IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received April 27, 2022, accepted May 13, 2022, date of publication May 26, 2022, date of current version June 14, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3178157

Automated Deployment of Virtual Network Function in 5G Network Slicing Using Deep Reinforcement Learning

ANUAR OTHMAN[1], (Member, IEEE), NAZRUL A. NAYAN[1,2], (Member, IEEE), AND SITI N. H. S. ABDULLAH[3]

[1]Department of Electrical Electronics and Systems Engineering, Universiti Kebangsaan Malaysia, Bangi 43600, Malaysia
[2]Institut Islam Hadhari, Universiti Kebangsaan Malaysia, Bangi 43600, Malaysia
[3]Center for Artificial Intelligence Technology, Universiti Kebangsaan Malaysia, Bangi 43600, Malaysia

Corresponding author: Nazrul A. Nayan (nazrul@ukm.edu.my)

This work was supported in part by the Ministry of Education, Malaysia, and Universiti Kebangsaan Malaysia under Grant FRGS/1/2019/TK04/UKM/02/4.

ABSTRACT Fifth-generation mobile technologies introduce the concept of network slicing, which allows the creation of logical networks consisting of network services and the associated physical and virtual network functions. The early form of network slicing allowed for fixed resource allocation and static network function deployment. However, this approach can lead to inefficiency and service degradation. This study aims to optimize the deployment of virtual network functions within a hybrid cloud infrastructure from the perspective of mission-critical communications. The first task involves designing a deep reinforcement learning-based scheme to determine a significant deployment policy that minimizes the overall delays and costs of logical networks. The scheme performance is evaluated by using a simulated traffic dataset that followed Poisson distributions for a wide range of configurations. In dynamic environments with stationary traffic patterns, simulation results show that the scheme outperforms the one-step look-ahead and fixed-location algorithms by 35.80% and 52.16%, respectively, on average. A value iteration-based scheme is used as a benchmark and only surpasses the proposed scheme by 3.5% on average. Simulation results using a real-world traffic dataset show that the scheme can support nonstationary traffic patterns and cater to large-scale scenarios with many suitable deployment locations by leveraging a function that indicates the relative importance of selecting one location over the others.

INDEX TERMS 5G mobile communication, decision support systems, edge computing, intelligent agents, machine learning, multi-layer neural network, network function virtualization.

I. INTRODUCTION

The advent of fifth-generation (5G) mobile technologies has introduced the sophisticated concept of network slicing, which allows the creation of logical networks in a common infrastructure with appropriate isolation, resources, and optimized topology [1]. This concept is considered one of the key enablers of ultralow latency in 5G systems [2]. A logical network is composed of network services and their associated functions that can be physical or virtual [3]. It can serve different uses, such as smart factories, autonomous vehicles, and mission-critical services (MCS) for public safety agencies [4]. In previous studies, we reviewed the characteristics of a public-safety-grade network [5] and discussed the resource allocation schemes in 5G from the perspective of MCS [6]. The early form of network slicing enabled the creation of logical networks with a fixed resource allocation and static network function deployment. However, this approach may lead to inefficiencies, especially when logical networks are underutilized, and may cause service degradation in the case of network function overload or dynamic topology changes. In the case of a virtual network function (VNF), migrating the function to a better location when the network is congested is crucial. In summary, network slicing improves 5G flexibility and scalability while introducing new challenges, particularly in orchestrating the diverse resources

The associate editor coordinating the review of this manuscript and approving it for publication was Alba Amato.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

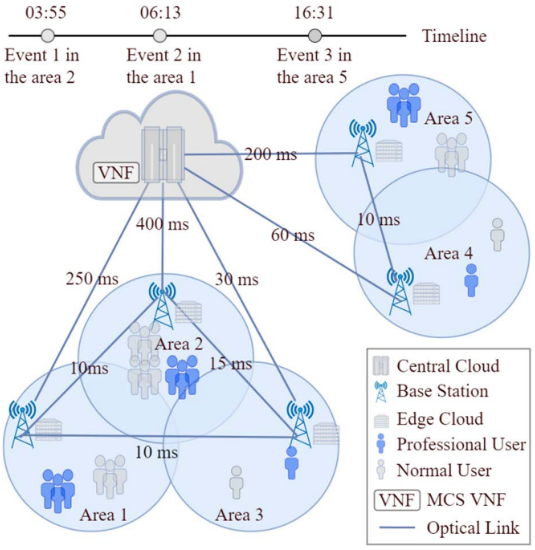

FIGURE 1. Illustration of MCS use case in a 5G mobile network. Edge clouds are hosted at different base stations and directly connected to a central cloud through fiber optics. Base stations are clustered based on their locations and all edge clouds within a cluster are interconnected using full mesh topology.

of a network function. The deployment of network functions must be managed efficiently within a hybrid cloud infrastructure comprising central and edge computing resources.

A sample scenario of network slicing is illustrated in Fig. 1, in which an agency procures a logical network, referred to as the main critical slice in this study, from a mobile network operator and provides an MCS to its professional users. This service requires a reliable and resilient network that can guarantee adequate quality of service in all scenarios [7]. Each base station serves all the users within its coverage area, and a user can only connect to a single base station. Other network slices have a lower priority for consumers. Initially, the traffic load generated by professional users at all locations is low, and the delay requirements are satisfied by the main critical slice. The slice is created by deploying an MCS VNF in the central cloud. However, several planned and unplanned events, such as local elections or riots, occur at random locations within a certain period (e.g., one day). Consequently, the traffic flow generated by the users at these locations increase considerably. The loads of the links between the central cloud and the affected base stations also increase. For example, during the first event, professional users in the area 2 start to experience delays higher than 300 ms, which exceeds the key performance indicator suggested by 3GPP for MCS [8]. An ancillary critical slice can then be created by deploying a distributed MCS VNF in the nearest edge cloud to reduce the overall latency within the affected area. Given that the location of the affected users changes frequently due to the randomness of events, the distributed MCS VNF may need to be migrated accordingly to maintain guaranteed service quality. A good strategy that considers all relevant factors, such as stochastic traffic load, must be established to avoid unnecessary costs and delays incurred from deployment and migration processes.

Professional users typically communicate in groups and work within predefined operational areas [9]. Therefore, this study assumes that only one ancillary critical slice can be created concurrently for a cluster of edge clouds within an operational area. When the traffic flow increases, the system must decide whether to allow the main critical slice to continue serving the affected users or create an ancillary slice by deploying the distributed MCS VNF in one edge cloud. The options include those nearest to the affected users (i.e., hosted at the local base station) or others (i.e., hosted at the neighboring base stations). In the two cases, no communication delay or cost to the central cloud occurs but at the expense of migration and processing delays and costs of the MCS VNF. The communication delay and cost associated with inter-base station links are involved if a neighboring base station is selected. If the main critical slice remains to serve professional users, then migration and processing delays and costs are not involved. However, in this case, communication delays and costs to the central cloud are exacerbated. Therefore, the task of the proposed VNF deployment scheme is to determine the best strategy, that is, an optimal policy for the creation of ancillary critical slices and migration of MCS VNFs within a hybrid cloud infrastructure.

This study investigated the VNF deployment problem in 5G network slicing from the perspective of mission-critical communication. In summary, our main contributions are as follows:

1) We defined the VNF deployment problem in a stochastic environment within a hybrid cloud infrastructure. We formulated the problem as a Markov decision process (MDP) and described its states, actions, state transitions, and reward function.
2) We proposed a reinforcement learning (RL)-based scheme to automatically determine a near-optimal deployment policy for minimizing the overall costs and delays associated with professional users within a cluster.
3) We evaluated the proposed scheme by using a simulated traffic dataset and compared its performance with that of a benchmark scheme that uses a dynamic programming-based algorithm. We measured its adaptability and scalability by using a real-world traffic dataset recorded for Shanghai Telecom.

The remainder of this paper is organized as follows. Section II reviews the related work on VNF deployment schemes within a hybrid cloud infrastructure. Section III presents the system model and problem formulation. Section IV describes the proposed automated VNF deployment scheme using RL-based algorithms. Section V presents a performance evaluation of the proposed scheme on simulated and real-world traffic datasets. Section VI provides the conclusion.

II. RELATED WORK

Recently, network slicing has elicited increasing interest in academia and industry. Challenges in orchestrating the

deployment of slice-constituent VNFs from the perspective of services with strict delay requirements were discussed in [10]–[13]. Solozabal *et al.* [10] proposed a hierarchical distributed MCS architecture in a non-standalone 5G system, which consists of deploying only the user plane or the user and control planes of the MCS VNF in an edge cloud. The target is to reduce the overall service latency and facilitate resource scaling of the VNF. The proposed architecture requires the standardization of the complete separation between the user and control planes. A dynamic deployment scheme that optimizes VNF placement for a set of network slices with stringent resource requirements was proposed in [11]. The objective is to maximize the number of network slices that are admitted to the network. The problem was formulated as an integer linear programming problem and a heuristic algorithm that leverages a best-fit decreasing strategy was proposed. However, the authors ignored the dynamic changes in service requirements in their work.

Tang *et al.* [12] proposed a dynamic VNF chain migration scheme that first predicts the future resource requirements of a chain by using an algorithm based on a deep belief network. The predicted data were then used to determine the migration policy by using a tabu search-based algorithm. However, the authors excluded other factors, such as processing and communication overheads, from the optimization objective. Although the above studies focused on VNF deployment, Guo *et al.* concentrated on placing an edge cloud at candidate base stations and allocating its corresponding users [13]. The authors formulated the task as a multi-objective optimization problem and proposed a scheme consisting of k-means and mixed-integer quadratic programming. However, the authors ignored the dynamic workload of the base stations. In summary, these studies only focused on optimizing the VNF or edge cloud deployment against static service requirements, without considering the stochastic changes in traffic load or user mobility.

Many researchers have leveraged the RL framework to address the dynamic deployment of VNFs or microservices in stochastic environments [14]–[16]. The RL framework is a machine learning paradigm that automates decision-making tasks directly from the experience gained through interaction with an uncertain environment. Wang *et al.* [14] proposed a dynamic coordination of microservices among hybrid clouds in an autonomous vehicle use case. The proposed scheme uses a tabular-based RL algorithm to dynamically select the edge cloud and process the tasks submitted by users depending on their trajectories and current microservice deployment. Luo *et al.* [15] proposed a deep RL-based scheme for automatically scaling a VNF chain distributed over several datacenters. The proposed scheme consists of predicting the traffic flow by using a recurrent neural network and exploiting the prediction to determine the placement and instance size of each VNF in the chain. However, the authors disregarded the resource limitation of a datacenter in the reward signal, which is common in hybrid cloud infrastructure. Our work is related to the recent work of Schneider *et al.* [16], who proposed the autonomous coordination of network services against stochastic traffic loads. The coordination task consists of placing and scaling service components, such as VNF or microservices, and scheduling the traffic flow of the service. The authors deployed a deep RL framework to learn the probability of selecting a cloud to process incoming flows. The present study used this framework to directly determine VNF placement and scaling. In summary, previous studies focused only on consumer services without considering those used by professional users (e.g., group-based communication services deployed on a reliable and resilient network slice).

In addition to network slicing, other key enablers of ultralow latency in 5G are multiservice air interfaces, mobile edge computing, and direct communication between devices. Nadeem *et al.* [17] highlighted the challenges in integrating these enablers, including resource allocation, device limitations, and mobility management. Ramly *et al.* [18] investigated the effects of various 5G radio spectra, speeds, and frequency diversities on the latency performance of industrial automation in a smart factory use case. Yousafzai *et al.* [19] proposed a computational offloading framework based on lightweight process migration for resource-intensive IoT applications by leveraging edge cloud technology. Ali *et al.* [20] reviewed the application of deep RL to optimize the computation offloading function in the Internet of vehicles. The proposed scheme automatically schedules an offloaded task from a vehicle and allocates resources to minimize the task latency and energy costs. The analytical results demonstrate that the proposed scheme can support dynamic environments with stochastic vehicle mobility. Fodor *et al.* [21] examined the key challenges in implementing device-to-device communications and proposed a concept that performs dynamic clustering of out-of-coverage devices. The aforementioned enabling technologies improve the reliability of 5G to meet the strict latency requirements of diverse use cases.

III. PROBLEM FORMULATION

The function of the automated VNF deployment scheme is to determine a near-optimal policy for deployment and migration of MCS VNFs. First, physical and logical networks and traffic flow were modeled. The problem was then formulated as an MDP, which consists of a set of states, a set of actions, a transition function, and a reward function.

A. SYSTEM MODEL

A mobile network is represented as an undirected graph $\mathcal{G} = (\mathcal{J} \subseteq \mathcal{K}, \mathcal{L})$, where $\mathcal{J} = \{1, 2, \ldots, J\}$, $\mathcal{K} = \{1, 2, \ldots, K\}$, and $\mathcal{L} = \{1, 2, \ldots, L\}$ denote a set of edge clouds, base stations, physical backhauls, and inter-base station links, respectively. An edge cloud can be hosted at one of the base stations, which is a key requirement outlined by the Next Generation Mobile Network Alliance to enhance the flexibility of 5G systems [3]. Each base station can host at most one edge cloud, and only one ancillary critical slice can be created within a cluster of K base stations, as illustrated in Fig. 2. An ancillary critical slice in a cluster comprises a set of network services.

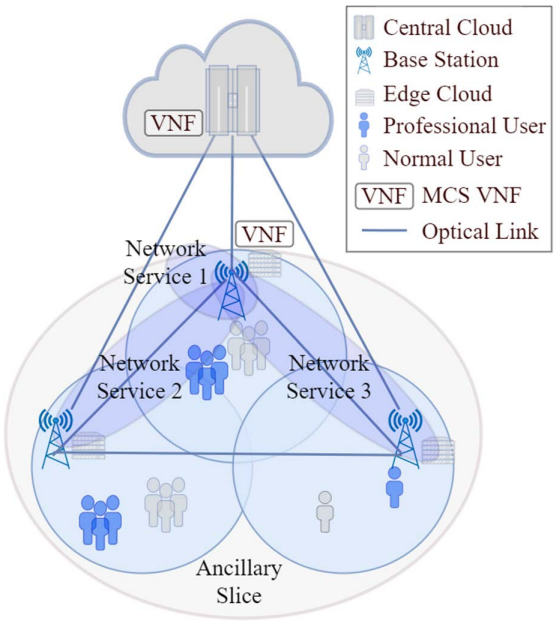

FIGURE 2. Illustration of MCS VNF deployment in 5G network slicing. An ancillary slice can be created by deploying a distributed MCS VNF at edge cloud to reduce the overall latency of professional users.

A network service, also known as a service function chain, comprises ordered VNFs. It is modeled as $\mathcal{G}^V = (\mathcal{N}^V, \mathcal{L}^V)$, where $\mathcal{N}^V$ and $\mathcal{L}^V$ denote a set of VNFs and virtual backhaul and inter-base station links, respectively. The components of each network service are the MCS VNF, the physical network function, and the VNF of the 5G radio access network. The MCS VNF includes 5G core network functionalities and can be shared by all network services belonging to the critical slice within the cluster. It was inspired by the proposed distributed architecture [22] that enables its user and control plane functions to be deployed in the edge cloud.

The professional users within the cluster are grouped according to their base station and the group set is denoted as $\mathcal{U} = \mathcal{K}$. Each traffic flow generated by a member of the group k at its local base station is defined by its base station identifier, time of arrival, requested data rate, and termination time. The flow then traverses all the components of its corresponding network service in a specified order. The central or edge cloud that accommodates the MCS VNF is denoted as the serving cloud. An edge cloud $j \in \mathcal{J}$ can support several VNFs, and a physical link $l \in \mathcal{L}$ can be mapped with several virtual ones. The delay of a physical link depends on the bit rate and distance between the two nodes that the link connects, where a node can be either one of the base stations or the central cloud.

B. STATE SPACE

The system state space defines a set of all possible configurations of the critical slice and the number of traffic flows of professional users $\mathcal{U}$. The MDP state space $\mathcal{S}$ is defined as follows: A state $s \epsilon \mathcal{S}$ is a two-sized tuple $(x(t), \mathbf{n}(t))$, where $x(t)$ indicates whether only a main critical slice is active $(x(t) = 0)$ or an ancillary slice has been created by deploying the distributed MCS VNF in the edge cloud j at timestep t, $(x(t) = j)$. The vector $\mathbf{n}(t)$ represents the total traffic flows of the professional users $\mathcal{U}$ at timestep t. Subsequently, any modification of the total traffic flow of the group k at timestep t, $n_k(t)$ triggers the decision process of the proposed VNF deployment scheme.

C. ACTION SPACE

The system is triggered by events that correspond to changes in the total traffic flow. For simplicity, the events are assumed to never occur simultaneously, and each event is treated as a different event. The set $\mathcal{A} = \{0\} \cup \mathcal{J}$ represents all possible actions that can be performed on the system based on the current system state s. The latter comprises a triggering event and current configuration of the critical slice. The action j consists of deploying the MCS VNF in the edge cloud j to serve professional users $\mathcal{U}$, whereas the action 0 corresponds to deactivating the MCS VNF in the serving cloud and reinstating $\mathcal{U}$ to the main critical slice.

D. TRANSITION FUNCTION

As a result of the action taken in each state s, the system transitions to the next state s' following a transition function $P(s, a, s')$. The arrival of the traffic flow of the group k is assumed to follow a Poisson process $\mathcal{P}_k$ with an associated rate of λ_k. Upon admission to the network, the traffic flow remains active following an exponentially distributed time with an average $1/\mu_k$. On the basis of these assumptions, the transition rates between system states are derived. Transitions to the next state with the addition or removal of the flow of group k occur at rates of λ_k and μ_k, respectively. The transition function of the system is defined as follows: For $s = (x(t), \mathbf{n}(t)) \forall a$,

$$P(s, a, s) = \begin{cases} \dfrac{\mu_k}{v(n_k(t))}, & if\ n_k(t) > 0 \\ \dfrac{\lambda_k}{v(n_k(t))}, & if\ n_k(t) \geq 0, \end{cases} \tag{1}$$

where

$$v\left(n_k\left(t\right)\right) = \sum\nolimits_k \lambda_k + \rho_k \mu_k,$$
$$\rho_k = \begin{cases} 1, & if\ n_k(t) > 0 \\ 0, & if\ n_k(t) = 0. \end{cases} \tag{2}$$

E. REWARD FUNCTION

In addition to the state transition, the system generates an immediate reward or penalty signal that is influenced by the action a taken in the previous state s [23]. In the VNF deployment problem, the creation of an ancillary slice in one of the edge clouds or the reinstatement of the professional users $\mathcal{U}$ to the main slice incurs different penalties represented by the overall delays experienced by $\mathcal{U}$ and overall costs to the agency. The overall delay o_d is defined as the sum of the processing (d_p), communication (d_c), and migration

(d_m) delays, and the overall cost o_c is the sum of the processing (c_p), communication (c_c), and migration (c_m) costs. The function of the automated VNF deployment scheme is to determine a significant policy that minimizes the long-term penalty, that is, the negative reward associated with professional users within a cluster. Therefore, in optimizing a single objective, the long-term reward or return, R_T is defined as the negative sum of either o_d or o_c over time window T. For the optimization of multiple objectives, R_T is defined as the negative sum of the weighted normalized o_d and o_c over time window T,

$$R_T = -\sum \left[(1-\omega) \left(\frac{o_d - o_{d,min}}{o_{d,max} - o_{d,min}} \right) + \omega \left(\frac{o_c - o_{c,min}}{o_{c,max} - o_{c,min}} \right) \right], \tag{3}$$

where weight ω, $(0 \leq \omega \leq 1)$ represents the trade-off between achieving the two objectives.

1) PROCESSING DELAY AND COST

After the deployment of MCS VNF in the serving cloud, the processing of multiple traffic flows from the professional users $\mathcal{U}$ at this new location incurs additional costs and delays owing to the allocation of additional processing capacity and the availability of a limited amount of resources. By contrast, given that the central cloud has more resources than all edge clouds, its processing and queuing delays are considered negligible. The central cloud always hosts the main critical slice to serve other professional users in the network. Thus, the processing cost is ignored in the optimal policy computation of the ancillary critical slice. Let $p_x(t)$ and $y_x(t)$ denote the average processing and queuing delays per traffic flow of the serving cloud at time step t, $g_k(t)$ denote the average processing capacity required by each flow of the group k at time step t, f_x denote the cost per unit of processing capacity of the serving cloud and Δt denote the time interval between two consecutive decision processes. Then, d_p and c_p can be calculated as:

$$d_p(t) = \sum\nolimits_k (n_k(t) \times (p_x(t) + y_x(t))), \tag{4}$$

$$c_p(t) = \sum\nolimits_k (n_k(t) \times g_k(t) \times \Delta t \times f_x). \tag{5}$$

2) COMMUNICATION DELAY AND COST

Communication between a member of professional users $\mathcal{U}$ and its MCS VNF involves several steps. First, data are transmitted from the user terminal to its local base station through a radio link. Depending on its associated network service, data are either processed at the same location or transmitted to the central cloud via a backhaul link or to a neighbor base station via an inter-base station link. Therefore, the deployment of MCS VNFs has different effects on the communication delay and cost owing to the different transmission media involved. An inter-base station link is established only when the serving cloud is not located at the local base station of the group k, and other links are always established in all cases, even when the group k is served by the serving cloud at its local base station. Let $h_k(t)$ denote the average radio link delay of each flow of the group k at time step t, $b_{k,x}$ denote the communication delay per traffic flow of the inter-base station or backhaul links between the base station k and the serving cloud, $m_k(t)$ denote the average bandwidth capacity required by each flow of the group k at time step t, and $e_{k,x}$ denote the cost per unit bandwidth capacity of the inter-base station or backhaul links. Then, d_c and c_c can be calculated as:

$$d_c(t) = \sum\nolimits_k \left(n_k(t) \times \left(h_k(t) + b_{k,x}\right)\right), \tag{6}$$

$$c_c(t) = \sum\nolimits_k \left(n_k(t) \times m_k(t) \times \Delta t \times e_{k,x}\right). \tag{7}$$

3) MIGRATION DELAY AND COST

The creation of an ancillary critical slice incurs additional costs and delays due to the process of deploying an MCS VNF in one of the edge clouds and migrating its user data traffic from one location to another. The migration process is inspired by the hierarchical MCS architecture [10], where user data traffic can be directly forwarded to a distributed MCS VNF without involving a control plane in the central cloud. The deployment and migration processes follow the procedure proposed by Clark *et al.* [24], which consists of replicating the MCS VNF and its user data at a new location, whereas those at the original location remain active. When a certain threshold is reached, the original MCS VNF is stopped and the remaining user data are migrated. Upon completion, the professional users $\mathcal{U}$ are then served by the new MCS VNF in the serving cloud. For the migration delay computation, only the migration downtime is considered, that is, the time that the professional users $\mathcal{U}$ take to shift from the previous MCS VNF to the new one. For the migration cost computation, the time required to complete the deployment and migration processes is considered. Let v_x denote the initialization time of the serving cloud, $z_k(t)$ denote the size of the remaining user data of the group k to be migrated, $w_k(t)$ denote the average bandwidth capacity required by the group k during the deployment and migration processes, and Δt_m denote the total completion time of the two processes. Then, d_m and c_m can be calculated as:

$$d_m(t) = v_x + \sum\nolimits_i \frac{z_k(t)}{w_k(t)}, \tag{8}$$

$$c_m(t) = \sum\nolimits_k \left(\frac{w_k(t)}{m_k(t)} \times \Delta t_m \times e_{k,x} \right). \tag{9}$$

IV. AUTOMATED VNF DEPLOYMENT SCHEME

An optimal policy for an MDP problem can be computed efficiently by using a dynamic programming framework. However, it requires high computational capacity for a system with a large state space, thereby limiting its scalability [23]. The framework also requires the availability of complete information on the MDP model, including the transition function, which is typically unavailable in real-world scenarios. By contrast to dynamic programming, the RL framework does not require a perfect MDP model.

A. VALUE ITERATION

Algorithm 1, which is a dynamic programming-based algorithm known as value iteration, was first used. It implements the Bellman optimality equation as an update rule, which can be expressed as

$$Q(s,a) = R(s,a) + \sum_{s'} P(s,a,s') \max_{a' \in A} Q(s',a'). \quad (10)$$

The algorithm converges to an optimal policy for discounted finite MDP problems [23]. The Bellman optimality equation states that the value of taking action a in state s under an optimal policy must equal the expected long-term reward for the best action. Algorithm 1 iteratively improves the approximations of the value function $Q(s,a)$, which estimates the expected long-term reward for all state-action pairs. On the basis of the system model in Section III, the environment is a finite MDP because its state space $\mathcal{S}$ and action space $\mathcal{A}$ are finite. The environment dynamics, that is, the arrival rate λ_k and departure rate μ_k of the traffic flows of each group are assumed to be available. The transition function $P(s,a,s)$ is then derived and subsequently exploited in the value function approximation step. Algorithm 1 converges to an optimal policy when the values of two consecutive approximated value functions differ only by a small amount, denoted by θ.

Algorithm 1 Value Iteration

1: Initialize the vector $Q(s,a) = 0$.
2: Initialize the iteration step to 1.
3: Initialize Δ to 0.
4: **while** $\Delta > \theta$ **do**
5: Update the vector $q = Q(s,a)$.
6: Update the vector $Q(s,a)$ using (10)
7: Update $\Delta = max(\Delta, |q - Q(s,a)|)$.
8: **end while**

B. Q-LEARNING

For the RL-based scheme, the transition function $P(s,a,s')$ of the MDP model was assumed to be unknown. Algorithm 2 is a Q-learning algorithm that learns an optimal policy through interactions between its agent and Markovian environment. The convergence of Q-learning to an optimal policy is guaranteed if its learning rate parameter and Markovian environment satisfy certain conditions, as demonstrated by Watkins *et al.* [25]. However, this algorithm only supports discrete state and action spaces. A Q-learning agent works by successively selecting an action a in state s and observes a reward r and the next state s', as illustrated in Fig. 3. The Q-learning agent then updates its estimation of the expected long-term reward (i.e., the Q-value) of taking action a in previous state s. The update is performed by using a constant learning rate α and discount factor γ as follows:

$$Q(s,a) = Q(s,a) + \alpha \left[r + \gamma \max_{a' \in A} Q(s',a') - Q(s,a) \right]. \quad (11)$$

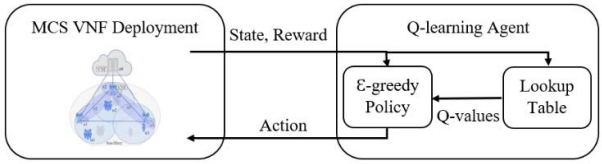

FIGURE 3. High-level view of a Q-learning agent and its components. The agent learns an optimal policy through the experience gained from successive interactions with a Markovian environment.

The learning rate assigns more weight to recent rewards, and the discount factor indicates the present value of future rewards [23]. Q-learning is a tabular-based method because the Q-values for all possible state-action pairs are stored in a table. These values are evaluated to determine the action in each state, where the agent always selects the one that yields the highest Q-value in the case of a greedy policy. Q-learning uses an $\mathcal{E}$-greedy policy where the agent applies the greedy policy most of the time and selects an action randomly with a small probability $\mathcal{E}$. This policy allows the agent to learn an optimal policy by exploiting its knowledge of the most rewarding actions and discovering such knowledge by exploring other possible actions in each state.

Algorithm 2 Q-Learning

1: Initialize the vector $Q(s,a) = 0$.
2: **while** episode number $<$ maximum number of episodes **do**
3: Initialize the state.
4: Initialize the iteration step to 1.
5: **while** s is not a terminal state **do**
6: Using ϵ-greedy policy, select an action a in state s.
7: Observe a reward r and the next state s'.
8: Update the old estimate of $Q(s,a)$ with the new sample using (11).
9: **end while**
10: **end while**

C. DEEP Q-NETWORK

Algorithm 3 is used for the deep RL-based scheme with the assumption of an incomplete MDP model. A deep Q-network (DQN) agent [26] supports discrete and continuous state spaces, but continuous action spaces are not supported. A DQN agent learns an optimal policy by successively selecting an action a in state s, followed by the observation of a reward r and the next state s'. By contrast to Q-learning, an agent does not store an individual Q-value for each state-action pair in a lookup table. It uses a neural network to approximate a value function that can encode the Q-values for all the state-action pairs. The neural network is designated as a critic network $Q(s,a|\theta_Q)$, and its parameters θ_Q are updated by minimizing the loss function L between its approximated Q-values and target values by using an optimization algorithm, such as gradient descent as follows:

$$L = \frac{1}{M} \sum_{i}^{M} \left(y_i - Q(s_i, a_i|\theta_Q) \right)^2, \quad (12)$$

where

$$y_i = \begin{cases} r_i, & \text{if } s'_i \text{ is a terminal state} \\ r_i + \gamma \max\limits_{a' \in A} Q'\left(s'_i, a' | \theta_{Q'}, \alpha_{Q'}, \beta_{Q'}\right), \\ & \text{otherwise.} \end{cases} \quad (13)$$

The target Q-values are obtained by using a target critic network $Q'\left(s, a|\theta_{Q'}\right)$, where its parameters $\theta_{Q'}$ are periodically updated with the critic network parameters θ_Q or at every time step with a smoothing factor. This approach improves DQN's stability against divergence compared with other off-policy algorithms with function approximation and bootstrapping, such as Q-learning with linear function approximation [27]. The DQN agent learns faster than gradient-based convergent methods, such as residual gradient (RG) algorithm that uses gradient descent to minimize the loss of mean squared Bellman error, L_{MBSE} [28]. Wang *et al.* [29] discussed the inefficiencies in RG's learning behavior, which increased its learning time to $\mathcal{O}\left(M^2\right)$, quadratic in the problem size. The worst-case computational complexity of DQN is also $\mathcal{O}\left(M^2\right)$ because the loss function L of DQN is equivalent to L_{MBSE} every time its target network is copied from its critic network.

Algorithm 3 Deep Q-Network (DQN)

1: Initialize the size of the experience buffer to N.
2: Initialize the critic network $Q(s.a|\theta_Q)$ with random parameter values θ_Q.
3: Initialize the target critic network $Q'(s, a|\theta_{Q'})$ with the same parameter values $\theta_{Q'} = \theta_Q$.
4: **while** episode number $<$ maximum number of episodes **do**
5: Initialize the state.
6: Initialize the iteration step to 1.
7: **while** s is not a terminal state **do**
8: Using ϵ-greedy policy, select an action a in state s.
9: Observe a reward r and 1 he next state s'.
10: Store the experience $(s, a, r.s')$ in the experience buffer.
11: Sample a random minibatch of M experiences (s_i, a_i, r_i, s'_i) in the experience buffer.
12: Set the value function target y_i using (13).
13: Update the critic network parameters θ_Q by one-step minimization of the loss L across all sampled experiences using (12).
14: Update the target critic network parameters $\theta_{Q'}$ either at every timestep or periodically.
15: **end while**
16: **end while**

The DQN agent enhances data efficiency by fully exploiting past experiences using the experience replay technique, as illustrated in Fig. 4. The use of neural networks provides inherent support for parallelism [30] and improves DQN scalability to cater to systems with large state and action spaces. This pitfall can be uncovered by generalizing the experiences learned from observed states and exploiting them

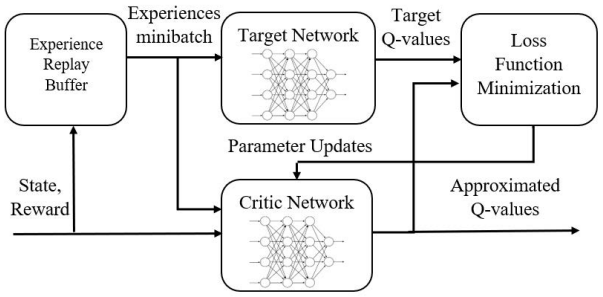

FIGURE 4. Overview of the experience replay technique that pools a sequence of experiences over many episodes into a buffer. Subsequently, a random minibatch of experiences is sampled and used to compute the target Q-values and to update the critic network parameters.

for new states with similar features. The critic network of the DQN agent is constructed by using a neural network with two input paths representing the system state and action, and a single output path representing the approximated Q-value. The input paths consist of two fully connected hidden layers with 64 nodes for the system state and a single fully connected hidden layer with 64 nodes for the action. The two paths are then concatenated into a single path consisting of two fully connected hidden layers with 256 nodes. The rectified linear unit (ReLU) and adaptive moment estimation (ADAM) are defined as the activation function and optimization algorithm, respectively.

D. DUELING DQN

Algorithm 4 is a dueling DQN agent [31] that leverages the efficient learning of a state value function to find an optimal policy in an environment where the number of actions is large. To achieve this, a dueling DQN agent separates the final hidden layer of its critic network into two paths: the first path handles the approximation of the state value function, $V\left(s|\theta, \beta\right)$, and the second approximates a function known as the advantage function, $A\left(s, a|\theta, \alpha\right)$. The parameters of the first and second paths are represented by β and α, respectively, and θ represents the parameters of the remaining layers of the critic network. The loss function L is defined as follows:

$$L = \frac{1}{M} \sum\nolimits_i^M \left(y_i - Q\left(s_i, a_i|\theta_Q, \alpha_Q, \beta_Q\right)\right)^2, \quad (14)$$

where

$$y_i = \begin{cases} r_i \quad if\ s'_i\ is\ terminal \\ r_i + \gamma \max\limits_{a' \in A} Q'\left(s'_i, a'|\theta_{Q'}, \alpha_{Q'}, \beta_{Q'}\right) \\ \quad otherwise. \end{cases} \quad (15)$$

The advantage function indicates the importance of taking action a in state s, relative to the other possible actions. The advantage value is obtained by subtracting the value of being in state s from the value of taking action a in that state. The two paths are then combined in a proceeding aggregation layer to approximate the action value function $Q\left(s, a|\theta, \alpha, \beta\right)$ that encodes the Q-values for all possible state-action pairs. The advantage values averaged over all possible actions were

Algorithm 4 Dueling DQN

Initialize the size of the experience buffer to N.
Initialize the critic network $Q(s, a|\theta_Q, \alpha_Q, \beta_Q)$ with random parameter values θ_Q, α_Q, β_Q.
Initialize the target critic network $Q'(s, a|\theta_{Q'}, \alpha_{Q'}, \beta_{Q'})$ with the same parameter values $\theta_{Q'} = \theta_Q$, $\alpha_{Q'} = \alpha_Q$, $\beta_{Q'} = \beta_Q$.
while episode number $<$ maximum number of episodes **do**
    Initialize the state.
    Initialize the iteration step to 1.
    while s is not a terminal state **do**
        Using ϵ-greedy policy, select an action a in state s.
        Observe a reward r and the next state s'.
        Store the experience (s, a, r, s') in the experience buffer.
        Sample a random minibatch of M experiences (s_i, a_i, r_i, s'_i) in the experience buffer.
        Set the value function target y_i using (15).
        Update the critic network parameters θ_Q by one-step minimization of the loss L across all sampled experiences using (14).
        Update the target critic network parameters $\theta_{Q'}$, $\alpha_{Q'}$, $\beta_{Q'}$ either at every timestep or periodically.
    end while
end while

further subtracted from the estimated advantage value of a state-action pair to improve the agent's stability. The modified critic network is known as a dueling network and can be trained by using the same techniques used for a DQN agent, such as experience replay, target network, and double DQN. The computational complexity of a dueling DQN agent is $\mathcal{O}\left(M^2\right)$, quadratic in the problem size due to the loss minimization.

The dueling network of the agent is constructed by using a neural network with an input path representing the system state and an output path representing the approximated Q-value. The hidden layers include three fully connected layers with 64 nodes, followed by two fully connected layers with 256 nodes. The final layer outputs are then separated into two paths, handling the approximation of the state value function and advantage function. The two paths are combined in the aggregation layer. This architecture was inspired by the work of Wang *et al.* [31], who demonstrated the agent's performance in learning the optimal policies in the Atari domain. The ReLU and ADAM were defined as the activation function and optimization algorithm, respectively.

V. PERFORMANCE EVALUATION

An automated VNF deployment scheme was implemented by using MATLAB R2019b and Python 3.6.4 with essential libraries, including NumPy, Matplotlib, Random, TensorFlow, and Keras. Its performance was evaluated against baselines through comprehensive simulations. The OpenAI Gym toolkit was used to construct the MDP environment in Python. For geospatial information analysis of a real-world dataset, Quantum Geographical Information System (QGIS) 3.16.3 was used.

A. EVALUATION SETUP

1) EVALUATION METRICS

We selected the average return as the evaluation metric due to its superior stability compared with the highly biased maximum average and maximum returns [32]. To report the results, we used the policy optimization view of deep RL agents, which shows the return optimization of a single target policy over several learning episodes rather than the online learning view that considers the entire learning process [33]. On the basis of the available data, the learning process consisted of 1400 and 180 episodes (i.e., days) for the simulated and real-world datasets, respectively. For each scenario involving a real-world dataset, five simulation trials were conducted by using different edge-cloud clusters. The average return of each deep RL agent across the five trials with a 95% confidence interval was then represented. Each return was averaged for the last 60 evaluation episodes. We performed significance testing consisting of Welch's t-test and bootstrap confidence interval test on the final average return to further illustrate the performance range of the proposed scheme [34].

2) SIMULATED INPUT DATASET

A simple scenario with two edge clouds and two professional user groups with varying flow arrival rates of λ_k and active periods of $1/\mu_k$ for all groups was considered. The two edge clouds have the same delay characteristics of 5 ms for the average processing and queuing delays per traffic flow. The average radio link delay of each flow of the group k at timestep t, $h_k\,(t)$ is 5 ms. The communication delays per traffic flow of the inter-base station and backhaul links of the base station k, $b_{k,x}$ are 15 and 30 ms, respectively. The initialization time of an edge cloud is 40 ms, and the size of the remaining user data of the group k to be migrated, $z_k\,(t)$ is 10 Mb per traffic flow. The average bandwidth capacity required by each flow of the group k during the entire deployment and migration processes, $w_k\,(t)$ is 250 Mb/s, and the total process completion time, Δt_m is equal to 9600 ms per traffic flow.

The average processing capacity required by each flow of the group k at time step t, $g_k\,(t)$ is 1000 Hz and the cost per 1000 Hz processing capacity of an edge cloud, f_x is 1 per min. The average bandwidth capacity required by each flow of the group k at timestep t, $m_k\,(t)$ is 0.1 Mb/s, and the cost per 0.1 Mb/s bandwidth capacity of the inter-base station and backhaul links of the base station k, $e_{k,x}$ is 0.75 and 1.5 per min, respectively. On the basis of the parameters defined in Section III, we simulated our input dataset that contained the daily traffic flow of the groups 1 and 2 for a period of 1400 days. Each day is represented by an episode

of 1440 timesteps, where each timestep represents 1 min in a real-world scenario. We computed the transition probabilities of the simulated input dataset $P(s, a, s)$ by counting the number of occurrences of each possible next state for a given current state. The results confirm that the transition probabilities of the simulated data match those derived from the system model.

3) SHANGHAI TELECOM DATASET

The performance of the proposed scheme was evaluated by using a real-world dataset from the Shanghai Telecom mobile network. On the basis of this dataset, Wang *et al.* conducted experiments to evaluate their proposed solutions for managing the placement of edge clouds in smart cities [35] and service recommendations in mobile edge computing [36]. Dulac-Arnold *et al.* [37] highlighted the key challenges in implementing deep RL agents in real-world scenarios, including the need for sample-efficient algorithms and robust approaches to handle partially observable environments. The Shanghai Telecom dataset contains records of Internet access from 9481 mobile users for a period of six consecutive months, where each record indicates the start and end times of a user connection. From a total of 3233 base stations, we selected 2740 base stations located in the metropolitan area of Shanghai. We then grouped them into 274 clusters of varying sizes to simulate the grouping of professional users in accordance with the hypothetical areas of operation. We used the k-means clustering function of QGIS, which assigns each base station to the cluster with the nearest mean. In each cluster, we assume that each base station hosts an edge cloud, professional users generate only 40% of the traffic flow, and the remaining flows originate from commercial users. We retained the definition of the parameters associated with network delays in the simulated dataset.

4) DEEP RL HYPERPARAMETERS

Henderson *et al.* [33] discussed several factors that affect the performance and reproducibility of deep RL agents, including hyperparameter tuning, network architecture selection, and random seed definition. These predetermined factors are crucial for avoiding the misinterpretation of results and ensuring good experimental practice. We first investigated the effects of batch size and network architecture on agent performance by modifying only the hyperparameters of interest, while setting others to default values. We used a batch size of 256 for the default values of the two agents. The network architecture, activation function, and optimization algorithm defined in Sections IV-C and IV-D were used as the default configurations.

We set the following values for the remaining hyperparameters: 1) smoothing factor for target critic updates = 0.001, 2) buffer size = 10,000, 3) discount factor = 0.99, and 4) probability threshold for epsilon-greedy exploration = 0.99, with a decay rate of 0.01. We then varied the batch sizes from 128 to 512 while setting the learning rate of DQN and dueling DQN agents to 0.05 and 0.01, respectively.

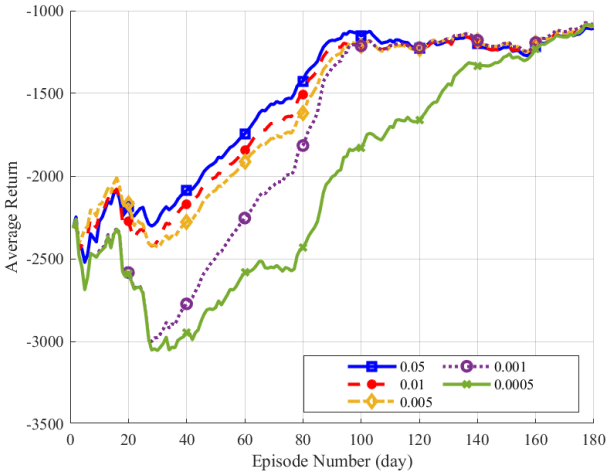

FIGURE 5. DQN agent's performance for different values of learning rate. Its convergence to an optimal policy improves with the increase in the learning rate because large learning rates allow the agent to update its critic network's parameters faster.

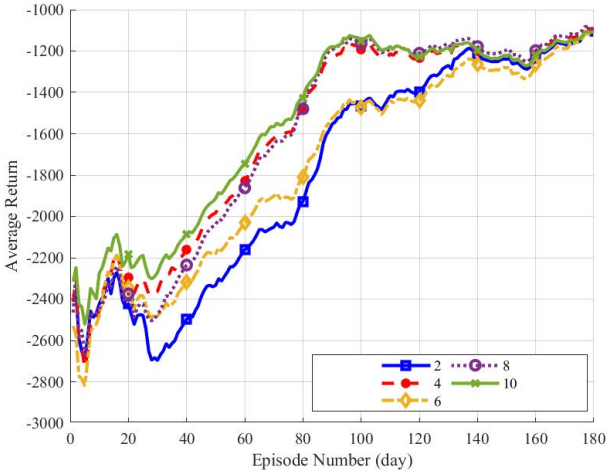

FIGURE 6. DQN agent's performance for different random seeds. Its convergence to an optimal policy is susceptible to the seed used to initialize the weight of its critic network.

For the network architecture of the two agents, we varied the first three hidden layers from 32 to 128, whereas the remaining layers ranged from 128 to 512. Table 1 shows the final average return over the last 60 evaluation episodes and the standard errors across the five trials for different hyperparameter configurations. Using the best configuration set for each agent, we ran another ten trials with learning rates ranging from 0.05 to 0.0005 and five random seeds. The corresponding learning curves of the DQN agent are shown in Figs. 5 and 6. The best-performing DQN and dueling DQN agents with learning rates of 0.05 and 0.01, respectively and random seed equals to ten were selected for the next evaluation step on the remaining clusters.

5) BASELINE ALGORITHMS

The performance of the RL-based scheme was compared with the benchmark produced by the value iteration algorithm and the two baselines provided by the one-step look-ahead

TABLE 1. Effects of batch size and network architecture.

Hyperparameter	Value	DQN	Dueling DQN
Batch size	128	2701 ± 1089	2587 ± 233
	256	1549 ± 457	2588 ± 234
	512	1857 ± 364	2588 ± 234
Network architecture	32 and 128	1663 ± 423	2582 ± 231
	64 and 256	1549 ± 457	2587 ± 233
	128 and 512	1896 ± 447	2581 ± 231

(OSLA) [14] and fixed-location algorithms. Similar to RL agents, OSLA analyzes the overall delays and costs in selecting the best location to deploy an MCS VNF. However, it only considers the short-term reward of each state-action pair rather than the expected long-term reward used by the RL agents. The fixed-location algorithm initially selects a random edge cloud for the deployment of the MCS VNF during the creation of the ancillary critical slice and always retains the same location although the total traffic flow changes during the slice reconfiguration phase.

B. OPTIMALITY

1) DELAY MINIMIZATION

We began by evaluating the performance of our scheme in terms of learning an optimal policy through extensive comparisons with the value iteration and baseline algorithms using the simulated input dataset. The traffic pattern of this dataset was stationary because the flow arrival rate and duration of its groups remained the same during the entire simulation period. We fixed the flow arrival rate λ_1 at three events per day while varying the values of λ_2 from six to 30 events for the same period. We set the active flow period to 20 min for all $1/\mu_k$. For our simulations, we first focused on minimizing the overall delay o_d because this is the utmost requirement of a critical slice. Fig. 7 illustrates the average return resulting from each algorithm for different ratios of flow arrival rates between groups 1 and 2. The results show that the value iteration surpasses the Q-learning and DQN agents by 5.4% and 3.5%, respectively, on average. The value iteration converges to the exact optimal policies in this scenario because the MDP environment is finite and the complete information on the MDP model, including the transition function, is available [23]. The Q-learning agent outperforms the OSLA and fixed-location algorithms by 38.65% and 60.60%, respectively, on average. The DQN agent performs slightly better than the Q-learning agent, with an average improvement of 39.94% and 61.56% over the OSLA and fixed-location algorithms, respectively.

The performance gaps between the baselines and our scheme enlarge with the increase in the ratio between λ_1 and λ_2 because the larger the traffic flow rate of a group, the higher its total number of accumulated traffic flows during the majority of the time. Our scheme prioritizes the edge cloud 2 in the VNF deployment policy, resulting in lower accumulated communication delays and fewer occurrences of VNF migration tasks. By contrast, the OSLA algorithm performs

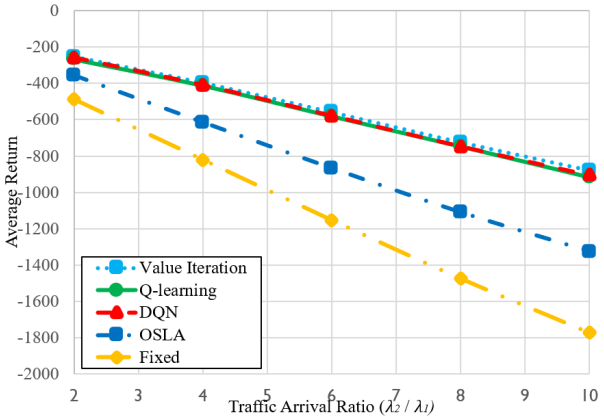

FIGURE 7. Delay minimization with respect to the traffic arrival ratio. The performance of the RL agents is comparable to that based on value iteration. The performance gaps between the baselines and the agents enlarge with the increase in the traffic arrival ratio.

the migration task more frequently because it favors an edge cloud that can provide the best immediate reward at each iteration based on the current system state. The fixed-location algorithm performs the worst because it always prefers a random edge cloud when the total traffic flow is greater than zero. In conclusion, the proposed scheme automatically learns near-optimal policies in dynamic environments with stationary traffic patterns.

2) COST MINIMIZATION

In real-world VNF deployments, it is common for a public safety agency to optimize another objective, such as reducing its overall costs o_c. In this scenario, the flow duration was maintained at 20 min, and the flow arrival rate was fixed at three and 12 flows per day for groups 1 and 2, respectively. The inclusion of the time interval between two consecutive decision processes Δt in the reward calculation transforms the discrete-time VNF deployment problem into a continuous problem, known as a semi-MDP. Thus, the value iteration algorithm was not used because it requires computationally expensive transformations [38]. Fig. 8 illustrates the average return resulting from each algorithm for different ratios of flow arrival rates between groups 1 and 2. The results show that the Q-learning and DQN agents outperform the fixed-location algorithm by 33.44% and 33.88%, respectively, on average. Their performance is comparable to that of OSLA, which has the advantage of knowing the Δt value one step ahead.

In this scenario, RL agents always prioritize the edge cloud 2 in their VNF deployment policies, except when traffic flows are generated only by the group 1. The OSLA algorithm also favors the edge cloud 2. However, it selects either the central cloud or the current serving cloud because they offer the same immediate reward when the total traffic flows of both groups equalize, return to zero, or exceed a certain limit. Despite the high frequency of migration tasks, the resulting policy is optimal because the migration cost is compensated by low processing and communication costs. In conclusion,

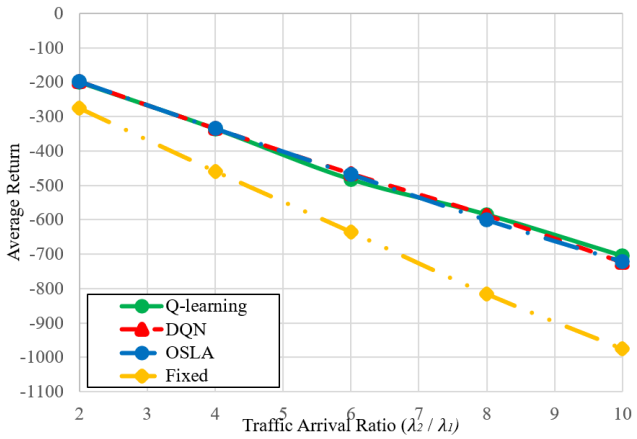

FIGURE 8. Cost minimization with respect to the flow arrival ratio. The performance of the RL agents is comparable to that of the OSLA algorithm. the fixed-location algorithm performs the worst because it prioritizes random edge clouds.

our scheme can automatically learn the best policy for a continuous-time problem by relying only on current observations compared with OSLA, which requires time interval information Δt one step ahead.

C. EXTENSIBILITY

We evaluated the extensibility of our scheme in terms of supporting multiple objectives using the simulated input dataset. In the computation of the expected return R_T, we considered the overall delays o_d and costs o_c. Weight ω represents the trade-off between achieving the two opposing objectives. In our case, minimizing the overall delays when the traffic load is high may cause the agent to favor the central cloud over one of the edge clouds due to the low processing capacity of the latter and the high delay in the migration task. However, this decision may not be the most economical because edge clouds incur low communication costs that are further reduced if the cloud with the most traffic flows is selected. The high communication cost of the central cloud is exacerbated when the network is congested. Therefore, the scheme must determine an optimal policy that balances these objectives in accordance with the different weight configurations.

For our simulations, we retained the best-performing Q-learning and DQN agents in Section V-A and varied weight ω accordingly. We fixed the flow arrival rate λ_1 at three events per day, λ_2 at 12 events for the same period, and an active flow period of 20 min for all $1/\mu_k$. Fig. 9 illustrates the weighted normalized average return resulting from each algorithm for different values of ω. The results show that the Q-learning agent outperforms the OSLA and fixed-location algorithms by 35.55% and 51.79%, respectively, on average. The DQN agent performs slightly better than the Q-learning agent, with an average improvement of 35.80% and 52.16% over the OSLA and fixed-location algorithms, respectively. In conclusion, our scheme can support multiple objectives common in real-world scenarios.

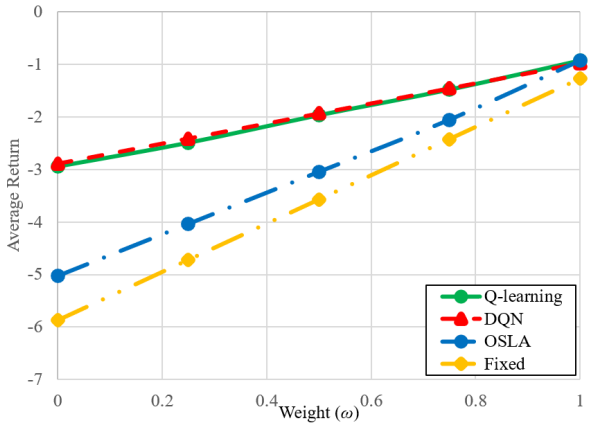

FIGURE 9. Cost and delay minimizations with respect to the weight ω. The performance gaps between the baselines and the agents decrease with the increase in the weight of the overall costs.

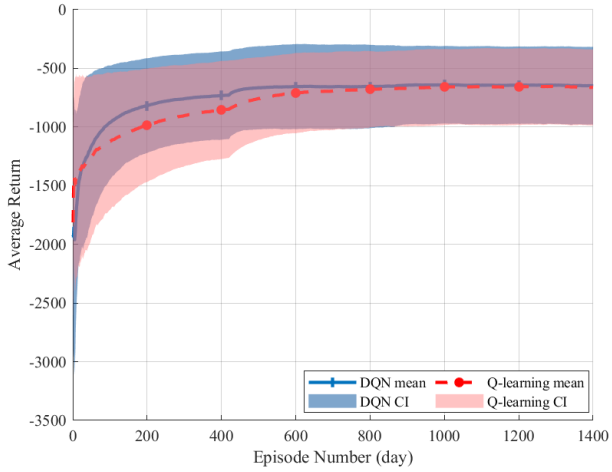

FIGURE 10. Average return of the Q-learning and DQN agents for clusters of two edge clouds. The former converges to an optimal policy at the 470th episode, whereas the latter requires additional 370 episodes to achieve the same target.

D. ADAPTABILITY

We evaluated the adaptability of our scheme to dynamic changes in traffic patterns by using simulated and real-world traffic datasets. We focused on minimizing the overall delays o_d and started by analyzing the convergence rate of our scheme. Fig. 10 illustrates the learning curves of the Q-learning and DQN agents on the simulated input dataset. The average returns of the two agents across the five trials, with 95% confidence intervals, are presented. The results show that the DQN agent outperforms Q-learning by an average of 44%. This result is obtained because Q-learning updates only a single Q-value at each iteration, that is, for the current state action pair that it experiences, whereas DQN updates the critic network parameters for all the state-action pairs in each iteration. The DQN agent is more sample efficient because it can fully exploit its experience through the experience replay technique. Significance testing consisting of Welch's t-test and bootstrap confidence interval test across the entire training distribution yields a p-value of 0.6319 and a bootstrap confidence interval of $[-162, -43]$, respectively.

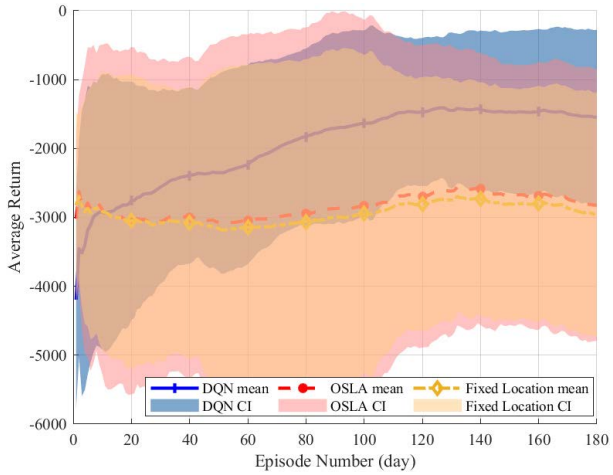

FIGURE 11. Average return of the DQN, OSLA, and fixed-location algorithms for clusters of two edge clouds. The difference between the algorithms' empirical means increases after the 12th episode, indicating that DQN can learn the characteristics of stochastic traffic patterns and subsequently finds an optimal policy.

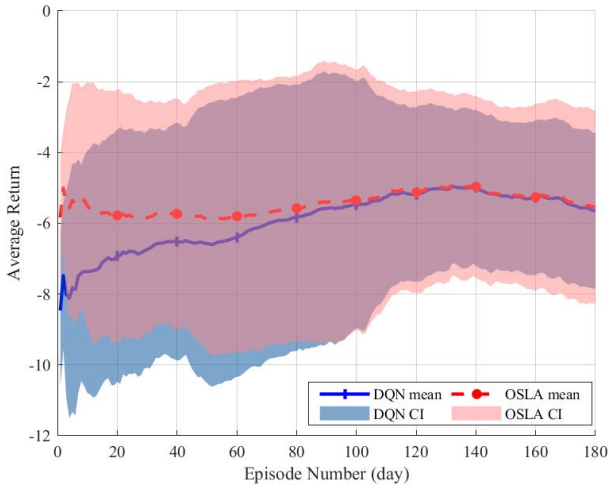

FIGURE 12. Average return of the DQN and OSLA algorithms for clusters of two edge clouds. The empirical means of both algorithms equalize after the 120th episode, indicating that their performance is comparable after the DQN converges to a near-optimal policy.

We then measured the adaptability of our scheme on nonstationary traffic patterns from the real-world Shanghai Telecom dataset. From 274 clusters, we selected five clusters with similar sizes of two edge clouds. By contrast to the simulated input dataset, which represents traffic patterns with static flow arrival rates of λ_k and active periods of $1/\mu_k$, real-world traffic flow may randomly switch between several arrival rates and active periods at different times of the day. We used the best-performing DQN agent in Section V-A for the next evaluation step on the remaining four clusters with different traffic patterns. The value iteration algorithm was not used in this scenario because the environment dynamics, that is, the arrival rate and active period of the traffic flows are unavailable. Thus, the transition function cannot be derived to establish a perfect MDP model that is required by the algorithm [23]. Fig. 11 illustrates the average return of the DQN, OSLA, and fixed-location algorithms across the five trials, with 95% confidence intervals. The results show that the DQN agent finds an optimal policy that prioritizes edge clouds with the most accumulated traffic flows in the long term. The significance testing of the final average return between DQN and OSLA results in a p-value of 0.1766 and a bootstrap confidence interval of [953, 1968], whereas the test between DQN and fixed-location results in a p-value of 0.1142 and a bootstrap confidence interval of [1209, 1958]. In conclusion, the proposed scheme can support dynamic environments with non-stationary traffic patterns at the expense of longer training periods.

The scalability and extensibility of the DQN agent in supporting multiple objectives were evaluated using the Shanghai Telecom dataset. The best-performing DQN agent was retained, and the weight ω was fixed at 0.5, across the five trials. In this case, the agent must balance the overall delays and costs minimizations. The weighted normalized average returns of the DQN and OSLA algorithms across five trials with a 95% confidence interval are shown in Fig. 12. Although OSLA has the advantage of knowing the Δt value one step ahead, the DQN agent can learn these characteristics by successively interacting with the environment. The significance testing between DQN and OSLA results in a p-value of 0.9310 and a bootstrap confidence interval of $[-0.3933, 0.3902]$. In conclusion, the DQN agent can support multiple objectives in a nonstationary environment at the expense of longer convergence time.

E. SCALABILITY

We analyzed the scalability of our scheme to cater to large-scale clusters with many edge clouds while focusing on minimizing the overall delay o_d. For this purpose, we selected two sets of five clusters from the Shanghai Telecom dataset. Each cluster in the first and second sets contains five and ten edge clouds, respectively. We also used a dueling DQN agent in this scenario. Fig. 13 illustrates the average returns with a 95% confidence interval of the DQN, OSLA, and fixed-location algorithms across the five trials for clusters of five edge clouds. The DQN agent improves over OSLA only after the 92nd episode, indicating that its performance decreases with the increase in cluster size because a larger cluster size produces a higher dimension of the state and action spaces. Consequently, the number of state-action pairs increases, thereby requiring prolonged training to determine an optimal policy. The significance testing between DQN and OSLA results in a p-value of 0.2301 and a bootstrap confidence interval of $[-37, 666]$, and the test between DQN and fixed-location results in a p-value of 0.0003 and a bootstrap confidence interval of [12587, 16533].

The VNF deployment policy determined by the DQN agent for the last episode of the second cluster of the five edge clouds is shown in Fig. 14. The DQN agent can learn the environment's dynamics in this scenario. When a single traffic flow is generated by the group 3 at the 459th minute, the DQN agent prioritizes the edge cloud 1 over other locations

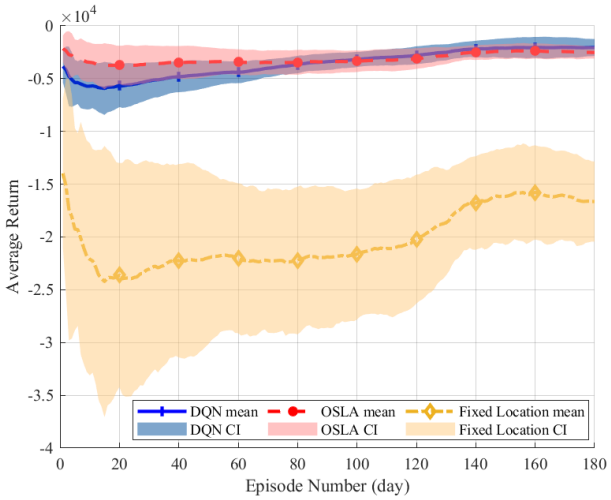

FIGURE 13. Average return of the DQN, OSLA, and fixed-location algorithms for clusters of five edge clouds. The fixed-location algorithm performs the worst because of the high traffic in large clusters, which increases the overall delays if VNF is not deployed in the edge cloud with the most traffic flows.

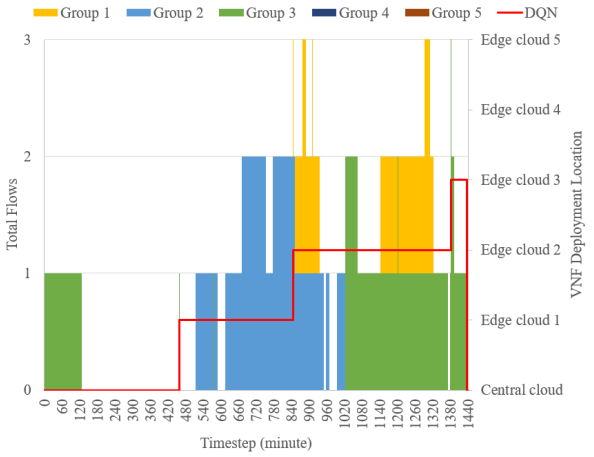

FIGURE 14. VNF deployment policy of the DQN algorithm in a cluster of five edge clouds. The DQN agent always prioritizes the edge clouds 1 and 2 in its VNF deployment policy.

because this action yields the highest Q-value, albeit with a lower processing capacity than the central cloud and a high initial migration cost. This location is retained even when the traffic flows return to zero shortly. The agent migrates the VNF to the edge cloud 2 when its group generates moderate traffic flows for a long period. In the long term, this policy results in lower accumulated communication delays and fewer occurrences of VNF migration tasks due to the heavy traffic flows generated by the group 1 that is, from the 649th to 936th minute and the moderate traffic flows generated intermittently by the group 2 that is, from the 937th to 1060th minute. By contrast, the OSLA algorithm always prefers the central cloud over other locations because it offers the best immediate reward owing to its high processing capacity and exclusion of migration delays.

For the clusters with ten edge clouds, the average returns of the DQN, dueling DQN, and OSLA algorithms across the five trials with a 95% confidence interval are illustrated in

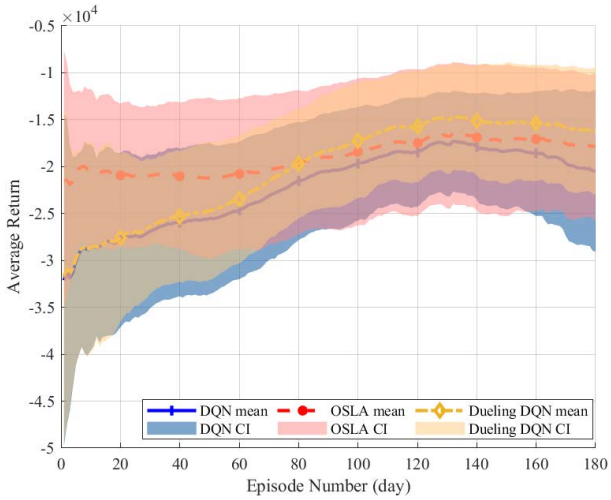

FIGURE 15. Average return of the DQN, dueling DQN and OSLA algorithms for clusters of ten edge clouds. The dueling DQN agent improves over other algorithms after the 85th episode, which demonstrates its performance in finding an optimal policy in an environment where the number of actions is large.

Fig. 15. The DQN agent's performance deteriorates considerably because the higher the number of edge clouds in a cluster, the faster its total traffic flow changes. Consequently, the time interval between two consecutive decision processes is reduced, and the effect of the long-term reward is diminished. The pool of candidates for the serving cloud enlarges with the increase in the number of edge clouds with similar flow arrival rates in a cluster. Consequently, the action yielding the highest Q-value fluctuates among these candidates, thereby affecting the resulting DQN agent policy. The dueling DQN agent performs the best in this scenario. The significance testing between dueling DQN and OSLA results in a p-value of 0.6706 and a bootstrap confidence interval of [266, 3323], and the test between dueling DQN and DQN results in a p-value of 0.3142 and a bootstrap confidence interval of [3144, 6108].

The VNF deployment policy determined by the dueling DQN and OSLA algorithms for the last episode of the second cluster of ten edge clouds is shown in Fig. 16. The dueling DQN agent always prioritizes the corresponding edge clouds with the highest traffic flows in its VNF deployment policy. This result is achieved because of the agent's efficient approach to learn an advantage function that indicates the importance of taking an action in a state, relative to other possible actions. The agent prioritizes the edge cloud 3 most of the time and migrates the VNF several times to the edge cloud 4 when its group generates high traffic flows, that is, from the 419th to 1329th minute. In the long term, this policy results in lower accumulated communication delays and fewer occurrences of VNF migration tasks due to the heavy traffic flows generated by the group 3 that is, from the 626th to 1437th minute.

By contrast, the OSLA algorithm only prefers the edge clouds 3 and 4 during high traffic flows. When the total traffic flow returns to zero, the central cloud is selected because

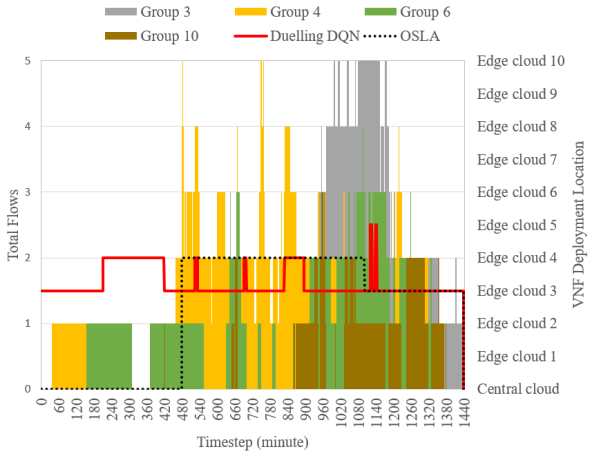

FIGURE 16. VNF deployment policies of the dueling DQN and OSLA algorithms in a cluster of ten edge clouds. The DQN agent always prioritizes the edge clouds 3 and 4 while the OSLA algorithm only prefers the two edge clouds during high traffic flows.

TABLE 2. Computational load in different scenarios.

Cluster Size	DQN	Dueling DQN
2	0.0312 s	0.0156 s
5	0.0516 s	0.0158 s
10	0.0335 s	0.0168 s

it offers the lowest processing and communication delays. This location is retained during low traffic flows because it excludes the high delay in the initial migration task. The DQN agent cannot learn the environment's dynamics in this scenario and always prioritizes the central cloud in its VNF deployment policy. We measured the computational load incurred by each agent to complete one iteration for different scenarios. The results shown in Table 2 were obtained with an Intel Core i5 2.5 GHz CPU platform. The computational times are very low, and dueling DQN outperforms DQN by an average of 56.41%. In conclusion, our scheme leveraging a dueling DQN agent can cater to large-scale scenarios with non-stationary traffic patterns and many suitable deployment locations.

VI. CONCLUSION

Network slicing improves 5G flexibility and scalability in supporting various use cases on common infrastructure, such as critical communications for public safety agencies. However, it introduces new challenges in terms of orchestrating the diverse resources of a logical network, especially within a hybrid cloud environment consisting of central and edge clouds. Orchestration tasks include the deployment of a slice-constituent VNF during the creation and reconfiguration phases of a network slice. This study proposed a deep RL-based scheme to best determine a deployment policy from the perspective of a critical slice that offers group communication services to professional users. To this end, a VNF deployment task was presented, and the problem was formulated as an MDP. Subsequently, a deep RL-based scheme was designed to minimize the overall delays and costs of professional users within the clusters of edge clouds.

The performance of the proposed scheme was evaluated against a dynamic programming-based scheme and baselines by using simulated and real-world traffic datasets. The results show that, on average, the proposed scheme outperforms the OSLA and fixed-location algorithms in terms of the weighted sum of overall delays and costs by 35.80% and 52.16%, respectively, in dynamic environments with stationary traffic patterns. For delay minimization, the integration of a DQN agent enhances the adaptability of the scheme to support non-stationary traffic patterns. Welch's t-test between DQN and OSLA for clusters of five edge clouds results in a p-value of 0.2301, whereas the test between DQN and fixed-location results in a p-value of 0.0003. Further integration with a dueling DQN agent enables the scheme to support large scale environments. Welch's t-test between the dueling DQN and DQN agents for clusters of ten edge clouds results in a p-value of 0.3142. However, deep RL agents require proper hyperparameter tuning, network architecture selection, and random seed definition to ensure optimal performance in real-world scenarios. In our future work, we intend to investigate the application of recent progress in deep RL, which includes state-of-the-art algorithms with high sample efficiency and advanced techniques that provide greater capability for continuous-time problems.

REFERENCES

[1] *Study on Management and Orchestration of Network Slicing for Next Generation Network (Release 15)*, document Rec. TR 28.801 V15.1.0, 3GPP, Jan. 2018.

[2] A. H. Kelechi, M. H. Alsharif, A. M. Ramly, N. F. Abdullah, and R. Nordin, "The four-C framework for high capacity ultra-low latency in 5G networks: A review," *Energies*, vol. 12, no. 18, p. 3449, Sep. 2019.

[3] *Description of Network Slicing Concept V1.0*, NGMN Alliance, Frankfurt, Germany, V1.0, Jan. 2016.

[4] *5G White Paper V1.0*, NGMN Alliance, Frankfurt, Germany, Feb. 2015.

[5] A. Othman and N. A. Nayan, "Public safety mobile broadband system: From shared network to logically dedicated approach leveraging 5G network slicing," *IEEE Syst. J.*, vol. 15, no. 2, pp. 2109–2120, Jun. 2021.

[6] A. Othman and N. A. Nayan, "Efficient admission control and resource allocation mechanisms for public safety communications over 5G network slice," *Telecommun. Syst.*, vol. 72, no. 4, pp. 595–607, Dec. 2019.

[7] A. Jarwan, A. Sabbah, M. Ibnkahla, and O. Issa, "LTE-based public safety networks: A survey," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 2, pp. 1165–1187, 2nd Quart., 2019.

[8] *Technical Specification Group Services and System Aspects; Mission Critical Push to Talk (MCPTT) Over LTE; Stage 1 (Release 13)*, document Rec. TS 22.179 V13.0.0, 3GPP, Dec. 2014.

[9] "A discussion on the use of commercial and dedicated networks for delivering mission critical mobile broadband services," Crit. Commun. Assoc., Newcastle upon Tyne, U.K., White Paper 1.2, Feb. 2017.

[10] R. Solozabal, A. Sanchoyerto, E. Atxutegi, B. Blanco, J. O. Fajardo, and F. Liberal, "Exploitation of mobile edge computing in 5G distributed mission-critical push-to-talk service deployment," *IEEE Access*, vol. 6, pp. 37665–37675, 2018.

[11] D. G. Estevez *et al.*, "Deliverable D4.2: Final design and evaluation of resource elasticity framework version 1.0," 5G-MoNArch, 5G Infrastruct. Public Private Partnership, Heidelberg, Germany, Tech. Rep. D4.2, Apr. 2019.

[12] L. Tang, X. He, P. Zhao, G. Zhao, Y. Zhou, and Q. Chen, "Virtual network function migration based on dynamic resource requirements prediction," *IEEE Access*, vol. 7, pp. 112348–112362, 2019.

[13] Y. Guo, S. Wang, A. Zhou, J. Xu, J. Yuan, and C. Hsu, "User allocation-aware edge cloud placement in mobile edge computing," *Software, Pract. Exper.*, vol. 50, no. 5, pp. 489–502, May 2020.
[14] S. Wang, Y. Guo, N. Zhang, P. Yang, A. Zhou, and X. Shen, "Delay-aware microservice coordination in mobile edge computing: A reinforcement learning approach," *IEEE Trans. Mobile Comput.*, vol. 20, no. 3, pp. 939–951, Mar. 2021.
[15] Z. Luo, C. Wu, Z. Li, and W. Zhou, "Scaling geo-distributed network function chains: A prediction and learning framework," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 8, pp. 1838–1850, Aug. 2019.
[16] S. Schneider, A. Manzoor, H. Qarawlus, R. Schellenberg, H. Karl, R. Khalili, and A. Hecker, "Self-driving network and service coordination using deep reinforcement learning," in *Proc. 16th Int. Conf. Netw. Service Manage. (CNSM)*, Nov. 2020, pp. 1–9.
[17] L. Nadeem, M. A. Azam, Y. Amin, M. A. Al-Ghamdi, K. K. Chai, M. F. N. Khan, and M. A. Khan, "Integration of D2D, network slicing, and MEC in 5G cellular networks: Survey and challenges," *IEEE Access*, vol. 9, pp. 37590–37612, 2021.
[18] A. M. Ramly, N. F. Abdullah, and R. Nordin, "Cross-layer design and performance analysis for ultra-reliable factory of the future based on 5G mobile networks," *IEEE Access*, vol. 9, pp. 68161–68175, 2021.
[19] A. Yousafzai, I. Yaqoob, M. Imran, A. Gani, and R. M. Noor, "Process migration-based computational offloading framework for IoT-supported mobile edge/cloud computing," *IEEE Internet Things J.*, vol. 7, no. 5, pp. 4171–4182, May 2020.
[20] E. S. Ali, M. K. Hasan, R. Hassan, R. A. Saeed, M. B. Hassan, S. Islam, N. S. Nafi, and S. Bevinakoppa, "Machine learning technologies for secure vehicular communication in internet of vehicles: Recent advances and applications," *Secur. Commun. Netw.*, vol. 2021, pp. 1–23, Mar. 2021.
[21] G. Fodor, S. Parkvall, S. Sorrentino, P. Wallentin, Q. Lu, and N. Brahmi, "Device-to-device communications for national security and public safety," *IEEE Access*, vol. 2, pp. 1510–1520, 2014.
[22] A. Sanchoyerto, R. Solozabal, B. Blanco, and F. Liberal, "Analysis of the impact of the evolution toward 5G architectures on mission critical push-to-talk services," *IEEE Access*, vol. 7, pp. 115052–115061, 2019.
[23] R. S. Sutton and A. G. Barto, "Reinforcement learning: An introduction," in *A Bradford Book*. Cambridge, MA, USA: MIT Press, 2018.
[24] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield "Live migration of virtual machines," in *Proc. Conf. Sym. Netw. Syst. Design Implement.*, vol. 2, 2005, pp. 273–286.
[25] C. J. Watkins and P. Dayan, "Technical note: Q-learning," *Mach. Learn.*, vol. 8, nos. 3–4, pp. 279–292, 1992.
[26] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 33–529, 2015.
[27] S. Zhang, H. Yao, and S. Whiteson, "Breaking the deadly triad with a target network," in *Proc. 38th Int. Conf. Mach. Learn.*, 2021, pp. 12621–12631.
[28] L. Baird, "Residual algorithms: Reinforcement learning with function approximation," in *Machine Learning Proceedings*. A. Prieditis and S. Russell, Eds. San Francisco, CA, USA: Morgan Kaufmann, 1995, pp. 30–37.
[29] Z. T. Wang and M. Ueda, "Convergent and efficient deep Q network algorithm," 2021, *arXiv:2106.15419*.
[30] A. Ansari and A. A. Bakar, "A comparative study of three artificial intelligence techniques: Genetic algorithm, neural network, and fuzzy logic, on scheduling problem," in *Proc. 4th Int. Conf. Artif. Intell. With Appl. Eng. Technol.*, Dec. 2014, pp. 31–36.
[31] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, "Dueling network architectures for deep reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 1995–2003.
[32] R. Islam, P. Henderson, M. Gomrokchi, and D. Precup, "Reproducibility of benchmarked deep reinforcement learning tasks for continuous control," 2017, *arXiv:1708.04133*.
[33] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," 2017, *arXiv:1709.06560*.
[34] C. Colas, O. Sigaud, and P.-Y. Oudeyer, "How many random seeds? Statistical power analysis in deep reinforcement learning experiments," 2018, *arXiv:1806.08295*.
[35] S. Wang, Y. Zhao, J. Xu, J. Yuan, and C.-H. Hsu, "Edge server placement in mobile edge computing," *J. Parallel Distrib. Comput.*, vol. 127, pp. 160–168, May 2019.
[36] S. Wang, Y. Zhao, L. Huang, J. Xu, and C.-H. Hsu, "QoS prediction for service recommendations in mobile edge computing," *J. Parallel Distrib. Comput.*, vol. 127, pp. 134–144, May 2019.
[37] G. Dulac-Arnold, D. Mankowitz, and T. Hester, "Challenges of real-world reinforcement learning," *Mach. Learn.*, vol. 110, no. 9, pp. 2419–2468, 2021.
[38] A. Gosavi, "Relative value iteration for average reward semi-Markov control via simulation," in *Proc. Winter Simulations Conf. (WSC)*, Dec. 2013, pp. 623–630.

ANUAR OTHMAN (Member, IEEE) received the B.Sc. degree from the University of Rouen, France, in 2001, and the M.Sc. degree from the University of Strathclyde, U.K., in 2008. He is currently pursuing the Ph.D. degree with the National University of Malaysia, with a focus on critical communications services in 5G. He has over 15 years of experience in telecommunications projects for mobile operators and public safety agencies. His research interests include convergence of narrowband and broadband radio communications for business and mission critical markets.

NAZRUL A. NAYAN (Member, IEEE) received the B.E. degree in information and communication engineering from The University of Tokyo, in 1998, and the M.E. degree in electrical and electronics, and the Ph.D. degree in electronics and information systems engineering from Gifu University, Japan, in 2008 and 2011, respectively. He was a Design Engineer at Hitachi Ltd., Tochigi, Japan, from 1998 to 2000, a Test Engineer at Unisem Malaysia Berhad from 2001 to 2003, a Senior Research and Development Engineer at STATSChipPAC Malaysia, from 2003 to 2005, and an Academic Researcher at Mimos Berhad, Kuala Lumpur, in 2012. He was a Postdoctoral Researcher at the Institute of Biomedical Engineering, University of Oxford, U.K., from 2014 to 2016. He was a Research Member of common room, Kellogg College, University of Oxford, from 2015 to 2016. He has published 27 research articles in international journals and has also presented the papers at 21 international conferences from 29 proceeding papers that he has published. He has received several awards, such as the Japan's Monbukagakusho Scholarship Awards, from 2005 to 2011, the Student Research Award, IEICE, Tokai Division, Japan, in 2010, the Academic Excellence Award, Gifu University, in 2011, and the Excellent Service Award, UKM, in 2014 and 2018.

SITI N. H. S. ABDULLAH received the degree in computing from the University of Manchester Institute of Science and Technology, U.K., the master's degree in artificial intelligence from Universiti Kebangsaan Malaysia, and the Ph.D. degree in computer vision from the Faculty of Electrical Engineering, Universiti Teknologi Malaysia. She was involved in conducting national and international activities, such as Royal Police Malaysia, Cybersecurity Malaysia, Cyber Security Academia Malaysia, Federation of International Robot Soccer Association (FIRA), Asian Foundation, Global Ace Professional Certification Scheme, MIAMI, MACE and IDB Alumni. She was the Chairperson of the Center for Cyber Security in from 2017 to 2020. She is the Deputy Dean of Research and Innovation with the Faculty of Information Science and Technology, Universiti Kebangsaan Malaysia. She has published four books *Pattern Recognition*, *Computational Intelligence in Data Science Application*, *Smart Prediction of Suspect's Serial Crime Location*, and *Optical Script Reader: Texture Binary Innovation* and more than 50 journal articles and 100 conference papers. Her research interests include digital forensics, pattern recognition, and computer vision surveillance system.