c_{\max} \times h_{\max} + 1 - L_t \tag{8}$$

considers the reward portion for handling the new job load $L_t = c_t \times h_t$ of a requested task. For instance, serving low-load task such as $L = 3$ is awarded more than serving a task with $L = 18$. Similarly, rejecting a low-load task such as $L = 3$ should be more penalized, i.e., negatively rewarded especially when $u \geq u_h$, than rejecting a task with the same utility and higher load such as $L = 18$. The two reward parts are added when $a_t =$ serve, and subtracted if $a_t =$ reject. We define six different reward component $r_{(a,u)} \in \{r_{sh}, r_{sl}, r_{rh}, r_{rl}, r_{bh}, r_{bl}\}$, where r_{sh} is the reward for serving a high-utility request, r_{sl} is the reward for serving a low-utility request, r_{rh} is the reward for rejecting a high-utility request, r_{rl} is the reward for rejecting a low-utility request, r_{bh} is the reward for rejecting a high-utility request due to being busy, and r_{bl} is the reward for rejecting a low-utility request due to being busy. Note that having a separate reward for rejecting due to a busy state makes it easier for the RL agent to differentiate between similar states for the *reject* action. A request is determined as high utility or low utility based on the threshold u_h, which is a design parameter that depends on the level of latency requirement in an IoV and smart city environment.

V. Optimal Policies and DQN

The state value function $V(s)$ represents the long-term value of being in a state s. That is, starting from state s how much value on average the EC will collect in the future, i.e., the expected total discounted rewards from that state onward. Similarly, the action-value function $Q(s, a)$ tells how valuable it is to take a particular action a from the state s. It represents the expected total reward which the EC may get after taking the particular action a from the state s onward. The state-value and the action-value functions are given by the Bellman expectation equations [52]

$$V(s) = \mathbb{E}[G_t|s] = \mathbb{E}\big[r_t + \gamma V(s')|s\big] \tag{9}$$

$$Q(s, a) = \mathbb{E}[G_t|s, a] = \mathbb{E}\big[r_t + \gamma Q(s', a')|s, a\big] \tag{10}$$

where the state value $V(s)$ and the action value $Q(s, a)$ are recursively presented in terms of the immediate reward r_t and the discounted value of the successor state $V(s')$ and the successor state–action $Q(s', a')$, respectively. a' denotes the action at the next state s'.

Starting at the initial state s_0, the EC objective can be achieved by maximizing the expected total return $V(s_0) = \mathbb{E}[G_0|s_0]$ over a particular time period T. To achieve this goal, the EC should learn an optimal decision policy to take proper actions. However, considering the large dimension of sate space [see (6)] and the intractable number of state–action combinations, it is infeasible for RL tabular methods to keep track of all state–action pairs and continuously update the corresponding $V(s)$ and $Q(s, a)$ for all combinations in order to learn the optimal policy. Approximate DRL methods such as DQN is a more efficient alternative for the high-dimensional EC MDP to quickly learn an optimal decision policy to take proper actions, which we discuss next.

A policy π is a way of selecting actions. It can be viewed as a mapping from states to actions as it describes the set of probabilities for all possible actions to select from a given state, i.e., $\pi = \{P(a|s)\}$. A policy helps in estimating the value functions in (9) and (10). π_1 is said to be better than another policy π_2 if the state value function following π_1 is greater than that following π_2 for all states, i.e., $\pi_1 > \pi_2$ if $V_{\pi_1}(s) > V_{\pi_2}(s)$ $\forall s \in \mathcal{S}$. A policy π is said to be optimal if it maximizes the value of all states, i.e., $\pi^* = \arg\max_\pi V_\pi(s)$ $\forall s \in \mathcal{S}$. Hence, to solve the considered MDP problem, the DRL agent needs to find the

optimal policy through finding the optimal state-value function $V^*(s) = \max_\pi V_\pi(s)$, which is similar to finding the optimal action-value function $Q^*(s,a) = \max_\pi Q_\pi(s,a)$ for all state–action pairs. From (9) and (10), we can write the Bellman optimality equations for $V^*(s)$ and $Q^*(s,a)$ as

$$V^*(s) = \max_{a\in\mathcal{A}} Q^*(s,a) = \max_{a\in\mathcal{A}} \mathbb{E}\big[r_t + \gamma V^*(s')|s,a\big] \quad (11)$$

$$Q^*(s,a) = \mathbb{E}\left[r_t + \gamma \max_{a'\in\mathcal{A}} Q^*(s',a')|s,a\right]. \quad (12)$$

The expression of optimal state-value function $V^*(s)$ greatly simplifies the search for optimal policy as it subdivides the targeted optimal policy into local actions: take an optimal action a^* from state s, which maximizes the expected immediate reward followed by the optimal policy from successor state s'. Hence, the optimal policy is simply taking the best local actions from each state considering the expected rewards. Dealing with $Q^*(s,a)$ to choose optimal actions is even easier, because with $Q^*(s,a)$ there is no need for the EC to do the one-step-ahead search and instead it picks the best action that maximizes $Q^*(s,a)$ at each state. The optimal action for each state s is given by

$$a^* = \arg\max_{a\in\mathcal{A}} Q^*(s,a) = \arg\max_{a\in\mathcal{A}} \mathbb{E}\big[r_t + \gamma V^*(s')|s,a\big]. \quad (13)$$

The optimal policy can be learned by solving the Bellman optimality (11) and (12) for a^*. This can be done for tractable number of states by estimating the optimal value functions using tabular solution methods such as dynamic programming, and model-free RL methods, which include Monte Carlo, SARSA, expected SARSA, and QL [4]. However, for high-dimensional state space, such as ours given in (6), tabular methods are not tractable in terms of computational and storage complexity. DRL methods address the high-dimensionality problem by approximating the value functions using DNNs.

DQN is a powerful DRL method for addressing RL problems with high-dimensional input states and output actions [38]. DQN extends QL to high-dimensional problems by using DNN to approximate the action-value functions without keeping a Q-table to store and update the Q-values for all possible state–action pairs as in QL. Fig. 3 demonstrates the DQN method for EC in the network slicing problem, in which the DQN agent at EC learns about the IoV and smart city environment by interaction. The DQN agent is basically a DNN that consists of an input layer, hidden layers, and an output layer. The number of neurons in the input and output layers is equal to the state and action dimensions, respectively, whereas the number of hidden layers and the number of neurons in each hidden layer are design parameters to be chosen. Feeding the current EC state s to the DNN as an input and regularly updating its parameters, i.e., the weights of all connections between neurons, DNN is able to predict the Q-values at the output for a given input state. The DRL agent at EC sometimes takes random actions to explore new rewards, and at other times, exploits its experience to maximize the discounted cumulative rewards over time and keeps updating the DNN weights. Once the DNN weights converge to the optimal values, the agent learns the optimal policy for taking actions in the observed environment.

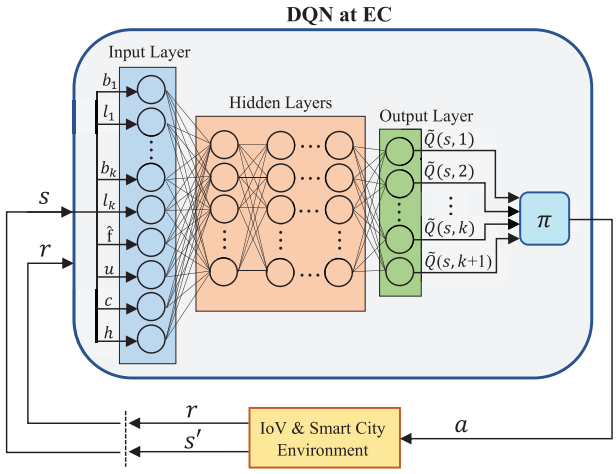

Fig. 3. Interaction of the DQN-based EC with the IoV and smart city environment. Given the EC input state $s = (b_1, l_1, \ldots, b_k, l_k, \hat{\mathrm{f}}, u, c, h)$, the DQN agent predicts the action-value functions and follows a policy π to take an action a, which ends up in state s', and collects a reward r accordingly.

For a received service request, if the requested resources are affordable, i.e., $c_t \le (\mathrm{N_i} - \mathrm{b_{it}})$ for any $\mathrm{f}_i \in \{\hat{\mathrm{f}}_i, \mathcal{N}_i\}$, the EC makes a decision whether to *serve* the request by the primary FN or one of its neighbors, or *reject* and refer it to the cloud. From (13), the optimal action at state s is given by

$$a^* = \begin{cases} i \in \tilde{\mathcal{A}}, & \text{if } Q^*(s,i) = \max\limits_{a\in\tilde{\mathcal{A}}} Q(s,a) \\ k+1, & \text{otherwise} \end{cases} \quad (14)$$

where $\tilde{\mathcal{A}}$ denotes the set of possible *serve* actions to execute the service task by $rf_i \in \{\hat{\mathrm{f}}_i, \mathcal{N}_i\}$. The procedure to learn the optimal policy from the IoV and smart city environment using the model-free DQN algorithm is given in Algorithm 1.

Algorithm 1 shows how the EC learns the optimal policy π^* for the considered MDP. It requires the EC design parameters k, $\mathcal{N}$, $\mathrm{N_i}$, and u_h, and selecting the DNN hyperparameters γ, the target update rate ρ, the probability ϵ of making a random action for exploration, the replay memory capacity D to store the observations (s,a,r,s'), the minibatch size n of samples used to train the DNN model and update its weights w, and the data of the IoV and smart city users u, c, and h. Note that u, c, and h can be real data from the IoV and smart city environment, as well as from simulations if the probability distributions are known. The DNN target model at line 2 is used to stabilize the DNN model and avoid divergence by reducing the correlation between the action-values $Q(s,a)$ and the targets $y = r + \gamma \max_{a'} Q(s',a')$ through only periodical updates of the target model weights $\hat{w}$. In each iteration, we take an action and observe the collected reward and the successor state. Actions are taken according to a policy π such as the ϵ-greedy policy in which a random action with probability ϵ is taken to explore new rewards, and an optimal action [see (14)] is taken with probability $(1-\epsilon)$ to maximize the rewards. Model is trained using experience replay as shown in lines 9–14. At line 9, a minibatch of n random observations is sampled from $\mathbb{M}$. The randomness in selecting samples eliminates correlations in the observations to avoid model overfitting. At line 11, we estimate the output

Algorithm 1 Learning Optimum Policy Using DQN

1: Select: $\{\gamma, \epsilon\} \in [0, 1]$, $\rho \in (0, 1]$, $n \in \{1, 2, \ldots, D\}$;
2: Create DNN model and target model with weights w and $\hat{w}$, respectively;
3: Initialize: w, $\hat{w}$, the replay memory $\mathbb{M}$ with size D;
4: Initialize: s;
5: **for** $t = 0, 1, 2, \ldots, T$ **do**
6: Take action a_t according to $\pi = \epsilon$-greedy, and observe r_t and s_{t+1};
7: Append the observation (s_t, a_t, r_t, s_{t+1}) to $\mathbb{M}$;
8: $s \leftarrow s_{t+1}$;
9: Sample a random minibatch of n observations from $\mathbb{M}$;
10: **for** $j = 1, 2, \ldots, n$ **do**
11: Predict $\hat{Q}_j(s_j|\hat{w})$;
12: $y_j = \begin{cases} r_j \ \text{ if } t+1 = T, \\ r_j + \gamma \max\limits_{a'} \hat{Q}_j(s', a'|\hat{w}) \ \text{ otherwise.} \end{cases}$
13: Fit the DNN model for (s_j, y_j) by applying gradient descent step on $\left(y_j - \tilde{Q}_j\right)^2$ with respect to w;
14: **end for**
15: **if** $(t \bmod \tau) = 0$ **then**
16: $\hat{w} \leftarrow \rho w + (1 - \rho)\hat{w}$;
17: **end if**
18: **if** w converges **then**
19: $w^* \leftarrow w$;
20: **break**
21: **end if**
22: **end for**
23: Use w^* to estimate $Q^*(s, a)$ required for π^* using (14).

vector $\hat{Q}$ of the target model for a given input state s in each experience sample using the target model weights $\hat{w}$. $\tilde{Q}$ and $\hat{Q}$ are the predicted vectors of the $k+1$ Q-values for a given state s with respect to w and $\hat{w}$, respectively. The way to compute the target for sample j is shown at line 12. At line 13, we update the model weights w by fitting the model for the input states and the corresponding targets. A gradient decent step is applied to minimize the squared loss $(y_j - \tilde{Q}_j)^2$ between the target and the model predictions. The gradient descent converges to the global minima of the quadratic loss function when the DNN is overparameterized, i.e., with large training data and large number of hidden neurons [53]–[55]. The target model weights are periodically updated every τ time steps as shown at line 16, where the update rate ρ exemplifies how much we believe in our experience. The algorithm stops when the DNN model weights w converge. The converged values are then used to determine optimal actions, i.e., π^* as in (14).

VI. Simulations

We next provide simulation results to evaluate the performance of the proposed network slicing approach in dynamic IoV and smart city environments under different performance evaluation criteria. We compare the DRL algorithm given in Algorithm 1 with the serve-all-utilities (SAU) algorithm in which the EC serves all coming tasks when requested resources are available, serve-high-utilities (SHU) algorithm where the EC filters high-utility requests and serve them if the available resources are enough, and the QL algorithm independently running at each FN following a local version of our MDP formulation [4]. The QL algorithm at each FN corresponds to the noncooperative scenario; hence, this comparison will help evaluate the importance of cooperation among FNs. In the noncooperative scenario, each FN operates as a standalone entity with no neighbors to handover tasks when busy, and no EC to manage the edge resources.

TABLE II
Utility Distributions Corresponding to a Variety of Latency Requirements of IoV and Smart City Applications in Various Environments

	$\mathcal{E}_1$	$\mathcal{E}_2$	$\mathcal{E}_3$	$\mathcal{E}_4$	$\mathcal{E}_5$
$P(u=1)$	0.015	0.012	0.008	0.004	0.001
$P(u=2)$	0.073	0.058	0.038	0.019	0.004
$P(u=3)$	0.365	0.288	0.192	0.096	0.019
$P(u=4)$	0.292	0.230	0.154	0.077	0.015
$P(u=5)$	0.205	0.162	0.108	0.054	0.011
$P(u=6)$	0.014	0.071	0.142	0.214	0.271
$P(u=7)$	0.013	0.064	0.129	0.193	0.244
$P(u=8)$	0.011	0.057	0.114	0.171	0.217
$P(u=9)$	0.009	0.043	0.086	0.129	0.163
$P(u=10)$	0.003	0.015	0.029	0.043	0.055
$P(u \geq u_h = 8)$	2.3%	11.5%	22.9%	34.3%	43.5%
$\bar{u}$	3.82	4.589	5.55	6.5	7.27

A. Simulation Environments

We evaluate the performances in various IoV and smart city environments with different compositions of user utilities. Specifically, we consider ten utility classes that represent different latency requirements to exemplify the variety of IoV and smart city applications in an F-RAN setting. By changing the distribution of utility classes, we generate five IoV and smart city environments as summarized in Table II. Higher density of high-utility applications makes the IoV and smart city environment richer in terms of URLLC applications. Denoting an IoV and smart city environment of a particular utility distribution with $\mathcal{E}$, we show in Table II the statistics of $\mathcal{E}_1$, $\mathcal{E}_2$, $\mathcal{E}_3$, $\mathcal{E}_4$, and $\mathcal{E}_5$. The probabilities in the first ten rows in Table II present detailed information about the proportion of each utility class in the environment corresponding to the latency requirement of diverse IoV and smart city applications. The last two rows interpret the quality or richness of IoV and smart city environments, where $\bar{u}$ is the mean of utilities in an environment, and $P(u \geq u_h)$ is the percentage of high-utility population. We started with a general environment given by $\mathcal{E}_3$ for the following IoV and smart city applications corresponding to the utility values $1, 2, \ldots, 10$, respectively: smart lighting and automation of public buildings, air quality management and noise monitoring, smart waste management and energy consumption management, smart parking assistance, in-vehicle audio and video infotainment, driver authentication service, structural health monitoring, safe share rides, smart amber alerting system and AI-driven and video-analytics tracking services, and driver distraction alerting system and autonomous driving. Then, we changed the utility distribution to obtain the other environments.

TABLE III
SIMULATION SETUP

Parameter	Description	Value
N_i	resource capacity of FN f_i	7
C	set of possible resource blocks	$\{1, 2, 3, 4\}$
H	set of possible holding times	$5\times\{1, 2, 3, 4, 5, 6\}$
ω_g	weight for GoS	$\{0.7, 0.5, 0.3\}$
ω_u	weight for resource utilization	$\{0.3, 0.5, 0.7\}$
u_h	threshold for a "high-utility"	8
D	capacity of DNN replay memory	2000
γ	reward discount factor	0.9
α	learning rate	0.01
ϵ	probability of random action	1.0 with 0.9995 decay
n	batch size	32
τ	$\hat{w}$ update interval	1000
ρ	$\hat{w}$ update rate	0.2

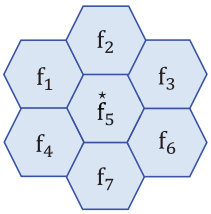

Fig. 4. Structure of the edge cluster considered in the simulations. The neighboring lists $\mathcal{N}_1, \mathcal{N}_2, \ldots, \mathcal{N}_7$ include the adjacent FNs only.

TABLE IV
CONSIDERED REWARDING SYSTEMS

Scenario	ω_g	ω_u	R	$\{r_{sh},\ r_{rh},\ r_{bh},\ r_{sl},\ r_{rl},\ r_{bl}\}$	r_L
1	0.7	0.3	R_1	$\{24, -12, -12,\ -3,\ 3,\ 12\}$	(see (8))
2	0.5	0.5	R_2	$\{24, -12, -12,\ 0,\ 0,\ 12\}$	(see (8))
3	0.3	0.7	R_3	$\{50, -50, -50,\ 50, -50, -25\}$	0

B. Simulation Parameters

The simulation parameters used in this section are summarized in Table III. We consider an edge cluster of size $k = 7$, where each FN has a computing and processing resource capacity of seven resource blocks, i.e., $N = 7$. The central FN f_5 acts as the EC, and the neighboring relationships are shown in Fig. 4. In a particular IoV and smart city environment $\mathcal{E}$, the threshold that defines "high utility" is set to $u_h = 8$, i.e., $u \in \{8, 9, 10\}$ is a high-utility application with higher priority for edge service. To make the resource allocation of the network slicing problem more challenging, we consider a request arrival rate of at least five times the task execution rate, i.e., holding times increment by five times the arrival interval. The probabilities of $c \in C = \{1, 2, 3, 4\}$ are 0.1, 0.2, 0.3, and 0.4, respectively, whereas the probabilities of $h \in H = \{5, 10, 15, 20, 25, 30\}$ are 0.05, 0.1, 0.1, 0.15, 0.2, and 0.4, respectively.

We consider a fully connected DNN structure for DQN with an input layer of 18 neurons, 2 hidden layers of 64 and 24 neurons, respectively, and an 8-neuron output layer. A linear activation function is used at the output layer, and ReLU activation is considered for the other layers. The Huber loss function and the RMSprop optimizer are considered with 0.01 learning rate, 10^{-4} learning decay, and momentum of 0.9. The ϵ-greedy policy is adopted in DNN training where ϵ starts at 1.0 for 10% of the time in training and then decays at a rate of 0.9995 to a minimum value of 10^{-3} to guarantee enough exploration over time. As it depends on the nature of the problem, there is no rule of thumb to tune DNNs. However, the key factors and main DNN hypeparameters to optimize for quick convergence include the loss function, the optimizer, the interval τ to update target weights, the update rate ρ, the exploration rate ϵ, the learning rate α, the discount factor γ, the randomness of the samples and the batch size n, replay memory size D, and a proper rewarding system to expedite the learning. The values of all hyperparameters in this section are chosen based on extensive experiments.

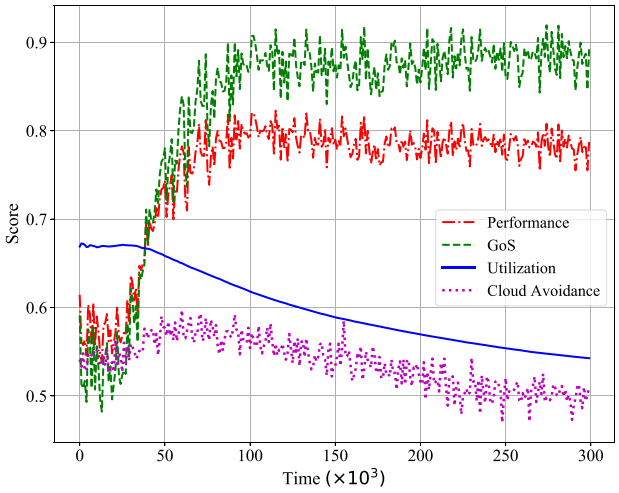

Fig. 5. Performance and main network KPIs for DQN-based EC while learning the optimum policy in the IoV and smart city environment $\mathcal{E}_3$ under scenario 1 of Table IV.

We examine the KPIs explained in Section IV-A, GoS, resource utilization, cloud avoidance, as well as the overall performance [see (3) and (4)] considering the three scenarios shown in Table IV with the weights $\omega_g = 1 - \omega_u = 0.7$, $\omega_g = \omega_u = 0.5$, and $\omega_g = 1 - \omega_u = 0.3$. Each scenario in Table IV represents a new problem, hence the rewarding systems R_1, R_2, and R_3 are chosen to facilitate learning the optimal policy in each scenario. The two reward components, $r_{(a,u)} \in \{r_{sh}, r_{rh}, r_{bh}, r_{sl}, r_{rl}, r_{bl}\}$ and r_L for each rewarding system are provided in Table IV. Note that unlike R_2 and R_3, R_1 encourages rejecting low-utility requests with higher loads to accommodate the performance requirement of scenario 1, which puts higher weight on GoS with $\omega_g = 0.7$. On the other hand, R_3 promotes serving regardless of the request utility and the task load as the performance in scenario 3 is in favor of achieving higher resource utilization with $\omega_u = 0.7$.

C. Simulation Results

We train the DRL agent at the EC in various environments and under different performance scenarios provided in Tables II and IV, respectively. By interaction with the environment as illustrated in Fig. 3, the EC learns the optimal policy using the DQN method given in Algorithm 1. Considering the environment $\mathcal{E}_3$ and the performance scenario 1, Fig. 5 shows an example for the learning curve of the proposed DQN-based EC in terms of the overall performance and KPIs,

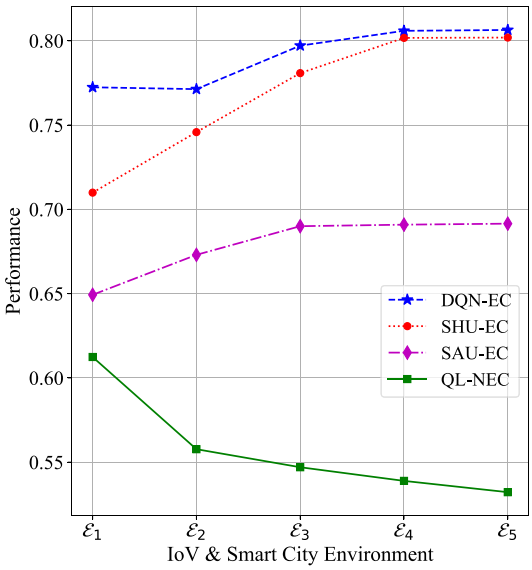

Fig. 6. Performance of the edge slice when the EC applies the DRL Algorithm 1, SAU and SHU for the coordinated FNs, and the uncoordinated QL-based FNs case with NEC. Considering scenario 1 in Table IV with $\omega_g = 1 - \omega_u = 0.7$.

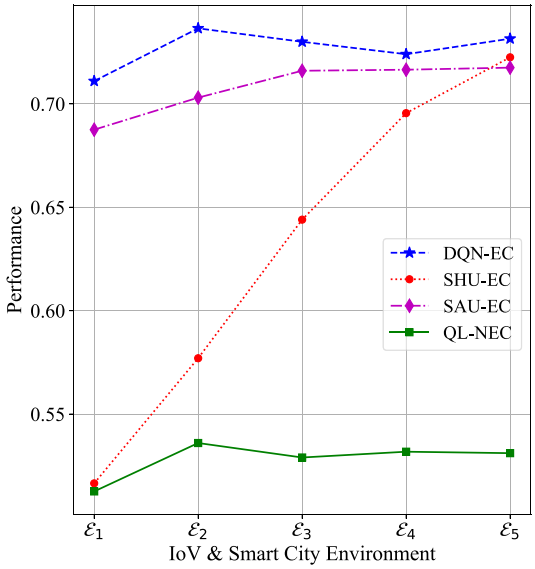

Fig. 7. Performance of the edge slice when the EC applies the DRL Algorithm 1, SAU and SHU for the coordinated FNs, and the uncoordinated QL-based FNs case with NEC. Considering scenario 2 in Table IV with $\omega_g = 1 - \omega_u = 0.5$.

which quickly converge to the optimal scores. Starting with exploration through taking random actions for 30k time steps, the EC initially performs improperly and provides a relatively low GoS (i.e., many high-utility requests are missed) while utilizing the resources mainly for low-utility requests. However, as the algorithm learns the optimum actions from reward feedback, the exploration rate decays and the performance starts to improve. As a result, the EC quickly aligns with the objectives of scenario 1 putting more emphasis on GoS by prioritizing high-utility users for edge service.

Next, we compare DQN-EC given in Algorithm 1 with SAU-EC, SHU-EC, and QL with no EC (QL-NEC) under the three scenarios given in Table IV. Figs. 6–8 show that the DRL-based EC adapts to each scenario and outperforms the other algorithms in all IoV and smart city environments. For scenario 1 in Fig. 6, SHU-EC has a comparable performance to DQN-EC because SHU algorithm promotes serving high-utility requests all the time, which matches with the focus on GoS in scenario 1 design objective with $\omega_g = 0.7$. However, in poor IoV and smart city environments with less high-utility population such as $\mathcal{E}_1$, the performance gap increases. This gap shrinks as environment becomes richer and SHU-EC achieves a performance as high as the DQN-EC score in $\mathcal{E}_4$ and $\mathcal{E}_5$. The performance of SAU-EC slightly increases while moving from $\mathcal{E}_1$ to $\mathcal{E}_3$ and becomes stable afterward even for the richer environments $\mathcal{E}_4$ and $\mathcal{E}_5$ since SAU-EC does not prioritize high-utility tasks. Unlike the other algorithms, QL-NEC shows a declining trend since the network slicing problem becomes more challenging with uncoordinated FNs while moving toward richer environments in this scenario. Fig. 7 represents scenario 2 with equal weights for GoS and resource utilization, where SAU-EC is the second performing algorithm following DQN-EC. With less importance for GoS, the performance of SHU-EC is as low as the QL-NEC in $\mathcal{E}_1$ and although it grows while moving to richer environments, it does not reach a comparable level until $\mathcal{E}_4$ and $\mathcal{E}_5$. The uncoordinated FNs with QL-NEC is more steady in scenario 2. Fig. 8 shows the performances in scenario 3 in which more emphasis is put on resource utilization than GoS with $\omega_u = 0.7$. It is observed that SHU-EC fails to achieve a comparable level of performance compared to DQN-EC while SAU-EC does.

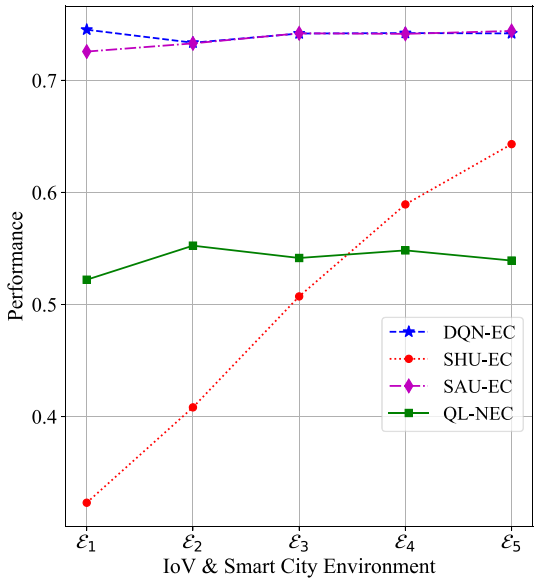

Fig. 8. Performance of the edge slice when the EC applies the DRL Algorithm 1, SAU and SHU for the coordinated FNs, and the uncoordinated QL-based FNs case with NEC. Considering scenario 3 in Table IV with $\omega_g = 1 - \omega_u = 0.3$.

Fig. 9 provides the detailed KPI scores for GoS, resource utilization, and cloud avoidance for all algorithms considering the three design scenarios in all environments. DQN-EC always adapts to the design objective and the IoV and smart city environment. It maximizes GoS in scenario 1 as shown in Fig. 9(a), balances GoS and utilization for scenario 2 as observed in Fig. 9(b), and promotes resource utilization for

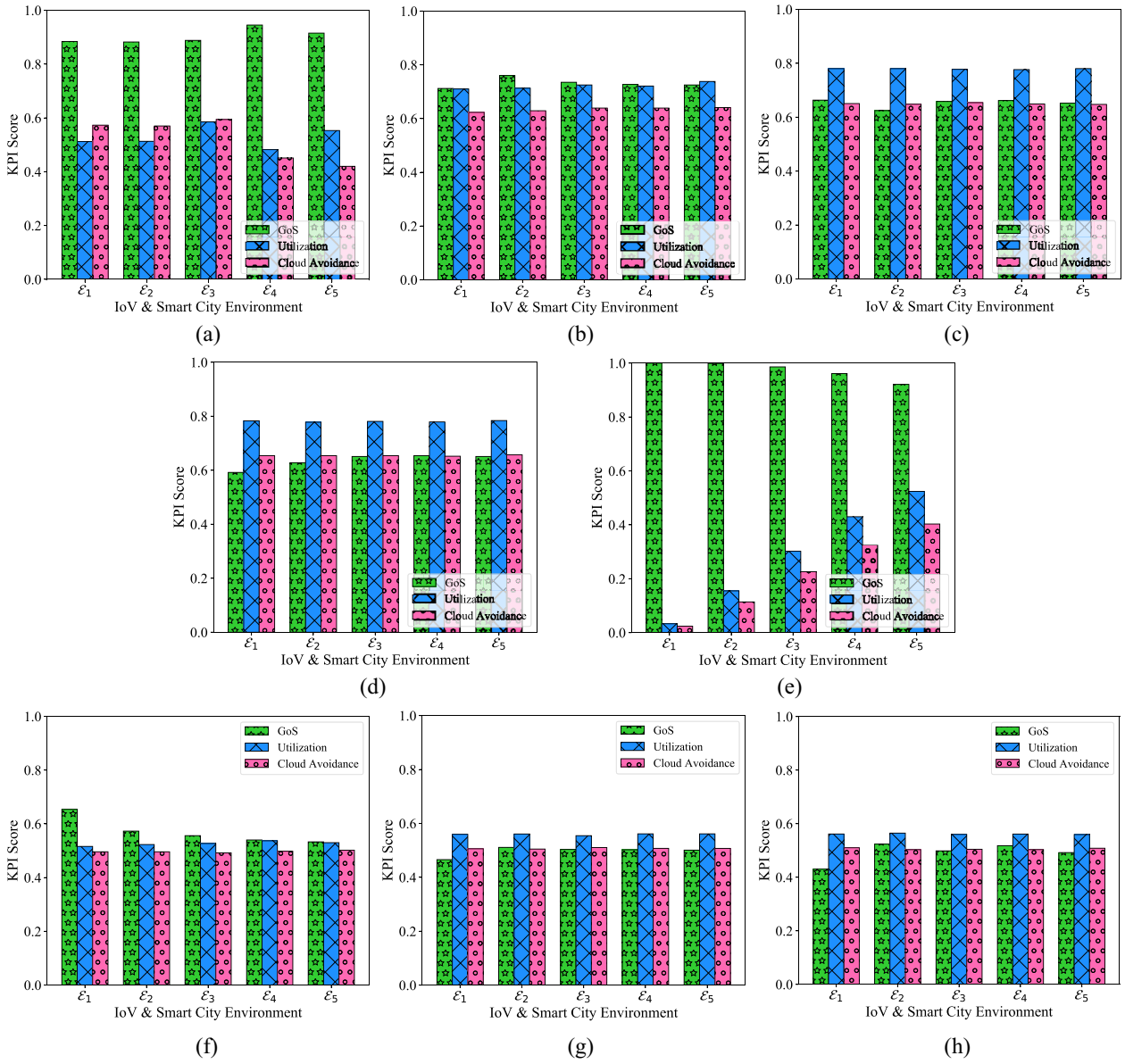

Fig. 9. Score of the main three individual KPIs, GoS, resource utilization, and cloud avoidance when the EC applies Algorithm 1, SAU and SHU for coordinated FNs, and the uncoordinated QL at FNs with no EC, under the three scenarios in Table IV. (a) DQN-EC, $\omega_g = 0.7$ and $\omega_u = 0.3$. (b) DQN-EC, $\omega_g = \omega_u = 0.5$. (c) DQN-EC, $\omega_g = 0.3$ and $\omega_u = 0.7$. (d) SAU-EC, all scenarios. (e) SHU-EC, all scenarios. (f) QL-NEC, $\omega_g = 0.7$ and $\omega_u = 0.3$. (g) QL-NEC, $\omega_g = \omega_u = 0.5$. (h) QL-NEC, $\omega_g = 0.3$ and $\omega_u = 0.7$.

scenario 3 as shown in Fig. 9(c). QL-NEC in Figs. 9(f)–(h) tries to behave similarly as it learns by interaction, but unfortunately the uncoordinated FNs in the edge slice cannot achieve that. Note that DQN-EC learns the right balance between GoS and resource utilization in each scenario. For instance, even though SHU-EC is the second performing in Fig. 6 following DQN-EC, it has lower utilization and cloud avoidance scores, i.e., less edge-slice contribution to handle service requests as shown in Fig. 9(e). Similarly, SAU-EC is well performing in scenario 2 compared to DQN-EC as shown in Fig. 7; however, it does not learn to balance GoS and utilization as DQN-EC does in Fig. 9(b).

Finally, we test the performance of the proposed DQN algorithm in a dynamic IoV and smart city environment. In Fig. 10, we consider the design objectives of scenario 1 in Table IV and a sampling rate of 5×10^{-4}. To generate a dynamic IoT environment we start with 40 samples for the initial environment and then change $\mathcal{E}$ every 30 samples. More samples are considered for the initial $\mathcal{E}$ since we start with vacant resource blocks for all FNs in the edge slice. We consider a dynamic IoV and smart city environment whose composition of high-utility requests, i.e., low-latency tasks, changes over a day. Starting in the morning busy hours with $\mathcal{E}_4$, the density of high-utility requests drops over time to $\mathcal{E}_1$ during the late morning hours. It starts growing to reach $\mathcal{E}_2$ by noon and $\mathcal{E}_3$ in the evening, and then peaks again during the night busy hours with $\mathcal{E}_5$. These five environments represent different distributions over diverse levels of utilities, i.e., different latency requirements of the various IoV and smart city applications. Hence, they can be thought as different traffic profiles during

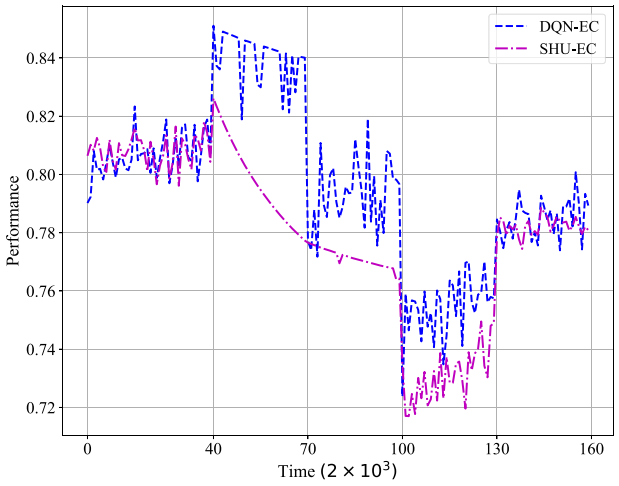

Fig. 10. Performance of the proposed DQN and the straightforward SHU policy for network slicing in a dynamic IoV and smart city environment considering the design objective of scenario 1 in Table IV. Although SHU performs well in rich environments, it cannot adapt to the other environments as expected. The proposed DQN policy on the other hand learns to adapt to different environments.

the day in terms of the required QoS. The changes during the day directly affect the overall distribution of the environment over time and make it dynamic. In the proposed algorithm, once the EC detects the traffic profile, i.e., the environment, it applies the corresponding optimal policy π_4^* given in (14) to maximize the expected rewards in $\mathcal{E}_4$. Right after the density of low-latency tasks drops over time to $\mathcal{E}_1$, i.e., at $t = 80$K, the EC keeps following π_4^* until it detects the change from the statistics of task utilities, which results in a slight degradation in its performance since π_4^* is no longer optimal for the new environment $\mathcal{E}_1$. However, after a short learning period, the EC adapts to the dynamics, and switches to the new optimal policy π_1^*. Similarly, as seen for the other transitions from $\mathcal{E}_1$ to $\mathcal{E}_2$, $\mathcal{E}_2$ to $\mathcal{E}_3$, and $\mathcal{E}_3$ to $\mathcal{E}_5$, DQN-EC successfully adapts to the changing IoV and smart city environments. Whereas, the straightforward SHU-EC policy performs well in only the rich environments for which it was designed, and cannot adapt to the changes in the environment as expected.

VII. Conclusion

We developed an infinite-horizon MDP formulation for the network slicing problem in a realistic fog-RAN with cooperative FNs; and proposed a DRL solution for the ECs, which are the FNs that serve as cluster heads, to learn the optimal policy of allocating the limited edge computing and processing resources to vehicular and smart city applications with heterogeneous latency needs and various task loads. The DQN-based EC quickly learns the dynamics through interaction with the environment and adapts to it. DQN-EC dominates the straightforward and noncooperative RL approaches as it always learns the right balance between GoS and resource utilization under different performance objectives and environments. In a dynamic environment with changing distributions, DQN-EC adapts to the dynamics and updates its optimal policy to maximize the performance.

References

[1] A. T. Nassar, A. I. Sulyman, and A. Alsanie, "Achievable RF coverage and system capacity using millimeter wave cellular technologies in 5G networks," in *Proc. 27th Can. Conf. Elect. Comput. Eng. (CCECE)*, 2014, pp. 1–6.

[2] A. I. Sulyman, A. T. Nassar, M. K. Samimi, G. R. MacCartney, T. S. Rappaport, and A. Alsanie, "Radio propagation path loss models for 5G cellular networks in the 28 GHz and 38 GHz millimeter-wave bands," *IEEE Commun. Mag.*, vol. 52, no. 9, pp. 78–86, Sep. 2014.

[3] "Cisco annual Internet report (2018-2023) White Paper," Cisco Syst., San Jose, CA, USA, White Paper, Mar. 2020. Accessed: Jun. 5, 2021. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11–741490.pdf?dtid=osscdc000283

[4] A. Nassar and Y. Yilmaz, "Reinforcement learning for adaptive resource allocation in fog ran for IoT with heterogeneous latency requirements," *IEEE Access*, vol. 7, pp. 128014–128025, 2019.

[5] K. Winter, "For self-driving cars, there's big meaning behind one big number: 4 terabytes," Intel.com, Santa Clara, CA, USA. Accessed: Jun. 5, 2021. [Online]. Available: https://www.intc.com/news-events/press-releases/detail/237/intel-editorial-for-self-driving-cars-theres-big

[6] M. Peng, Y. Sun, X. Li, Z. Mao, and C. Wang, "Recent advances in cloud radio access networks: System architectures, key techniques, and open issues," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 3, pp. 2282–2308, 3rd Quart., 2016.

[7] Z. Zhao, M. Peng, Z. Ding, W. Wang, and H. V. Poor, "Cluster content caching: An energy-efficient approach to improve quality of service in cloud radio access networks," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 5, pp. 1207–1221, May 2016.

[8] M. Peng, C. Wang, V. Lau, and H. V. Poor, "Fronthaul-constrained cloud radio access networks: Insights and challenges," *IEEE Wireless Commun.*, vol. 22, no. 2, pp. 152–160, Apr. 2015.

[9] W. Wang, V. K. Lau, and M. Peng, "Delay-aware uplink fronthaul allocation in cloud radio access networks," *IEEE Trans. Wireless Commun.*, vol. 16, no. 7, pp. 4275–4287, Jul. 2017.

[10] S. Wang, X. Zhang, Y. Zhang, L. Wang, J. Yang, and W. Wang, "A survey on mobile edge networks: Convergence of computing, caching and communications," *IEEE Access*, vol. 5, pp. 6757–6779, 2017.

[11] Y.-Y. Shih, W.-H. Chung, A.-C. Pang, T.-C. Chiu, and H.-Y. Wei, "Enabling low-latency applications in fog-radio access networks," *IEEE Netw.*, vol. 31, no. 1, pp. 52–58, Jan. 2017.

[12] S.-H. Park, O. Simeone, and S. S. Shitz, "Joint optimization of cloud and edge processing for fog radio access networks," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, 2016, pp. 315–319.

[13] G. P. Fettweis, "The tactile Internet: Applications and challenges," *IEEE Veh. Technol. Mag.*, vol. 9, no. 1, pp. 64–70, Mar. 2014.

[14] T. Lin, H. Rivano, and F. Le Mouël, "A survey of smart parking solutions," *IEEE Trans. Intell. Transp. Syst.*, vol. 18, no. 12, pp. 3229–3253, Dec. 2017.

[15] T. Anagnostopoulos *et al.*, "Challenges and opportunities of waste management in IoT-enabled smart cities: A survey," *IEEE Trans. Sustain. Comput.*, vol. 2, no. 3, pp. 275–289, Jul.-Sep. 2017.

[16] P. Barsocchi, P. Cassara, F. Mavilia, and D. Pellegrini, "Sensing a city's state of health: Structural monitoring system by Internet-of-Things wireless sensing devices," *IEEE Consum. Electron. Mag.*, vol. 7, no. 2, pp. 22–31, Mar. 2018.

[17] Z. Hu, Z. Bai, K. Bian, T. Wang, and L. Song, "Real-time fine-grained air quality sensing networks in smart city: Design, implementation, and optimization," *IEEE Internet Things J.*, vol. 6, no. 5, pp. 7526–7542, Oct. 2019.

[18] L. Ruge, B. Altakrouri, and A. Schrader, "Soundofthecity—continuous noise monitoring for a healthy city," in *Proc. IEEE Int. Conf. Pervasive Comput. Commun. Workshops (PERCOM Workshops)*, 2013, pp. 670–675.

[19] P. T. Daely, H. T. Reda, G. B. Satrya, J. W. Kim, and S. Y. Shin, "Design of smart LED streetlight system for smart city with Web-based management system," *IEEE Sensors J.*, vol. 17, no. 18, pp. 6100–6110, Sep. 2017.

[20] M. Teliceanu, G. C. Lazaroiu, and V. Dumbrava, "Consumption profile optimization in smart city vision," in *Proc. 10th Int. Symp. Adv. Topics Elect. Eng. (ATEE)*, 2017, pp. 876–881.

[21] F. Heimgaertner, S. Hettich, O. Kohlbacher, and M. Menth, "Scaling home automation to public buildings: A distributed multiuser setup for openhab 2," in *Proc. Global Internet Things Summit (GIoTS)*, 2017, pp. 1–6.

[22] "Study on scenarios and requirements for next generation access technologies (release 14), v14.2.0," 3GPP, Sophia Antipolis, France, 3GPP Rep. TR 38.913,Mar. 2017, pp. 23–25. Accessed: Jun. 5, 2021. [Online]. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=2996

[23] G. J. Sutton *et al.*, "Enabling technologies for ultra-reliable and low latency communications: from phy and mac layer perspectives," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 3, pp. 2488–2524, 3rd quart., 2019.

[24] J. Wang, J. Liu, and N. Kato, "Networking and communications in autonomous driving: A survey," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 2, pp. 1243–1274, 2nd quart., 2019.

[25] P. Schulz *et al.*, "Latency critical IoT applications in 5G: Perspective on the design of radio interface and network architecture," *IEEE Commun. Mag.*, vol. 55, no. 2, pp. 70–78, Feb. 2017.

[26] R. H. Goudar and H. N. Megha, "Next generation intelligent traffic management system and analysis for smart cities," in *Proc. Int. Conf. Smart Technol. Smart Nation (SmartTechCon)*, 2017, pp. 999–1003.

[27] L. Lin, X. Liao, H. Jin, and P. Li, "Computation offloading toward edge computing," *Proc. IEEE*, vol. 107, no. 8, pp. 1584–1607, Aug. 2019.

[28] H. Xiang, W. Zhou, M. Daneshmand, and M. Peng, "Network slicing in fog radio access networks: Issues and challenges," *IEEE Commun. Mag.*, vol. 55, no. 12, pp. 110–116, Dec. 2017.

[29] A. T. Nassar and Y. Yilmaz, "Dynamic network slicing for fog radio access networks," in *Proc. IEEE Global Conf. Signal Inf. Process. (GlobalSIP)*, 2019, pp. 1–5.

[30] I. Afolabi, T. Taleb, K. Samdanis, A. Ksentini, and H. Flinck, "Network slicing and softwarization: A survey on principles, enabling technologies, and solutions," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 3, pp. 2429–2453, 3rd Quart., 2018.

[31] T. Dang and M. Peng, "Delay-aware radio resource allocation optimization for network slicing in fog radio access networks," in *Proc. IEEE Int. Conf. Commun. Workshops (ICC Workshops)*, 2018, pp. 1–6.

[32] L. Tang, X. Zhang, H. Xiang, Y. Sun, and M. Peng, "Joint resource allocation and caching placement for network slicing in fog radio access networks," in *Proc. IEEE 18th Int. Workshop Signal Process. Adv. Wireless Commun. (SPAWC)*, 2017, pp. 1–6.

[33] Y. Sun, M. Peng, S. Mao, and S. Yan, "Hierarchical radio resource allocation for network slicing in fog radio access networks," *IEEE Trans. Veh. Technol.*, vol. 68, no. 4, pp. 3866–3881, Apr. 2019.

[34] V. N. Ha and L. B. Le, "End-to-end network slicing in virtualized ofdma-based cloud radio access networks," *IEEE Access*, vol. 5, pp. 18675–18691, 2017.

[35] Y. L. Lee, J. Loo, T. C. Chuah, and L. Wang, "Dynamic network slicing for multitenant heterogeneous cloud radio access networks," *IEEE Trans. Wireless Commun.*, vol. 17, no. 4, pp. 2146–2161, Apr. 2018.

[36] A. T. Nassar and Y. Yilmaz, "Resource allocation in fog RAN for heterogeneous iot environments based on reinforcement learning," in *Proc. IEEE Int. Conf. Commun. (ICC)*, 2019, pp. 1–6.

[37] Y. Shi, Y. E. Sagduyu, and T. Erpek, "Reinforcement learning for dynamic resource optimization in 5G radio access network slicing," in *Proc. IEEE 25th Int. Workshop Comput. Aided Model. Design Commun. Links Netw. (CAMAD)*, 2020, pp. 1–6.

[38] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[39] R. Li *et al.*, "Deep reinforcement learning for resource management in network slicing," *IEEE Access*, vol. 6, pp. 74429–74441, 2018.

[40] F. Wei, G. Feng, Y. Sun, Y. Wang, S. Qin, and Y.-C. Liang, "Network slice reconfiguration by exploiting deep reinforcement learning with large action space," *IEEE Trans. Netw. Service Manag.*, vol. 17, no. 4, pp. 2197–2211, Dec. 2020.

[41] H. Xiang, S. Yan, and M. Peng, "A realization of fog-RAN slicing via deep reinforcement learning," *IEEE Trans. Wireless Commun.*, vol. 19, no. 4, pp. 2515–2527, Apr. 2020.

[42] Y. Hua, R. Li, Z. Zhao, X. Chen, and H. Zhang, "GAN-powered deep distributional reinforcement learning for resource management in network slicing," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 2, pp. 334–349, Feb. 2020.

[43] B. Khodapanah, A. Awada, I. Viering, A. N. Barreto, M. Simsek, and G. Fettweis, "Slice management in radio access network via deep reinforcement learning," in *Proc. IEEE 91st Veh. Technol. Conf. (VTC)*, 2020, pp. 1–6.

[44] R. Li, C. Wang, Z. Zhao, R. Guo, and H. Zhang, "The LSTM-based advantage actor-critic learning for resource management in network slicing with user mobility," *IEEE Commun. Lett.*, vol. 24, no. 9, pp. 2005–2009, Sep. 2020.

[45] W. Guan, H. Zhang, and C. V. Leung, "Slice reconfiguration based on demand prediction with dueling deep reinforcement learning," in *Proc. GLOBECOM IEEE Global Commun. Conf.*, 2020, pp. 1–6.

[46] Y. Abiko, T. Saito, D. Ikeda, K. Ohta, T. Mizuno, and H. Mineno, "Flexible resource block allocation to multiple slices for radio access network slicing using deep reinforcement learning," *IEEE Access*, vol. 8, pp. 68183–68198, 2020.

[47] Y. Liu, J. Ding, and X. Liu, "A constrained reinforcement learning based approach for network slicing," in *Proc. IEEE 28th Int. Conf. Netw. Protocols (ICNP)*, 2020, pp. 1–6.

[48] G. Sun, G. O. Boateng, D. Ayepah-Mensah, G. Liu, and J. Wei, "Autonomous resource slicing for virtualized vehicular networks with D2D communications based on deep reinforcement learning," *IEEE Syst. J.*, vol. 14, no. 4, pp. 4694–4705, Dec. 2020.

[49] Y. Xu *et al.*, "Constrained reinforcement learning for resource allocation in network slicing," *IEEE Commun. Lett.*, vol. 25, no. 5, pp. 1554–1558, May 2021.

[50] S. de Bast, R. Torrea-Duran, A. Chiumento, S. Pollin, and H. Gacanin, "Deep reinforcement learning for dynamic network slicing in ieee 802.11 networks," in *Proc. IEEE INFOCOM Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, 2019, pp. 264–269.

[51] G. Sun, Z. T. Gebrekidan, G. O. Boateng, D. Ayepah-Mensah, and W. Jiang, "Dynamic reservation and deep reinforcement learning based autonomous resource slicing for virtualized radio access networks," *IEEE Access*, vol. 7, pp. 45758–45772, 2019.

[52] R. S. Sutton, and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 1998.

[53] J. Fan, Z. Wang, Y. Xie, and Z. Yang, "A theoretical analysis of deep *Q*-learning," in *Proc. Learn. Dyn. Control*, 2020, pp. 486–489.

[54] S. S. Du, J. D. Lee, H. Li, L. Wang, and X. Zhai, "Gradient descent finds global minima of deep neural networks," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 1675–1685.

[55] S. S. Du, X. Zhai, B. Poczos, and A. Singh, "Gradient descent provably optimizes over-parameterized neural networks," 2018. [Online]. Available: arXiv:1810.02054.

Almuthanna Nassar received the B.Sc. degree in electrical engineering from Jordan University of Science and Technology, Irbid, Jordan, in 2006, and the M.Sc. degree in electrical engineering from King Saud University, Riyadh, Saudi Arabia, in 2014. He is currently pursuing the Ph.D. degree in electrical engineering with the University of South Florida, Tampa, FL, USA.

He is also with the Secure and Intelligent Systems Laboratory, EE Department, University of South Florida. He was a Radio Access Network Planning Specialist Manager with Etihad Etisalat Company (mobily), Riyadh, from 2009 to 2017. He has more than ten years of industry experience in RAN planning and design of multitechnology cellular networks. His current research interests include wireless communications, IoT networking, intelligent vehicular systems, and reinforcement learning.

Mr. Nassar received the IEEE ComSoc Student Travel Grant, IEEE International Conference on Communications in 2019, and the Dissertation Completion Award from the University of South Florida in 2021.

Yasin Yilmaz (Senior Member, IEEE) received the Ph.D. degree in electrical engineering from Columbia University, New York, NY, USA, in 2014.

He is currently an Assistant Professor of Electrical Engineering with the University of South Florida, Tampa, FL, USA, where he directs the Secure and Intelligent Systems Laboratory. His research interests include machine learning, statistical signal processing, and their applications to computer vision, cybersecurity, cyber-physical systems, the Internet-of-Things networks, communication systems, energy systems, transportation systems, social, and environmental systems.

Dr. Yilmax was a recipient of the Collaborative Research Award from Columbia University in 2015, the Young Investigator Award from the Southeastern Center for Electrical Engineering Education in 2017, the Best Paper Honorable Mention Award from the ACM Conference on Recommender Systems in 2019, and the First Place Award from the NIST Automated Streams Analysis for Public Safety Challenge in 2020.