To Deploy New or to Deploy More?: An Online SFC Deployment Scheme at Network Edge

Zongyang Yuan, Lailong Luo, Deke Guo, *Senior Member, IEEE*, Denis C.-K. Wong, Geyao Cheng, Bangbang Ren, and Qianzhen Zhang

***Abstract*—Service function chaining (SFC) dynamically links multiple virtual network functions (VNFs) to provide flexible and scalable network services for network entities and users. Implementing SFCs at the network edge provides instant VNF service yet is confined by the limited edge resources. Existing strategies suggest either to *deploy new* VNFs for diverse service provision or to *deploy more* installed VNFs for reliable service provision. However, these one-sided optimizations fail to realize comprehensive improvements in the network service quality. To this end, the motivation of this article is to consider a more comprehensive SFC deployment plan to provide more efficient network services. In this article, we propose DeepSFC, an online SFC deployment scheme at network edge. Our DeepSFC considers the impact of resource allocations and deployment locations on the average latency of overall service requests. It realizes an elegant tradeoff between the diversity and the availability of SFCs by adopting the deep reinforcement learning (DRL) method. To be specific, we first determine the type and number of VNFs that need to be deployed. Thereafter, we optimize the deployment locations of these chosen VNFs in the service chain, considering the impact of dynamic bandwidth in the real network. For more general scenarios wherein users' service requirements change or the deployed server crashes, we further relocate the VNF deployment with the joint consideration of performance degradation and migration cost. Evaluation results show that DeepSFC outperforms its competitors in various experimental settings and responds the requests with lower average latency.**

***Index Terms*—Deep reinforcement learning (DRL), edge computing, online service function chaining (SFC) deployment.**

I. Introduction

With the rapid development of 5G technology [1], [2], [3], its characteristics of low latency and high bandwidth have enabled the proliferation of applications, such as Smart City, Smart Factory, and the Internet of Vehicles [4], [5], [6]. These typical Internet of Things (IoT) applications have distinct or even contradictory network requirements. To adapt to these heterogeneous demands in various scenarios [7], network function virtualization (NFV) has been introduced [8], [9]. NFC may make IoT resource allocation more flexible, providing great scalability and autonomy for IoT service development and deployment [10], [11]. One potential application of NFV is service function chaining (SFC) [12], which dynamically links multiple virtual network functions (VNFs), e.g., name service, firewall, deep packet inspection, etc. Then, SFC could provide secure, fast, and stable network services for IoT applications.

Besides, multiaccess edge computing (MEC) technology has been widely employed [13], [14] for SFC. In MEC, VNF-related service requests could be responded at edge with lower latency than the cloud computing model [16], [17], which greatly improves the Quality of Service (QoS) of IoT applications [15].

There are two mainstream trends for implementing SFCs at network edge. The first category considers the dynamic feature of user needs and the limitation of edge resources. Drastic changes in user requirements may make the SFC deployment solution useless, affecting the service quality significantly. Then, there is a broad emphasis on functional diversity. This part of relevant designs fully uses limited resources to deploy new SFCs [23], trying to allow more service requests to be completed at the edge from the perspective of SFC diversity. Basically, these works improve resource utilization by optimizing the deployment location of VNFs [24], [25]. The second category considers the software vulnerability, hardware failure, or management module damage, which may make VNFs unavailable [18]. Unavailability can cause the entire functional chain to fall into a nonfunctional state. To this end, *function availability* is enhanced by *deploying more* VNF backups [19], [20], [21]. Service requests will be directed to backup VNFs for continuous service if VNF fails. Considering the limitation of resource capacity, existing works try to adaptively maintain a feasible number of VNF replicas.

However, these one-sided optimizations fail to realize the comprehensive improvements of the overall performance, which is jointly decided by both the diversity and availability of deployed SFCs. Therefore, with given resource capacity, there is an inherent contradiction between diversity (deploying new) and availability (deploying more). On the one hand,

Manuscript received 24 February 2023; revised 19 May 2023; accepted 27 June 2023. Date of publication 11 July 2023; date of current version 8 January 2024. This work was supported in part by the National Natural Science Foundation of China under Grant 62002378, and in part by the Research Funding of NUDT under Grant ZK20-3. *(Corresponding author: Lailong Luo.)*

Zongyang Yuan, Deke Guo, Geyao Cheng, Bangbang Ren, and Qianzhen Zhang are with the Science and Technology on Information Systems Engineering Laboratory, National University of Defense Technology, Changsha 410073, Hunan, China (e-mail: yuanzongyang@nudt.edu.cn; guodeke@gmail.com; chenggeyao13@nudt.edu.cn; renbangbang11@nudt.edu.cn; zhangqianzhen18@nudt.edu.cn).

Lailong Luo is with the Science and Technology on Information Systems Engineering Laboratory and the National Laboratory for Parallel and Distributed Processing, National University of Defense Technology, Changsha 410073, Hunan, China (e-mail: luolailong09@nudt.edu.cn).

Denis C.-K. Wong is with the Lee Kong Chian Faculty of Engineering and Science, Universiti Tunku Abdul Rahman, Kampar 31900, Malaysia (e-mail: deniswong@utar.edu.my).

Digital Object Identifier 10.1109/JIOT.2023.3293817

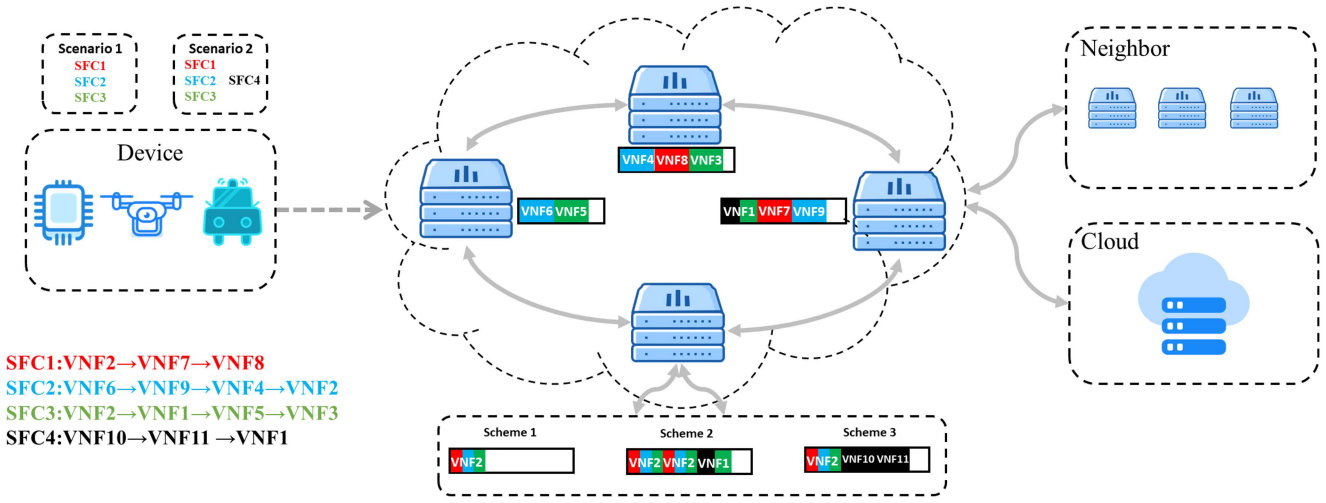

Fig. 1. Illustrative example of deploying SFCs on edge.

maintaining sufficient VNF backups can improve the availability, but may crowd out the space for new functions. On the other hand, if we pursuit of the function diversity, then some VNF backups could not be stored at edge. This may lead to the deployed functions unreliable, then, the service requests cannot be corresponded at the nearby edge. In addition, due to the dynamic bandwidth heterogeneity of different servers, the deployment location of the VNF on the chain has a nonnegligible impact on the delay (i.e., transmission delay, queuing delay, startup delay, etc.).

To meet the above needs of VNF-related service requests, in this article, we propose DeepSFC, an online SFC adaptive deployment scheme in edge computing networks. DeepSFC balances the resource allocation between SFC diversity and availability in an edge environment with limited resources, realizing an elegant tradeoff between these two rationales. DeepSFC adopts a deep reinforcement learning (DRL) scheme to improve overall user service quality. Specifically, DeepSFC monitors the current network link utilization rate and server resource consumption rate in real time, and rewards lower service average completion latency. Finally, DeepSFC chooses the SFC deployment scheme with the most considerable reward on the edge server.

For the more general case of users' service requirements changing or deployed servers crashing, the updated requirements could cause performance degradation for the original static SFC deployment scheme. In this case, we consider redeploying SFC with less resource consumption. Then define a satisfaction threshold for redeployment scenarios, which consists of the completion rate and average completion time of user requests. If the current satisfaction degree is below this threshold, DeepSFC will regenerate a deployment scheme according to the new user requirements and network parameters. In view of the enormous redeployment overhead, migration or retention of the same VNF in the previous SFC deployment solution is more prone in DeepSFC, which can reduce the transmission and deployment costs when implementing the new SFC deployment scheme.

The major contributions of this article can be summarized as follows.

1) We model the problem of SFC online deployment in edge computing as a nonlinear optimization problem. By recognizing the limitations of edge server resources and the dynamic characteristics of the edge network environment, we reveal that the existing solutions to the above problems are not fully optimized. Then, we propose DeepSFC, which adopts DRL to realize the tradeoff between SFC diversity and availability under the influence of different VNF deployment locations. Thus, more service requests can be responded to at the edge, with lower average latency.
2) For the more general scenarios of users' service requirements changing or deployed servers crashing, we further propose to relocate the VNF deployment with the joint consideration of performance degradation and redeployment cost.
3) Evaluation results show that DeepSFC outperforms its competitors in various experimental settings and responds the requests with lower average latency. In addition, DeepSFC can well balance the resource allocation of VNF backup under different failure rates, so as to avoid serious SFC unavailability.

The remainder of this article is organized as follows. Section II introduces the background of SFC deployment at the edge and then discusses the motivation of our DeepSFC. Section III presents the system model and the problem assumptions. Section IV proposes the DeepSFC deployment algorithm. Section V details cost-aware redeployment scheme. Thereafter, we conduct experimental evaluations of our algorithm in Section VI. The related work of this article is reported in Section VII. Finally, Section VIII concludes this article.

II. Motivation

In this section, we first introduce several typical SFC deployment scenarios at edge in Section II-A. Thereafter, the impact of different deployment locations is further analyzed in Section II-B. These discussions stimulate our model building.

A. Impact of Different Deployment Scenarios

As shown in Fig. 1, the set of SFCs required by all users can be expressed as: 1) SFC1 (VNF2 → VNF7 → VNF8); 2) SFC2 (VNF6 → VNF9 → VNF4 → VNF2); 3) SFC3 (VNF2 → VNF1 → VNF5 → VNF3); and 4) SFC4 (VNF10 → VNF11 → VNF1). SFC is composed of multiple VNFs sequenced into chains. For instance, SFC1 consists of VNF2, VNF7, and VNF8. The process of deploying SFCs is equivalent to bulk deployment of the VNFs that make up the SFC. Multiple VNF2s in the diagram indicate the presence of backups, and the backups are consistent with VNF2 functionality. Users connect to edge servers, which are capable of deploying VNFs. Due to resource constraints, these edge servers may not be able to deploy all SFCs and their backups. Thus, services not available on the edge server can be sought from the connected cloud or neighboring edge environments. To show the impact of different deployment scenarios, in this section, we discuss two different user requirements in three deployment scenarios.

In the first case, the user requirement for SFC is [SFC1, SFC2, SFC3], SFC1, SFC2, and SFC3 are also deployed in the edge environment, yet no backups. In this case, user requirements for SFC can be satisfied at edge. However, a software vulnerability, hardware failure, or a damaged management module may render the VNF unavailable. Then, some service requests that cannot pass through the entire chain will be severely affected. For instance, when VNF1 fails, the SFC3-related service requests cannot complete the VNF1 task. In this case, it is generally unacceptable for users to wait on the server, as self-recovery usually takes a long time and tends to lead to buffer overflows. Therefore, all users' service requests ready to go to the server where VNF1 is located to complete the task or queued on that server will resort to the cloud or adjacent edge environment to complete the task requirements with higher completion latency.

Next, we discussed the second SFC deployment scheme. Different from the first SFC deployment scheme, we add backups of VNF2 and VNF1 on servers where resources are sufficient. In this deployment scheme, when the original VNF2 fails, service requests can go from the server where the original VNF2 is located to the server where the backup VNF2 is located, so that service requests can pass through the entire SFC at edge. Therefore, compared to the first case of direct access to the cloud or neighboring edge environments, the average latency to complete service requests can be significantly reduced. In brief, ensuring the availability of SFCs facilitates the stable and reliable completion of service requests in the edge environment.

Then, we consider the third case, where user requirements change from SFC1, SFC2, and SFC3 to SFC1, SFC2, SFC3, and SFC4. Since SFC4 is not deployed at edge, the SFC deployment scheme only meets the service requirements of only a subset of users. However, if we use part of the resources used to instantiate the VNF backup to deploy the SFC4 that users need, then, requests for both the original SFC and those that require SFC4 can be done at the edge with lower latency, rather than going to a remote cloud or neighboring edge environment with higher latency. Thus, ensuring the diversity of SFCs facilitates the completion of more service requests in a low-latency edge environment.

Through the above analysis, we know that deploying new SFCs required by users on edge servers or backing up already instantiated SFCs can bring significant benefits. However, due to limited resources, we may not be able to achieve both benefits in different application scenarios, so we need to balance the two and find the SFC deployment scheme with the least latency to users' service requests.

B. Impact of Different Deployment Locations

Despite the SFC deployment scheme, the allocation schemes of the VNFs on the edge server also affects the completion latency of service requests. As shown in Fig. 2(a), the user is connected to node 1 (server 1) and has an SFC-related service request. The value between any two nodes represents the data transmission delay between any pair of servers. When two functions are executed in sequence at the same node, the queuing and startup delays are set as 1 in default. In Fig. 2(a), red b and arrows indicate the sequence of SFC-related tasks passing through the server in Fig. 2(b), blue c and arrows indicate Fig. 2(c). For simplicity, we ignore the value changes caused by the dynamic bandwidth.

Then, we analyze several SFC deployment schemes in the edge environment. Fig. 2(b) and (c) shows two different deployment schemes, respectively, and s1 in the line n1 means the s1 function is deployed at server 1. In Fig. 2(b), the service request arrives at node 1 and completes the first function of the SFC (delay 3). Then, the service request is transferred to node 2 (delay 1) and completes function 2 (delay 3). Finally, the request goes to nodes 3 and 4 in sequence to complete the last two functions, and the total service delay is 18. In the deployment scheme shown in Fig. 2(c), the service request passes through nodes 1, 4, 2, and 3 in sequence along the SFC, and the delay of this process is 21. It is clear that the scheme shown in Fig. 2(b) outperforms shown in Fig. 2(c) in terms of service completion latency. It declares that different deployment locations have different effects on the latency of service requests to complete functions.

In addition, we further consider the case of deploying redundant VNF backups. As shown in Fig. 2(d), when the function of node 2 fails, service requests that have not yet completed node2's function need to spend a delay of 5 to go to node 3, where the redundant backup is deployed. After completing the corresponding function on node 4, the total delay for SFC-related request completion is 19. However, in the scenario where the VNF backup is deployed at node 4 shown in Fig. 2(e), the latency of service requests completing all functions after the failure of node 2 is 18, which further illustrates the importance of VNF deployment location.

Through the above analysis, we find different VNF schemes and application scenarios can significantly affect the completion delay of service requests. Unilateral optimization may not effectively meet users' SFC diversity and availability needs. In addition, considering that the SFC deployment scheme has different effects on each service request, it is difficult to evaluate the deployment scheme by the completion time of one

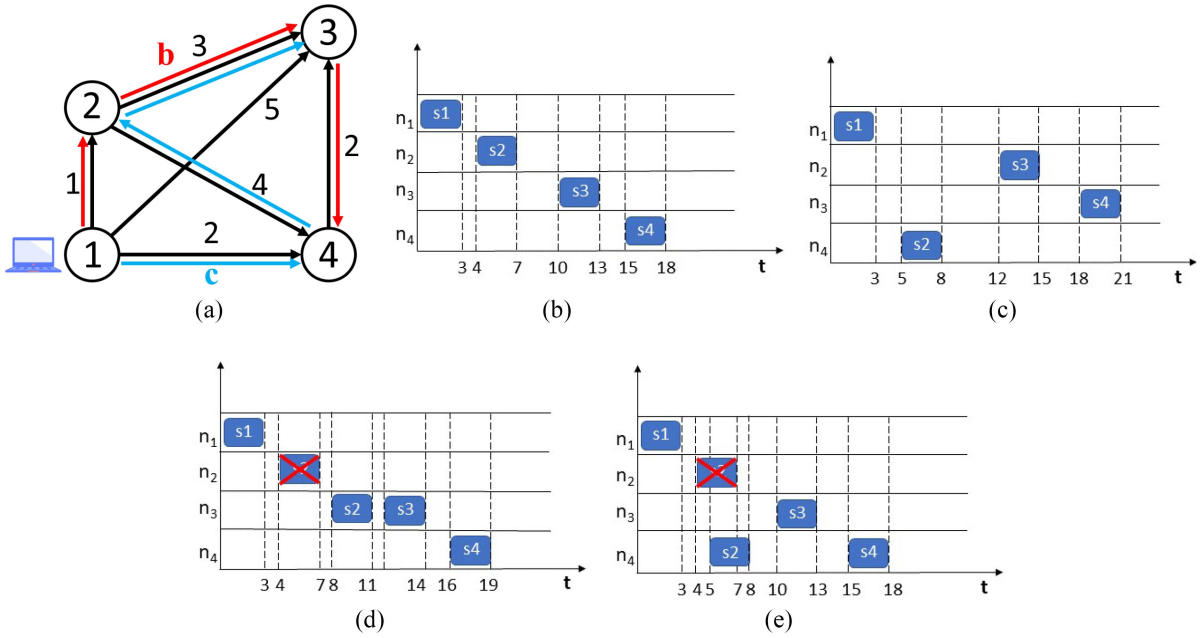

Fig. 2. Impact of different deployment scenarios on service request completion delay. (a) Network topology. (b) SFC deployment scheme 1. (c) SFC deployment scheme 2. (d) SFC deployment scheme 3 with a VNF failure. (f) SFC deployment scheme 4 with a VNF failure.

service request. Therefore, we propose an online SFC deployment scheme considering different deployment schemes and locations, which can comprehensively optimize the average completion delay of service requests. We hope this scheme will play a pioneering role in improving the QoS of service requests.

III. Problem Formulation

In this section, we first state the problem with fundamental assumptions in Section III-A. Thereafter, we analyze the service delay caused by both the SFC deployment schemes and their allocations. Finally, we describe the optimization objective in Section III-D. For clarity, the major notations used in this article are explained in Table I.

A. Problem Statement

We envision the edge network within the appropriate geographical scope as a whole, then assume that there are K edge servers in this edge network, denoted by $\xi = \{E_1, E_2, \ldots, E_K\}$, $E_j (1 \leq j \leq K)$ represents the jth edge server. These edge servers can be placed on the base station to receive service requests from mobile users, or can be placed in server rooms for wired interconnection with users. Compared to the cloud, these servers can provide services with lower latency, since they are placed closer to users. However, the computing and storage capacity of each edge server is limited. Understanding the resource capacity of each edge server is necessary to formulate an SFC deployment scheme in the edge computing environment. We define the computing and storage resource capacity on each edge server E_j as C_j and S_j. We hope to reasonably deploy VNFs on edge servers with limited resources to maximize their utilization efficiency, while considering the integrity of SFC during the deployment process. Note that, if we only deploy a part of the VNFs on a certain SFC, then that SFC will not work properly.

TABLE I
Summary of Notations

Notation	*Description*
K, M	number of edge servers and users, respectively
ξ	the set of edge servers E_j
C_j	computing resources of the edge server E_j
S_j	storage resources of the edge server E_j
FC	the set of SFCs that can be deployed in ξ
ϕ	The set of service requests
J_ϕ	service request completion locations set
h_ϕ	transfer vector between users and first edge servers
$D_R^{i,n}$	resource delay of service requests Q_n^i
$H_{i,n}$	transition matrix when service requests are migrated on edge servers
VNF_j^i	the set of locations where VNFs are deployed on edge servers
$D_P^{i,n}$	placement delay of service requests Q_n^i
$s(t)$	environment state in the edge server at time t
	set from a to b
$a(t)$	action taken by the agent at time t
$r(t)$	reward taken by the agent at time t
$Q, \hat{Q}$	Q-values of the eval net and target net, respectively
$\theta, \bar{\theta}$	parameters of the eval net and target net, respectively
γ	tolerance value to start the reset algorithm
$A(t)$	the ability of current deployment scheme to meet user and operator needs
τ, ι	weighting factor for users needs
σ	weighting factor for operator needs
D_{all}, D_{rate}	metrics for users needs

We assume there are M users in the edge network, denoted as $\nu = \{U_1, U_2, \ldots, U_M\}$. Usually, users can communicate with edge servers and the cloud through the network, and edge servers can communicate with neighbor edge environment and cloud, with service requests typically traveling through multiple links to the edge server. To ease the calculation,

we envisage the minimum bandwidth of the links can satisfy the transmission of the single largest user data. The set of SFCs that can be deployed in a local edge environment is defined as $FC = \{F_1, F_2, \ldots, F_x\}$. Please note that deploying only a portion of the VNF on the chain will make the SFC unavailable. Then, we define the user service request as $\phi = \{Q_1^i, Q_2^i, \ldots, Q_M^i\}$, Q_M^i indicates that user U_M has a demand for F_i.

Typically, if a user's demand for SFC cannot be met at edge, the user's service request will have to access the cloud or neighboring edge computing environment. Therefore, the higher the functional diversity, the more users' service requests can be completed on the edge servers, and the shorter the completion delay of service requests. In addition, the impact of function availability on the completion delay of service requests also cannot be ignored. When the SFC fails and there is no backup, service requests need to go to a cloud or neighboring edge environment with higher latency. Therefore, in order to minimize the average completion delay of all users' service requests, we divide the service request completion delay into two parts: 1) *resource delay* (the delay caused by the type and number of SFCs deployed in the edge computing environment) and 2) *placement delay* (the delay caused by the different locations where SFC is deployed on the edge servers). We model these two delay parts, respectively, and minimize their total delay as the optimization objective in the following sections.

B. Resource Delay

The resource delay is usually caused by the type and number of SFCs deployed on edge servers. We divide the resource delay into two cases: 1) SFCs working normally and 2) SFCs working in failure. When SFCs are working normally, service requests from users reach to the connected edge servers first. To surface the relationship between edge servers and service requests of user U_n, $1 \leq n \leq M$, we define

$$h_\phi = \left[h_{i,n}^1, h_{i,n}^2, \ldots, h_{i,n}^K\right] \tag{1}$$

where $h_{i,n}^K$ is a binary value, with $h_{i,n}^K = 1$ denoting that the service request Q_n^i will be transmitted to the server E_K. And it will be determined on the server E_K whether the service request is completed on the edge side, or to the cloud or the neighbor edge computing environment to complete. $h_{i,n}^k = 0$ otherwise. Therefore, we define

$$J_\phi = \left[J_{i,1}, J_{i,2}, \ldots, J_{i,M}\right] \tag{2}$$

where $J_{i,M}$ is a binary number. When $J_{i,M} = 1$, it means that the service request Q_M^i will go to the cloud or the neighbor edge computing environment with the corresponding function, and $J_{i,M} = 0$ means that the service request will be done in the local edge computing environment.

Considering that the bandwidth of the service request Q_n^i transmitted to the server E_K changes in real time, we define the bandwidth at time t as

$$b^f(t) = \left[b_1^f(t), b_2^f(t), \ldots, b_K^f(t)\right]^T \tag{3}$$

where $b_K^f(t)$ represents the bandwidth (in unit of bit/s) provided by E_K. We define the propagation delay of service requests to the first edge server E_K as P_k, since the users connected to server E_K are usually very close to it. Then, the delay for a service request from the user to the first edge server can be expressed as

$$D_f^{i,n} = P_k + \frac{|Q_n^i|}{h_\phi \circ b^f(t)} \tag{4}$$

where the symbol $\circ$ represents the operation of the inner product, $|Q_n^i|$ indicates the size of the service request Q_n^i. Since the order of service request arrivals is unpredictable, queuing delays, startup delays, and calculating delays are not part of our optimization.

Then, we discuss the delay caused by different destination decisions on the first server. If $J_\phi = 1$, service requests need to go to the cloud or the neighboring edge computing environments to complete tasks. We assume that each edge server is connected to the cloud and can indirectly connect to the neighboring edge computing environments through the edge network. Considering that neighboring edge computing environments usually have a large delay to arrival, we treat this part of the delay as the same as going to the cloud. We define

$$b^c(t) = \left[b_1^c(t), b_2^c(t), \ldots, b_k^c(t)\right]^T \tag{5}$$

where $b_K^c(t)$ represents the network bandwidth (in a unit of bit) from the server E_K to the cloud at time t, we define the propagation delay as P_k^c. Then, the delay of service request Q_n^i from edge servers to cloud or neighboring edge computing environments can be expressed as

$$D_c^{i,n} = \left(P_k^c + \frac{|Q_n^i|}{J_\phi \circ b^c(t)}\right). \tag{6}$$

When $J_\phi = 0$, users' service requests can be completed in the local edge computing environment. If SFCs on the edge servers work normally, users' service requests can be fulfilled at the edge without a trip to the cloud, so $D_c = 0$.

When SFCs fail to work, we discuss two cases for the failure of a VNF or a part of VNFs in one of the SFCs.

1) There is no backup of failed VNFs in the local edge computing environment. Since there is no backup of failed VNFs, waiting for the repair of failed VNFs in place is often unacceptable for service requests. At this time, service requests need to go to the cloud or neighboring edge computing environments to complete. This part of the delay can be expressed as (6) express.
2) There are backups of failed VNFs. In this case, service requests will go from the server where the failed VNF is located to the server where the backup VNF is located. This part of the delay will be discussed in detail in the next section.

Further, if the backup VNF does not fail, the user request can go to the server where the next VNF on the SFC is located, or complete all tasks and return to the user side. When the backup VNF also fails (usually rarely happens), the service request will repeat the above process if there are still available VNF backups. If both the VNF and its backup fail, the

service request will access to the cloud or a neighboring edge computing environment to complete the remaining tasks.

Then, we consider the failure probability of VNFs. Since VNF failure is usually unpredictable, we use the maintenance log of VNFs as a heuristic item, and define the failure rate of each VNF as: $\mathrm{FR_{VNF}}$. When we initially deployed SFC in the edge computing environment, there was no historical maintenance log as a reference, so we empirically referenced the historical maintenance logs of other edge servers to define the initial failure rate of the VNF. Finally, the model for resource delay can be expressed as

$$D_R^{i,n} = D_f^{i,n} + J_\phi D_c^{i,n} + \left(!J_\phi\right)\mathrm{FR_{VNF}} D_c^{i,n}. \tag{7}$$

Due to the resources of each edge server being limited, for VNFs deployed on edge servers, the following constraint should hold:

$$\forall \sum_{n=1}^{M} |C_j^{F_i}| < C_j \tag{8}$$

with $|C_j^{F_i}|$ denoting the amount of computing resources that F_i deploys on server E_j. When Q_i^n is not deployed on server E_j, $|C_j^{F_i}|$=0

$$\forall \sum_{n=1}^{M} |S_j^{F_i}| < S_j \tag{9}$$

with $|S_j^{F_i}|$ denoting the number of storage resources that F_i deploys on server E_j. Due to guarantee the functional integrity of VNF, each VNF can only be mapped to one server

$$\mathrm{VNF}_j^i = \begin{cases} 1, & \text{if no VNF backup exists} \\ \text{Integer value} > 1, & \text{if VNF backups exist} \end{cases} \tag{10}$$

with VNF_j^i denoting a VNF on the F_i chain deployed on server E_j. Due to ensure the integrity of SFC, all VNFs on SFC must be mapped to edge servers

$$\forall \sum_{j=1}^{K} VNF_j^i = F_i. \tag{11}$$

C. Placement Delay

After determining the type and quantity of SFCs deployed at the edge, we also need to consider the delay caused by the placement location schemes of SFCs, since the delay caused by different placement location schemes is different. We use the placement delay to judge the quality of a placement location scheme. Note that in the modeling of placement delay, we do not discuss cases where service requests cannot be fulfilled at edge. We divide the placement delay into two parts.

1) The service requests start from the server where the first VNF on the SFC is located, and pass through the delay of the server where the entire SFC is located in order, under normal working conditions.
2) When a failure occurs, the delay caused by service requests going to the server where the backup VNF is located in the edge computing environment. Based on the identified deployment scheme, we then explore the delay generated by the deployment location, we define

$$H_{i,n} = \begin{bmatrix} h_{i,n}^{11} & h_{i,n}^{12} & \cdots & h_{i,n}^{1K} \\ h_{i,n}^{21} & h_{i,n}^{22} & \cdots & h_{i,n}^{2K} \\ \cdots & \cdots & \cdots & \cdots \\ h_{i,n}^{K1} & h_{i,n}^{K2} & \cdots & h_{i,n}^{KK} \end{bmatrix}. \tag{12}$$

$h_{i,n}^{1K}$ is a binary value. With $h_{i,n}^{1K} = 1$ denoting that the service request Q_n^i will go from server E_1 to server E_K. We define this part of the bandwidth at time t as

$$B(t) = \begin{bmatrix} b^{11}(t) & b^{12}(t) & \cdots & b^{1K}(t) \\ b^{21}(t) & b^{22}(t) & \cdots & b^{2K}(t) \\ \cdots & \cdots & \cdots & \cdots \\ b^{K1}(t) & b^{K2}(t) & \cdots & b^{KK}(t) \end{bmatrix}. \tag{13}$$

We define the propagation delay as

$$P(t) = \begin{bmatrix} P^{11}(t) & P^{12}(t) & \cdots & P^{1K}(t) \\ P^{21}(t) & P^{22}(t) & \cdots & P^{2K}(t) \\ \cdots & \cdots & \cdots & \cdots \\ P^{K1}(t) & P^{K2}(t) & \cdots & P^{KK}(t) \end{bmatrix}. \tag{14}$$

Therefore, the delay of this part can be expressed as

$$D_{S_0-S_1}^{i,n} = H_{i,n} \circ P(t) + \frac{|Q_n^i|}{H_{i,n} \circ B(t)}. \tag{15}$$

In particular, with $D_{S_0-S_1} = 0$ denoting that the server where the first VNF is located in the judgment server, so there is no need to transmit across servers.

When the SFC is working normally, after service requests complete the task of the first VNF on the SFC, they will go to the edge servers where the remaining VNFs are located on the SFC according to the routing rules until the entire SFC is completed. Therefore, we define the ith part delay as

$$D_{\mathrm{SFC}}^{i,n} = D_{S_0-S_1}^{i,n} + D_{S_1-S_2}^{i,n} + \cdots + D_{S_{N-1}-S_N}^{i,n}. \tag{16}$$

Similarly, when the previous VNF on the SFC is located on the same edge server as the next VNF, we envision this delay equal to 0. Next, we discuss a special case in which the SFC fails due to a VNF failure (which is difficult to recover from in a short time) in the working situation described above. When there is a backup of the failed VNF, according to the routing rules formulated in advance, service requests need to go to the edge server where the backup VNF is located to complete the task. We define this part of the delay as

$$D_F^{i,n} = H_{i,n}^F \circ P^F + \frac{|Q_n^i|}{H_{i,n}^F \circ B^F(t)}. \tag{17}$$

In particular, if the failed VNF is not the last one on the SFC, user requests also need to go from the server where the failed backup VNF is located to the server where the next VNF of the failed VNF on the original SFC is located to complete the corresponding task. Consider that the backup VNF and the failed original VNF are not guaranteed to be on the same server, we define the delay for transmission from the failed backup server to the next server as

$$D_L^{i,n} = H_{i,n}^L \circ P^L + \frac{|Q_n^i|}{H_{i,n}^L \circ B^L(t)} \tag{18}$$

with $D_L = 0$ denoting the server where the backup VNF is located is the same server where the next VNF is located. If multiple backup VNFs on the SFC fail at the same time, we believe that a major failure has occurred and should be repaired at this time. Therefore, the delay in the simultaneous failure of multiple backup VNFs will not be discussed. The placement delay can finally be expressed as

$$D_P^{i,n} = D_{\text{SFC}}^{i,n} + D_F^{i,n} + D_L^{i,n}. \tag{19}$$

D. Optimization Objective: Minimize the Total Delay

In our scenario, through the above description, the whole process of service request completion can be expressed as: the user sends the service request to the most suitable edge server after a delay $D_f^{i,n}$. Then, the service request is determined on the edge server whether it is done in the local edge computing environment, or to go to the cloud and neighboring edge computing environments after a delay $D_c^{i,n}$. If the service request is completed in the local edge computing environment, the service request will be transmitted from the decision server to the server where the first VNF of the SFC is located after the delay $D_{S_0-S_1}^{i,n}$, in the case that the SFC is working normally. Then, the service request will pass through all the servers mapped by the SFC. If the SFC fails, the service request will go from the server where the failed VNF or the previous VNF of the failed VNF is located to the server where the backup VNF is located after the delay $D_F^{i,n}$, and then go to the server where the next VNF is located after the delay $D_L^{i,n}$, until the entire SFC is completed. Finally, after the service request completes the SFC task in the cloud, neighbor edge computing environment, or local edge environment, it may go to the next destination (i.e., back to the client). Yet this part is not discussed in this article.

In the case of limited resources, there is resource competition between service requests. There will be a lot of contradictions if we aim to optimize the delay of each service request. Therefore, we minimize the average completion delay of all service requests with the following optimization objective:

$$\min \sum_{n=1}^{x} \sum_{i=1}^{K} \left(D_P^{i,n} + D_R^{i,n} \right)$$
$$\text{s.t.} \quad (8) \sim (11). \tag{20}$$

In the above optimization problem, the objective function is to minimize the delay of all service requests, composed by the placement and resource delays ($D_P^{i,n}$ and $D_R^{i,n}$). Constraints set limits on the resources provided by the edge server and the integrity of SFC deployment. In addition, J_ϕ, $H_{i,n}$, $H_{i,n}^F$, and $H_{i,n}^L$ are unknown variables to be solved. After solving this problem, we can obtain an elegant balance between diversity and availability, so as to minimize the completion delay of service requests. When tackling the above optimization problems, however, we are faced with some challenges. First, the number of users and edge servers increases from hundreds to thousands in real-world scenarios. Traditional optimization algorithms need to spend a long time in scheduling to formulate the optimal strategy for deploying SFC on edge servers.

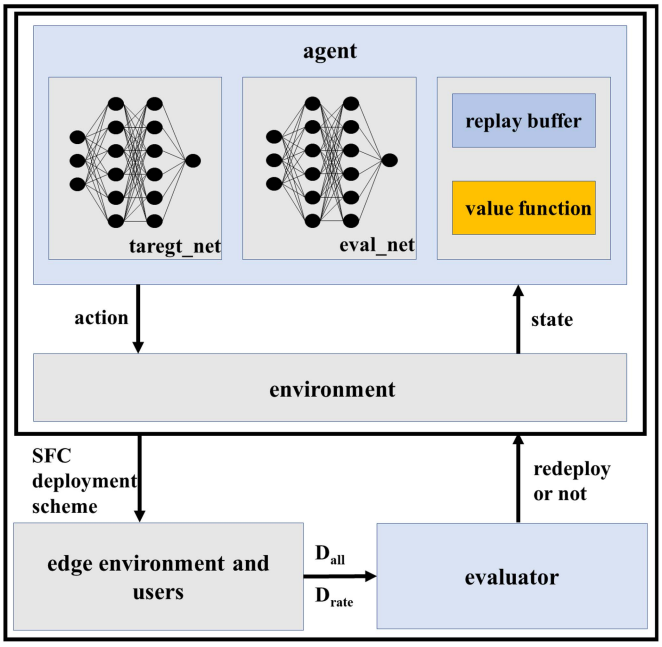

Fig. 3. Framework of DRL-based deployer at edge.

This challenge leads to a huge search space in solving the optimization problem, so any search method based on violence is prohibited. In addition, in our modeling process, network conditions and server load are assumed to be known. However, due to the dynamic nature of user needs and the network, this assumption is not shown in practice. Therefore, the solution of offline optimization may be far from the actual situation. The online SFC deployment scheme for real-time estimation of network conditions and server load should be considered.

To tackle the above challenges, we propose an online SFC deployment scheme DeepSFC, which is based on DQN techniques in the following section.

IV. DeepSFC Deployment Scheme

In this section, we introduce the DRL-based SFC online deployment scheme, which uses the DQN algorithm to minimize the average delay of all service requests described in Section II. The working process of DeepSFC is shown in Fig. 3. We first introduce the components and concepts of the reinforcement learning model in Section IV-A. Thereafter, we introduce the design of the DeepSFC working components in Section IV-B.

A. Components and Concepts

The deployment scheme of SFCs is generated by DQN. DQN usually consists of five parts: 1) state; 2) agent; 3) action; 4) reward; and 5) policy, which will be described in detail below.

State: The state is an important part of DQN and is the data used to represent the environment. In the interaction process between the agent and the environment, different states will be generated. In brief, after performing an action, the environment will be converted into a new state. In the process of deploying SFC, the state can be represented by

$s(t) = \{C_j(t), S_j(t), B(t), UR(t), FR_{\mathrm{VNF}}(t), S_K(t)\}$, in which $C_j(t)$ and $S_j(t)$ represent the remaining computing and storage resources on the edge server E_j at time t, respectively. $B(t)$ indicates the bandwidth of each communication link in the edge network at time t. $UR(t)$ and $FR_{\mathrm{VNF}}(t)$ denote users' requirements and fault VNF information at time t, respectively.

Agent: The agent is composed of an internal computing parameter system and a learning mechanism, which can interact with the environment to obtain information and feedback on the current state. It can evaluate the impact of the previous action on the environment according to the feedback information, and can also select the action with the current maximum reward value to act on the environment according to the current state. During training, the agent will iteratively loop the above process to improve its decision-making ability. The agent's goal in this article is to find an SFC deployment scheme that minimizes the latency of all service requests.

Action: Action is the agent's response to the current state of interacting with the environment, based on parameter information and learning strategies. In this article, the action is to deploy the VNF on the SFC to the edge server. To improve the convergence speed of the algorithm, we deploy multiple VNFs instead of one VNF each time. In this article, action can be represented by a vector $a(t)$. Choosing which actions to deploy is closely related to the current state's resource and SFC constraints. Usually, we choose actions that can bring higher long-term rewards.

B. Design of Working Components of DQN

Reward: For different states, the agent will choose different actions, and the actions acting on the environment will generate new states. The choice of each action will directly or indirectly affect the result of the final scheme. Here, our final scheme is the solution of deploying SFC. To minimize the average delay time of all service requests, the agent will evaluate the action selected by each state, and the result of the evaluation is the reward. When an action is beneficial to our final goal, the agent will give a certain reward to the action, and if it is harmful, it will give some punishment. The agent can update the deployment scheme according to the reward, so it can find a better action in the same state next time. Here, we set the reward as

$$r(t) = \mathbb{E}\big(-\big(D_{all} + \mu D_{s(t),a(t)}\big)\big) \tag{21}$$

where $0 < \mu < 1$ are weights, $\mathbb{E}$ is the value function, D_{all} indicates the overall average completion delay and calculated by

$$D_{\mathrm{all}} = \text{average} \sum_{n=1}^{x} \sum_{i=1}^{K} \Big(D_P^{i,n}(t) + D_R^{i,n}(t)\Big) \tag{22}$$

where average is average value function, $D_P^{i,n}(t) + D_R^{i,n}(t)$ indicates the service request completion delay under the influence of deployment policy in each time slot t. D_{all} indicates a long-term overall return. If the deployer makes a good decision, it can generate a suitable amount so that the average completion delay of all service requests will be low. $D_{s(t),a(t)}$ indicates the impact of the current action, which also means short-term rewards.

Policy: Different policies represent different deployment SFC schemes, which consist of a series of actions made by agents in different states. In DQN, the policy will be continuously updated during the learning process, and the quality of the policy will directly affect the average latency of all service requests.

In traditional Q-learning, after designing the above components, we will initialize a Q table to record the value corresponding to the state-action. If there are m states and n actions, the size of this Q-value table is $m * n$. When querying the Q table, first we determine the current state s, then find the row where the state s is located, and finally output the action with the largest Q value in this row, which is a complete decision-making process. However, if Q-learning is used to determine the deployment plan of SFCs in a scenario with many states and actions, it needs to build a huge Q-value table, which will bring a lot of space overhead. After the scale becomes larger, it takes a lot of time to calculate each value in this table accurately. Moreover, in many practical tasks, some states and actions will never appear, making it difficult to calculate the value function accurately and causes problems of computational cost and accuracy problems.

When training a neural network, the assumption is that the training data is independently distributed. There is always a correlation between the data collected through reinforcement learning, which may cause great fluctuations in the learned value function model. In addition, when the gradient descent method is used to update the model, the model training often needs to go through multiple rounds of iterations to converge. Each iteration needs to use a certain number of samples to calculate the gradient. If each calculated sample is discarded after calculating the gradient once, then, we need to spend more time interacting with the environment and collecting samples. Therefore, we use experience replay to train the learning process of reinforcement learning, and we set a replay buffer for learning previous experiences. So every epoch model updates the parameters, we can randomly sample some previous experiences for learning. Randomly extracting data can disrupt the correlation between the data and make the neural network update more efficient. Our replay buffer will store transitions that occurred in the past, and transitions consist of $s(t)$, $a(t)$, $r(t)$, and $s(t+1)$. We set the size of the replay buffer to $RB = 2000$. When the stored data exceeds RB, the stored transition will overwrite the earliest stored transition in the memory. At each iteration, we randomly extract experience from memory using mini-batches.

In addition, in the iterative process of Q-Learning, the parameters are updated by the reward at the current moment and the estimated reward at the next moment. Due to the instability of the data, each iteration may generate some fluctuations, and these fluctuations will be immediately reflected in the calculation of the next iteration, making it difficult for us to obtain a stable model. To alleviate the impact of related problems, we decouple these two parts and introduce target and eval network. Our goal is to delay the parameter update of the target network, thereby disrupting the correlation. Therefore,

Algorithm 1 Deployment Algorithm of SFC With DRL

Require: Service requests $\phi = \{Q_1^i, Q_2^i, ..., Q_M^i\}$.
Ensure: Quantum a(t) $1 \leq t \leq T$.
1: Initialize replay buffer to capacity RB
2: Initialize eval network Q with random weights θ
3: Initialize target network $\hat{Q}$ with random weights $\bar{\theta} = \theta$
4: **for** episode=1:MaxE **do**
5: **for** t=1:T **do**
6: Get state $s(t)$
7: $$a(t) = \begin{cases} argmaxQ(s(t), a(t), \theta), prob.\epsilon \\ randomaction, prob.1 - \epsilon \end{cases}$$
8: Execute action $a(t)$ and obtain reward $r(t)$ and $s(t+1)$
9: Store transition $(s(t), a(t), r(t), s(t+1))$ in replay buffer
10: Sample random mini-batch of transitions $(s(i), a(i), r(i), s(i+1))$ from replay buffer
11: continue
12: Update parameter θ
13: Every steps set $\hat{Q} = Q$
14: **end for**
15: **end for**

the structure of the target network is exactly the same as that of the eval network.

The Deployment Algorithm for SFCs is summarized in Algorithm 1. The deployer first initializes the replay buffer and internal parameters θ and $\bar{\theta}$ (lines 1–3 in Algorithm 1). After obtaining the current state $s(t)$ of the environment, the Agent uses the method of $\epsilon - greedy$ to select action $a(t)$, that is, the probability of ϵ selects the action with the maximum value of Q, and the probability of $1 - \epsilon$ randomly selects the action that satisfies the constraints (lines 6 and 7 in Algorithm 1). After executing action $a(t)$, the agent can obtain reward $r(t)$ according to the reward scheme, and can also obtain the next environment state $s(t)$. Then store them in the replay buffer in the form of $[s(t), a(t), r(t), s(t+1)]$. After o steps, the Agent randomly selects a small part of the experience in mini-batch from the replay buffer (lines 8–10 in Algorithm 1). After interacting with the environment, the agent uses stochastic gradient descent to update network parameters. At the same time, after t steps, the target network will copy the parameters of the eval network to update the parameters (lines 11–13 in Algorithm 1).

V. Cost-Aware Redeployment Scheme

For the more general scenarios of user's service requirements changing or deployed servers crashing, the updated requirements may cause performance degradation for the original static SFC deployment scheme. Therefore, it is necessary to regenerate a deployment scheme for the updated requirements. Note that, the regeneration for any slight changes is costly, due to the redeployment requiring a significant amount of VNFs migrations. In this case, we introduce a cost-aware redeployment scheme in this section. Our redeployment takes the migration or redeployment cost of the SFC as the redeployment index, and tries to minimize these cost.

We use $A(t)$ to represent the satisfaction of users and operators with the current SFC deployment scheme as follows:

$$A(t) = \sigma(\tau D_{\text{all}} + \iota D_{\text{rate}}) \tag{23}$$

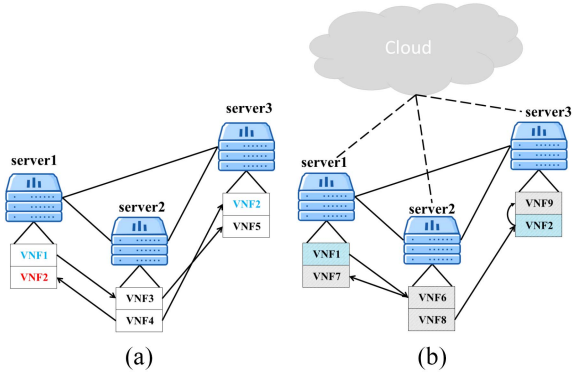

Fig. 4. Framework for cost-aware SFC redeployment. (a) Original SFC deployment scheme. (b) Newly generated SFC deployment scheme.

where $0 < \sigma < 1$ represents the operator weight, D_{all} represents the average completion delay of service requests, and D_{rate} represents the percentage of service requests completed at the edge. After being normalized and weighted by τ and ι, respectively, $A(t)$ can be the capability metric to indicate the users' satisfaction degree of the SFC deployment scheme.

To avoid frequent redeployment, we propose a tolerance threshold γ. Only when the current satisfaction $A(t)$ below this threshold γ, the SFC redeployment can be triggered, then, the agent formulates a new SFC deployment scheme according to the server's status and new user's requirements.

After the agent interacts with the new environment to generate a new deployment scheme, the new SFCs need to be remapped to the edge server. However, converting the two SFC deployment schemes in the edge environment may be costly. As shown in Fig. 4, Fig. 4(a) represents the original SFC deployment scheme, and Fig. 4(b) illustrates the new SFC deployment scheme. The blue VNFs in Fig. 4(a) represents the duplicated part of the original SFC deployment scheme and the newly generated SFC deployment scheme. The blue-crossed VNF in Fig. 4(b) represents the portion retained from the old deployment scenario, and the gray crossed VNF in Fig. 4(b) indicates the portion mapped from the cloud. Typically, during the conversion between the two SFC deployment solutions, the agent needs to remap VNFs from the cloud to the edge server. To be more intelligible, we assume that the cost of remapping a VNF is 1, and the remapping cost of the new deployment scheme shown in Fig. 4(b) is 6. However, we can notice the blue VNF1 and VNF2 are deployed on the same server in these two SFC deployment scenarios in Fig. 4(a) and (b), separately. Therefore, if we retain this part of VNFs and remap the rest during the remapping process, the cost of the new deployment scheme can be reduced to 4. Furthermore, except retaining VNFs during the conversion process, the VNF migration between neighbor servers could also decrease the redeployment cost, because it avoids the VNF remapping from the cloud to the edge servers. For instance, the red VNF2 in Fig. 4(a) is migrated from servers 1 to 3. Therefore, we retain or migrate VNFs in the old deployment scheme to the new one. This can result in lower cost and deployment delay for the operator than backup from the cloud.

Algorithm 2 Redeployment Algorithm Considering Cost

Require: Old deployment scheme ODS.
Ensure: The new redeployment policy P'.
1: **if** $A(t) < \gamma$ **then**
2: Execute algorithm. 1 to get $a(t)$
3: Convert $a(t)$ into the new set $N_{VNF} = \left\{VNF_1^j, VNF_2^j, \ldots, VNF_n^j\right\}$ of VNFs to be deployed
4: **for** $v = 1{:}n$ **do**
5: **if** VNF_v in ODS **then**
6: **if** VNF_v^j in ODS **then**
7: Set the value of P'_v with E_j, i.e., $P'_1 = E_j$
8: **else if** VNF_v^j not in ODS **then**
9: Find the server S where the VNF_v and its backup are located
10: Find the closest server E_s to j in A
11: Set the value of P'_v with $E_s s$, i.e., $P'_1 = E_s$
12: **end if**
13: **else**
14: Set the value of P'_v with cloud, i.e., P'_1 =cloud
15: **end if**
16: **end for**
17: **end if**

The pseudocode in Algorithm 2 describes the cost-aware redeployment algorithm. First, the algorithm will determine whether to trigger the redeployment. When the trigger condition is satisfied, the agent will formulate a new SFC deployment scheme (lines 1–3 in Algorithm 2). For the VNFs in the new deployment scheme, the agent will determine which ones need to be remapped from the cloud, which ones need to be remigrated from edge servers, and which ones are already on the servers that need to be deployed and do not need to be remapped (lines 5–13 in Algorithm 2).

VI. Performance Evaluation

In this section, we first introduce the experiment setting in Section VI-A. Thereafter, we evaluate the performance of DeepSFC in numerical and emulation experiments in Section VI-B.

A. Simulation Setup

We refer to the basic setup of a VNF platform on a commercial server. Combined with the actual situation, we believe each edge server has only one six-core CPU. Since there may be other tasks besides VNFs on the edge server, we assume that each edge server randomly has 1–6 cores. We assume that there are ten edge servers in the edge environment, each server is connected to the cloud and can connect to the servers in the neighbor edge environment through the edge network. In this experiment, we use several laptops as edge servers (OS: 64-bit Ubuntu 16.04.7; RAM: 16 GB DDR4 2666 MHz; CPU: Intel i7-9750H 2.6 GHz). We choose the basic settings of SFC in the real environment to formulate the number and connection order of VNFs. Due to the mobility of users, it is difficult to predict the arrival of service requests accurately. Therefore, we assume that users connect to edge servers uniformly and randomly, and each user can only generate at most one SFC service request per unit time.

In order to better evaluate the performance of DeepSFC, we conducted the comparison experiments with the following two types of baseline algorithms.

1) Deployment Scheme Baseline: To show the performance of our DeepSFC scheme, we compare it with four baselines.

1) *Random:* According to the set of VNFs that can be deployed, randomly deploy VNFs on edge servers, with the respect of resource constraints.
2) *Greedy:* In a resource-constrained edge environment, deploy as many VNFs as possible at edge servers according to user needs. It tends to find a deployment scheme that minimizes the waste of server resources, while satisfying resource constraints.
3) *Greedy-Backup:* Unlike Greedy, it takes into account of the VNF availability. Therefore, during the deployment process, some redundant backups will be deployed according to the failure of the VNFs [44].
4) *Central-Deploy:* In the deployment process, Central-Deploy tends to deploy SFC on edge servers closer to users centrally, so as to reduces the delay of the user request as much as possible. The drawback is that the VNF backups are not considered [26].

2) Relocate Baseline: To show the performance of our relocate scheme, we compare it with *Cloud* baseline relocate schemes. After determining the new deployment scheme, the *Cloud* method downloads all the required VNFs from the cloud, maps them to the server, and then serializes them into SFCs.

B. Numerical Results

In this section, we demonstrate the performance of our model and the baseline in terms of task efficiency, and proportion of completion on the edge, through evaluation results.

Considering the diversity of actual deployment scenarios, as shown in Fig. 5, we selected four representative scenarios for experiments. Lack means that the resources on the edge server are very scarce, and deployer cannot deploy all SFCs that the current users need. We quantify Lack as 80% of SFC deployment resources required by users. Middle implies that the resources on the edge server can deploy the SFCs required by users, but may not be able to deploy the VNF backups. We quantify Middle as 100% of SFC deployment resources required by users and 60% of SFC backup deployment resources. Boundary represents resources on edge servers can deploy all original SFCs and some VNF backups required by current users. We quantify Boundary as 100% of SFC deployment resources required by users and 90% of SFC backup deployment resources. Enough means that the resources on the edge server are sufficient to deploy all SFCs and VNF backups required by the users. We quantify Enough as 150% of SFC deployment resources required by users and 100% of SFC backup deployment resources.

1) Total Service Request Average Completion Time: Fig. 5(a) shows the evaluation results of the total average completion time of service requests when the VNF failure rate is 0%. We find that DeepSFC always outperforms Greedy,

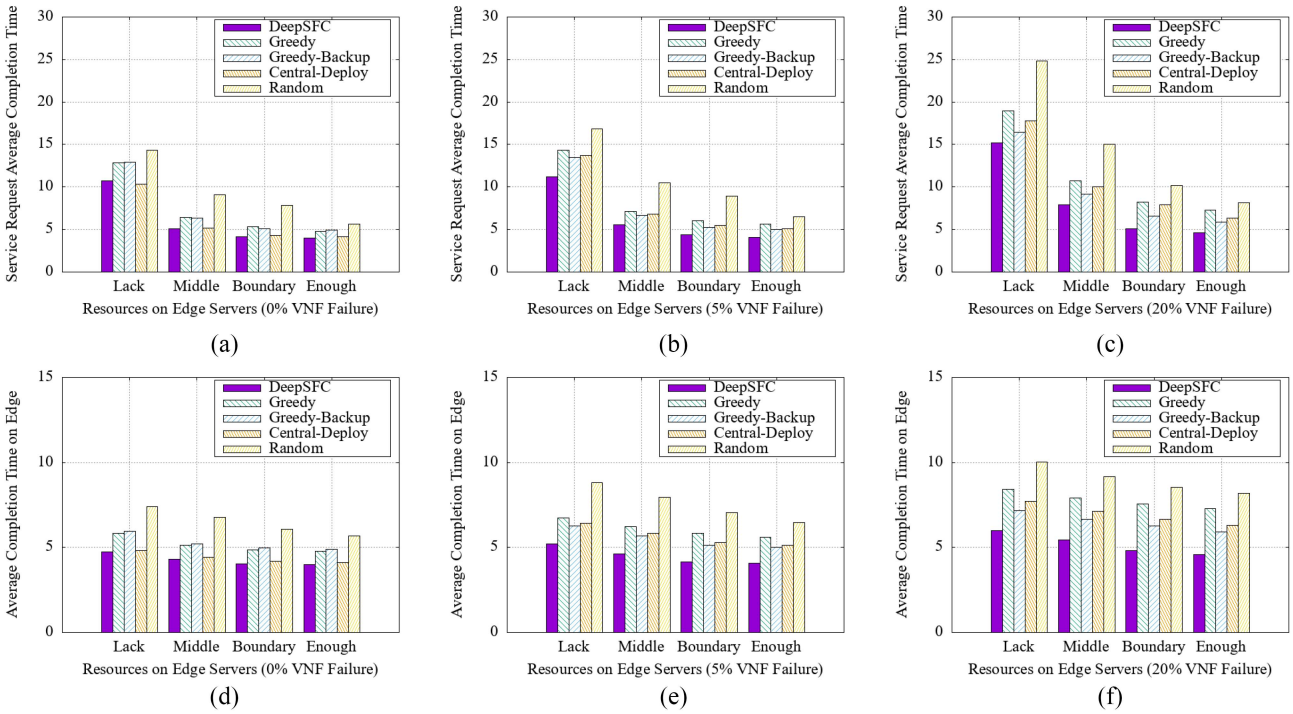

Fig. 5. Impact of different deployment scenarios and VNF failure rate on service request completion delay. (a) Total average completion time with 0% VNF failure. (b) Total average completion time with 5% VNF failure. (c) Total average completion time with 20% VNF failure. (d) Edge average completion time with 0% VNF failure. (e) Edge average completion time with 5% VNF failure. (f) Edge average completion time with 20% VNF failure.

Greedy-Backup, and Random, which can be attributed to DeepSFC considers optimizing placement delays. From scenario Lack to Enough, Central-Deploy and DeepSFC perform similarly. The reason for this phenomenon is that the advantages of DeepSFC in VNF backup deployment are not reflected when the VNF failure rate is 0%.

Fig. 5(b) shows the case where the VNF failure rate is 5%. Compared with Fig. 5(a), we can find that in different deployment scenarios, since some VNFs may fail, service requests need to go to the edge server where the backup VNFs are located, or even to the cloud or neighboring edge computing environments. Therefore, each algorithm's service request completion time has different degrees of increase. We can find the rate of change of DeepSFC is always lower than other algorithms, and the performance is better than other algorithms. In addition, as resources become more abundant in edge environment, the performance of Greedy and Central-Deploy does not change significantly, since they do not consider backing up VNFs to deal with VNF failures. The performance of Greedy-Backup and Random improves with abundant resources, but their performance is worse than DeepSFC, since they do not consider placement delay.

Fig. 5(c) shows the VNF failure rate rising to 20%. Compared with Fig. 5(a) and (b), we find that as the failure rate increases, the performance of DeepSFC is relatively better. When resources are lacking, DeepSFC will discard some SFCs with lower benefits (SFCs with low user requirements and high resource consumption), then deploy SFCs and VNF backups with higher benefits (VNFs with high user requirements or low resource consumption). With the gradual abundance of resources, although Greedy-Backup and Random do not consider the impact of deployment location on completion delays, while DeepSFC prefers to choose a server with a lower deployment delay to deploy VNF backup and SFC, the performance of DeepSFC is better than Greedy-Backup and Random. In addition, DeepSFC still outperforms other algorithms in terms of rate of change.

2) Service Request Average Completion Time on Edge: Fig. 5(d)–(f) shows the average completion time of service requests on different edge deployment scenarios. Since DeepSFC considers the impact of different VNF deployment locations, DeepSFC tends to deploy VNFs and their backups on a chain in locations with less latency. Hence, the performance of DeepSFC is better than other algorithms.

3) Proportion of Bakeup VNFs: In addition, considering that VNF backups greatly impact the long-term availability of SFCs, we evaluated the VNF backup ratios generated by different algorithms in different deployment scenarios. As shown in Fig. 6, Greedy and Central-Deploy do not consider backup VNFs, so their ratios are 0. Compared with other algorithms, when the VNF failure rate increases, DeepSFC generates more VNF backups to ensure the availability of SFC. In addition, when the VNF is not failed, or the failure rate is lower, DeepSFC is more inclined to use resources to deploy new instead of deploy more. After deploying the SFC required by users, DeepSFC tends to deploy more SFC backups when the failure rate is low, so as to improve resource utilization. When the VNF failure rate increases, DeepSFC tends to deploy failed VNF backups, which reflects the elegant tradeoff between VNF failure rate and resource utilization rate.

4) Different SFC Diverse Requirements: In this section, we compare DeepSFC with the existing methods in terms

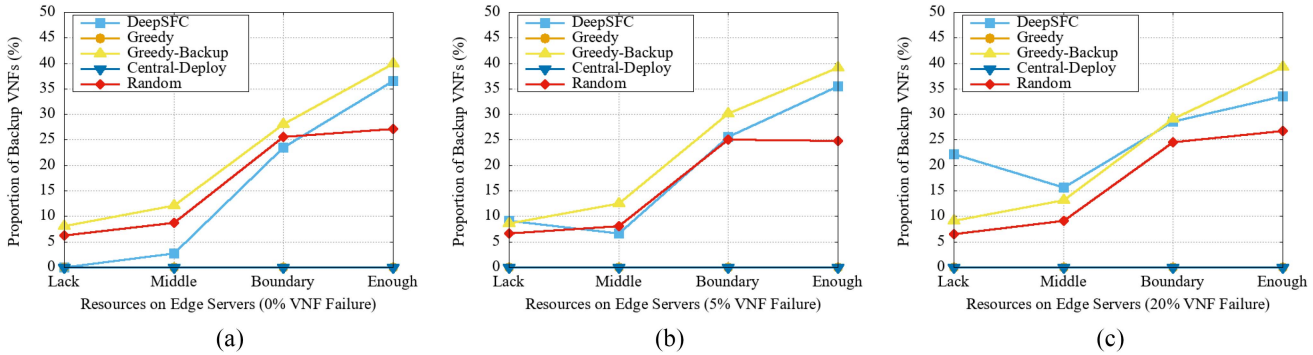

Fig. 6. Impact of different deployment scenarios and VNF failure rate on the VNF backup ratio. (a) Ratio of VNF backups without VNF failure rate. (b) Ratio of VNF backups when the failure rate of VNF is 5%. (c) Ratio of VNF backups when the failure rate of VNF is 20%.

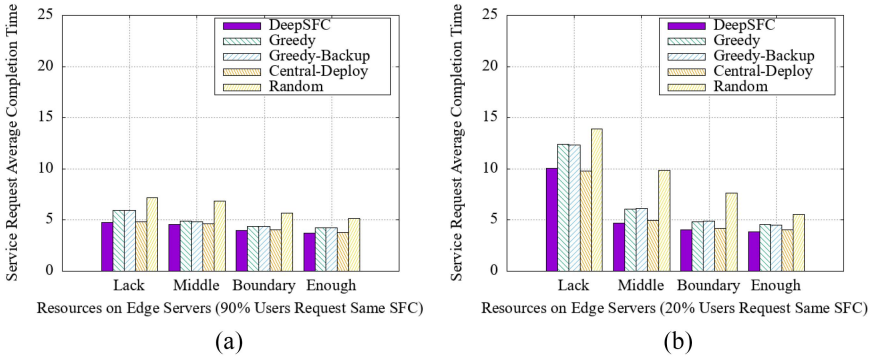

Fig. 7. Impact of different proportions of users requesting the same SFC. (a) Total average completion time with 90% users request same SFC. (b) Total average completion time with 20% users request same SFC.

of the proportion of users requiring the same SFC. Fig. 7(a) and (b) show two cases where 90% and 20% of the users requested the same SFC. The potential advantage of increased resources is that there are more options for SFC deployment locations. In Fig. 7(a) and (b), DeepSFC and Central-Deploy both show better performance, due to the fact that they both take into account the optimization of SFC deployment locations. Compared to Fig. 7(a), with a larger proportion of users requiring the same SFC, the overall completion delay of the service requests in Fig. 7(b) is greatly reduced. Since more users demand the same SFCs, in Fig. 7(b), the diversity of SFCs deployed by the algorithms is less considered, and therefore, the increase in the amount of resources is not as effective in improving the completion time as in Fig. 7(a).

5) Resource Utilization and Rate at Edge: Fig. 8 shows the resource utilization of different algorithms in different deployment scenarios, and we can find that although the resource utilization of DeepSFC is not the highest, it is 2%–3% lower than the optimal algorithm. But DeepSFC is superior in other indicators. Fig. 9 shows the proportion of service requests completed in the edge environment, which directly reflects the algorithm's performance when the overall VNF failure rate increases. We can find that DeepSFC performs well in different deployment scenarios, which demonstrates the flexibility and generalizability of DeepSFC.

6) Redeployment Cost: In the experiments, we explore the impact of the proportion of old VNFs in the new deployment scheme on the cost of relocation. As shown in Fig. 10,

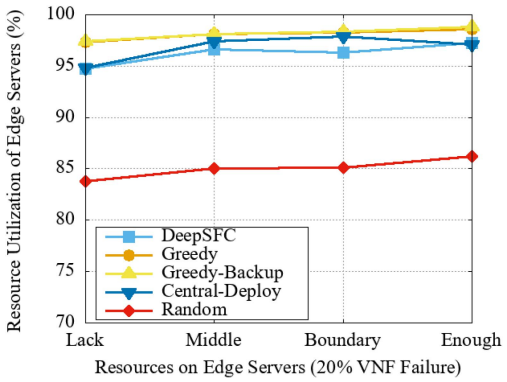

Fig. 8. Impact of different deployment scenarios on resource utilization when the failure rate of VNF is 20%.

we can find some fluctuations in the performance of cloud algorithms. This can be attributed to changes in the number of VNFs in deployment scenarios. As the proportion of old VNFs increases, cloud algorithms do not show significant performance improvements. However, since DeepSFC considers migrating or retaining VNFs from edge environments, DeepSFC is increasingly outperforming the cloud, which brings lower costs.

7) Learning Process of DQN-Based Deployer: As shown in Fig. 11, we present the learning process of a DQN-based deployer, which interacts with the environment according to a reward function. DQN-based deployer can learn in an iterative process. As seen from the figure, when the deployer starts

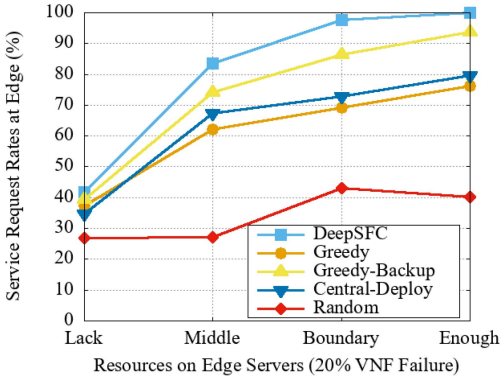

Fig. 9. Impact of different deployment scenarios on the completion ratio of service requests at the edge when the failure rate of VNF is 20%.

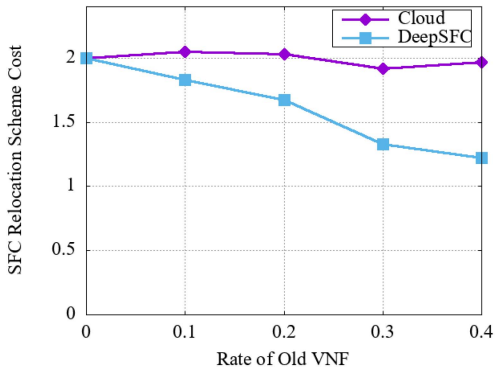

Fig. 10. Impact of different old VNF rates on the relocation scheme cost.

learning, the reward value gradually increases as the number of learning increases. Finally, it tends to be stable at 250, which indicates that the deployer has gradually formed a long-term reward-optimal deployment scheme.

VII. Related Work

Research related to deploying SFC at the edge has recently gained momentum [38], [39], [40], [42]. Current SFC-related designs mainly focus on deploying new VNFs to serve more requests, ignoring the possibility of functional failure, or backing up the existed VNFs to enhance the function availability at the expense of feature diversification. Another part of fine-grained design deploys VNFs on the same chain as close as possible to reduce on-chain latency. However, they neglected to deploy redundant VNF backups in fine-grained locations, which may cause some important VNFs to fail to work.

When deploying SFC at the edge, Jin et al. [23] hope to find the optimal deployment scheme of SFC by minimizing the resource consumption in the edge computing environment. Cziva et al. [26] presented a way to dynamically reschedule the optimal placement of VNFs based on temporal network-wide latency fluctuations using optimal stopping theory, which can minimize end-to-end latency from all users to their associated VNFs. Li et al. [27] developed a near-optimal approximation algorithm for the placement of his SFC by employing the Markov approximation technique, under the assumption of uncertainty in computing resources and data rates demanded by request executions. Xu et al. [28] balanced

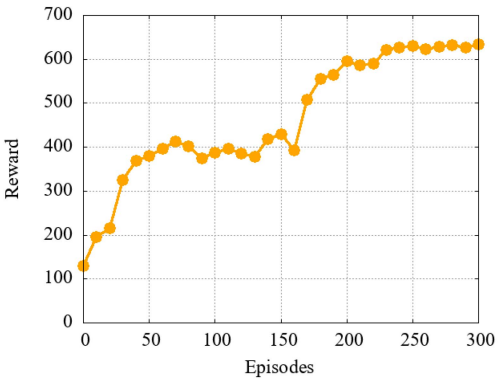

Fig. 11. Leaning process of DRL-based deployer.

the QoS and the consumption of computing and communication resources, design a multiobjective SFC deployment model to represent the diverse business requirements and specific network environments, and propose a learning-based online SFC deployment algorithm. Liu et al. [29] considered the constant dynamics of IoT networks through actor–critic and deterministic policy gradient scheme, and provide a DRL-based SFC-DOP algorithm to determine the location of the SFC. In the context of multiple source update systems, Chen et al. [33] proposed a VNF deployment scheme to minimize the long-term average AoI of all updates received at the destination, quantifying the freshness of the data from the perspective of the destination. Sun et al. [34] payed attention to the joint optimization problem of VNF Placement, CPU Allocation, and flow Routing (VNFPAR) in the scenarios consisting of VNFs that can dynamically change traffic. Basu et al. [35] considered a multilayer SFC formation for adaptive VNF allocation on dynamic slices, with VNF selection mechanisms to optimize resource utilization. In MEC, Masoumi et al. [36] considered the tradeoff between resource deployment costs and effective service delivery. To address dynamic workloads and orchestrate complex distributed environments, Harris et al. [37] investigated methods to dynamically place network functions at edge nodes in the network to maximize customer satisfaction. Considering the importance of services, Nguyen et al. [41] designed three algorithms to solve the VNF placement problem for weighted services.

At the same time, considering the unavoidable failure of VNFs in the working process, ensuring the high availability of VNFs has become an important issue to continuously meet the service requirements of end users [18], [19]. Among them, using redundancy to create backups for VNFs has become a representative method to solve the availability problem [20], [21]. Shang et al. [30] combined a static backup plan with a dynamic backup scheme to prevent VNF failures without a fixed network failure rate. When the original VNF or its static backup fails, the algorithm will immediately deploy a dynamic backup, enabling adaptive adjustment. Qiu et al. [32] discretized the long-term supply problem into a series of single-slot optimization problems to deal with real-time-varying failure probabilities, and proposed an online approximation scheme with a constant approximation ratio. Dinh and Kim [31] formulated a VNF redundancy allocation

scheme with cost-effectiveness in mind, which selects a suitable set of VNFs for backup by comparing the availability improvement potential of VNFs to minimize the redundancy allocation cost in the network. Yala et al. [43] proposed a VNF placement scheme between edge and cloud to optimize the tradeoff between availability and latency. Wang et al. [44] determined the backup of VNFs from the user's perspective to meet their service needs on SFC as much as possible. They resort to the combinatorial multiarmed bandit (CMAB) problem to propose online learning based on approximate VNF backup selection and deployment algorithm.

Current SFC-related designs mainly focus on deploying new VNFs to serve more requests, ignoring the possibility of functional failure, or backing up the existed VNFs to enhance the function availability at the expense of feature diversification. Another part of fine-grained design deploys VNFs on the same chain as close as possible to reduce on-chain latency. However, they neglected to deploy redundant VNF backups in fine-grained locations, which may cause some important VNFs to fail to work.

VIII. Conclusion

In this article, we propose the DeepSFC design at the network edge, with the target of reducing the average completion delay of SFC-related tasks in IoT. DeepSFC considers the impact of resource allocations and deployment locations on the average latency of overall service requests, which realizes an elegant tradeoff between SFC diversity and availability. When making an SFC deployment scheme, DeepSFC first determines the type and number of VNFs that need to be deployed. Thereafter, DeepSFC optimizes the deployment locations of these chosen VNFs in the service chain. For more general scenarios wherein user's service requirements change or the deployed server crashes, DeepSFC further relocates the VNF deployment with the joint consideration of performance degradation and migration cost. Theoretical analysis and experiments show that DeepSFC outperforms its competitors and realizes our design rationales.

Acknowledgment

The authors thank all the anonymous reviewers for the insightful feedback.

References

[1] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta, and D. Sabella, "On multi-access edge computing: A survey of the emerging 5G network edge cloud architecture and orchestration," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 3, pp. 1657–1681, 3rd Quart., 2017.

[2] Y. Chen, X. Zhao, X. Lin, Y. Wang, and D. Guo, "Efficient mining of frequent patterns on uncertain graphs," *IEEE Trans. Knowl. Data Eng.*, vol. 31, no. 2, pp. 287–300, Feb. 2019.

[3] J. Fan, Y. Yan, Y. Kaiguo, and Z. Lujing, "Security analysis of 5G authentication and key agreement protocol," *J. Tsinghua Univ. Sci. Technol.*, vol. 61, no. 11, pp. 1260–1266, 2021.

[4] M. F. Khan, "An approach for optimal base station selection in 5G HetNets for smart factories," in *Proc. IEEE 21st Int. Symp. World Wireless, Mobile Multimedia Netw.*, 2020, pp. 64–65.

[5] Z. Kan, H. Lu, H. Meng, Z. Qiang, L. Ning, and L. Lei, "Soft-defined heterogeneous vehicular network: Architecture and challenges," *IEEE Netw.*, vol. 30, no. 4, pp. 72–80, Jul./Aug. 2016.

[6] M. Pouryazdan, B. Kantarci, T. Soyata, and H. Song, "Anchor-assisted and vote-based trustworthiness assurance in smart city crowdsensing," *IEEE Access*, vol. 4, pp. 529–541, 2016.

[7] A. Abdulghaffar, A. Mahmoud, M. Abu-Amara, and T. Sheltami, "Modeling and evaluation of software defined networking based 5G core network architecture," *IEEE Access*, vol. 9, pp. 10179–10198, 2021.

[8] J. Ha and N. Park, "Unified control architecture for 5G convergence network," in *Proc. Int. Conf. Circuits, Devices Syst. (ICCDS)*, 2017, pp. 226–230.

[9] G. Baldoni et al., "Edge computing enhancements in an NFV-based ecosystem for 5G neutral hosts," in *Proc. IEEE Conf. Netw. Funct. Virtualization Softw. Defined Netw. (NFV-SDN)*, 2018, pp. 1–5.

[10] B. Mahapatra, A. K. Turuk, S. K. Patra, and R. Kumar, "Optimal placement of centralized BBU (C-BBU) for fronthaul and backhaul optimization in cloud-RAN network," in *Proc. Int. Conf. Inf. Technol. (ICIT)*, 2017, pp. 107–112.

[11] D. Gedia and L. Perigo, "Latency-aware, static, and dynamic decision-tree placement algorithm for containerized SDN-VNF in OpenFlow Architectures," in *Proc. IEEE Conf. Netw. Funct. Virtualization Softw. Defined Netw. (NFV-SDN)*, 2019, pp. 1–7.

[12] C. H. Park and D. R. Shin, "VNF management method using VNF group table in network function virtualization," in *Proc. 19th Int. Conf. Adv. Commun. Technol. (ICACT)*, 2017, pp. 210–212.

[13] A. A. Kherani et al., "Development of MEC system for indigenous 5G test-bed," in *Proc. Int. Conf. Commun. Syst. Netw. (COMSNETS)*, 2021, pp. 131–133.

[14] N. Slamnik-Kriještorac, G. M. Yilma, F. Z. Yousaf, M. Liebsch, and J. M. Marquez-Barja, "Multi-domain MEC orchestration platform for enhanced back situation awareness," in *Proc. IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, 2021, pp. 1–2.

[15] B. Brik, P. A. Frangoudis, and A. Ksentini, "Service-oriented MEC applications placement in a federated edge cloud architecture," in *Proc. IEEE Int. Conf. Commun. (ICC)*, 2020, pp. 1–6.

[16] W. Liang, Y. Ma, W. Xu, Z. Xu, X. Jia, and W. Zhou, "Request reliability augmentation with service function chain requirements in mobile edge computing," *IEEE Trans. Mobile Comput.*, vol. 21, no. 12, pp. 4541–4554, Dec. 2022.

[17] Y. Liu et al., "A Lagrangian-relaxation-based approach for service function chain dynamic orchestration for the Internet of Things," *IEEE Internet Things J.*, vol. 8, no. 23, pp. 17071–17089, Dec. 2021.

[18] T. Taleb, A. Ksentini, and B. Sericola, "On service resilience in cloud-native 5G mobile systems," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 3, pp. 483–496, Mar. 2016.

[19] J. Fan, C. Guan, Y. Zhao, and C. Qiao, "Availability-aware mapping of service function chains," in *Proc. IEEE Conf. Comput. Commun.*, 2017, pp. 1–9.

[20] J. Zhang, Z. Wang, C. Peng, L. Zhang, T. Huang, and Y. Liu, "RABA: Resource-aware backup allocation for a chain of virtual network functions," in *Proc. IEEE Conf. Comput. Commun.*, 2019, pp. 1918–1926.

[21] J. Fan, Z. Ye, C. Guan, X. Gao, K. Ren, and C. Qiao, "GREP: Guaranteeing reliability with enhanced protection in NFV," in *Proc. ACM SIGCOMM Workshop Hot Topics Middleboxes Netw. Funct. Virtualization*, 2015, pp. 13–18.

[22] Y. Liu, M. Peng, G. Shou, Y. Chen, and S. Chen, "Toward edge intelligence: Multiaccess edge computing for 5G and Internet of Things," *IEEE Internet Things J.*, vol. 7, no. 8, pp. 6722–6747, Aug. 2020.

[23] P. Jin, X. Fei, Q. Zhang, F. Liu, and B. Li, "Latency-aware VNF chain deployment with efficient resource reuse at network edge," in *Proc. IEEE Conf. Comput. Commun.*, 2020, pp. 267–276.

[24] Y. Ma, W. Liang, M. Huang, W. Xu, and S. Guo, "Virtual network function service provisioning in MEC via trading off the usages between computing and communication resources," *IEEE Trans. Cloud Comput.*, vol. 10, no. 4, pp. 2949–2963, Oct.–Dec. 2022.

[25] S. Song, C. Lee, H. Cho, G. Lim, and J. Chung, "Clustered virtualized network functions resource allocation based on context-aware grouping in 5G edge networks," *IEEE Trans. Mobile Comput.*, vol. 19, no. 5, pp. 1072–1083, May 2020.

[26] R. Cziva, C. Anagnostopoulos, and D. P. Pezaros, "Dynamic, latency-optimal VNF placement at the network edge," in *Proc. IEEE Conf. Comput. Commun.*, 2018, pp. 693–701.

[27] J. Li, W. Liang, and Y. Ma, "Robust service provisioning with service function chain requirements in mobile edge computing," *IEEE Trans. Netw. Service Manag.*, vol. 18, no. 2, pp. 2138–2153, Jun. 2021.

[28] S. Xu, Y. Li, S. Guo, C. Lei, D. Liu, and X. Qiu, "Cloud-edge collaborative SFC mapping for industrial IoT using deep reinforcement learning," *IEEE Trans. Ind. Informat.*, vol. 18, no. 6, pp. 4158–4168, Jun. 2022.
[29] Y. Liu, H. Lu, X. Li, Y. Zhang, L. Xi, and D. Zhao, "Dynamic service function chain orchestration for NFV/MEC-enabled IoT networks: A deep reinforcement learning approach," *IEEE Internet Things J.*, vol. 8, no. 9, pp. 7450–7465, May 2021.
[30] X. Shang, Y. Huang, Z. Liu, and Y. Yang, "Reducing the service function chain backup cost over the edge and cloud by a self-adapting scheme," in *Proc. IEEE Conf. Comput. Commun.*, 2020, pp. 2096–2105.
[31] N. T. Dinh and Y. Kim, "An efficient availability guaranteed deployment scheme for IoT service chains over fog-core cloud networks," *Sensors*, vol. 18, no. 11, p. 3970, 2018.
[32] Y. Qiu, J. Liang, V. C. M. Leung, X. Wu, and X. Deng, "Online reliability-enhanced virtual network services provisioning in fault-prone mobile edge cloud," *IEEE Trans. Wireless Commun.*, vol. 21, no. 9, pp. 7299–7313, Sep. 2022.
[33] Z. Chen, H. Li, K. Ota, and M. Dong, "Deep reinforcement learning for AoI aware VNF placement in multiple source systems," in *Proc. IEEE Global Commun. Conf.*, 2022, pp. 2873–2878.
[34] J. Sun, F. Liu, H. Wang, and D. O. Wu, "Joint VNF placement, CPU allocation, and flow routing for traffic changes," *IEEE Internet Things J.*, vol. 10, no. 2, pp. 1208–1222, Jan. 2023.
[35] D. Basu, S. Kal, U. Ghosh, and R. Datta, "DRIVE: Dynamic resource introspection and VNF embedding for 5G using machine learning," *IEEE Internet Things J.*, early access, Jan. 9, 2023, doi: 10.1109/JIOT.2023.3235382.
[36] M. Masoumi et al., "Dynamic online VNF placement with different protection schemes in a MEC environment," in *Proc. 32nd Int. Telecommun. Netw. Appl. Conf. (ITNAC)*, 2022, pp. 1–6.
[37] D. Harris and D. Raz, "Dynamic VNF placement in 5G edge nodes," in *Proc. IEEE 8th Int. Conf. Netw. Softwarization (NetSoft)*, 2022, pp. 216–224.
[38] Q. Wei, P. Han, and Y. Liu, "Mobility-aware multi-instance VNF placement in mobile edge computing networks," in *Proc. Int. Wireless Commun. Mobile Comput. (IWCMC)*, 2021, pp. 1303–1308.
[39] C. R. de Mendoza, B. Bakhshi, E. Zeydan, and J. Mangues-Bafalluy, "Near optimal VNF placement in edge-enabled 6G networks," in *Proc. 25th Conf. Innov. Clouds, Internet Netw. (ICIN)*, 2022, pp. 136–140.
[40] X. Shang, Z. Liu, and Y. Yang, "Online service function chain placement for cost-effectiveness and network congestion control," *IEEE Trans. Comput.*, vol. 71, no. 1, pp. 27–39, Jan. 2022.
[41] D. H. P. Nguyen, Y. Lien, B. Liu, S. Chu, and T. N. Nguyen, "Virtual network function placement for serving weighted services in NFV-enabled networks," *IEEE Syst. J.*, early access, Mar. 30, 2023, doi: 10.1109/JSYST.2023.3257776.
[42] W. Attaoui, E. Sabir, H. Elbiaze, and M. Guizani, "VNF and CNF placement in 5G: Recent advances and future trends," *IEEE Trans. Netw. Service Manag.*, early access, Mar. 31, 2023, doi: 10.1109/TNSM.2023.3264005.
[43] L. Yala, P. A. Frangoudis, and A. Ksentini, "Latency and availability driven VNF placement in a MEC-NFV environment," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, 2018, pp. 1–7.
[44] C. Wang, Q. Hu, D. Yu, and X. Cheng, "Proactive deployment of chain-based VNF backup at the edge using online bandit learning," in *Proc. IEEE 41st Int. Conf. Distrib. Comput. Syst. (ICDCS)*, 2021, pp. 740–750.

Zongyang Yuan received the B.S. degree in machinery from Northeast Agricultural University, Harbin, China, in 2021. He is currently pursuing the Ph.D. degree with the College of Systems Engineering, National University of Defense Technology, Changsha, China.

His research interests include Internet of Things and edge computing.

Lailong Luo received the B.S., M.S., and Ph.D. degrees from the School of Systems Engineering, National University of Defense Technology, Changsha, China, in 2013, 2015, and 2019, respectively.

He is currently an Associate Professor with the School of Systems Engineering, National University of Defense Technology. His research interests include probabilistic data structures and data analysis.

Deke Guo (Senior Member, IEEE) received the B.S. degree in industry engineering from Beijing University of Aeronautics and Astronautics, Beijing, China, in 2001, and the Ph.D. degree in management science and engineering from the National University of Defense Technology, Changsha, China, in 2008.

He is currently a Professor with the College of System Engineering, National University of Defense Technology. His research interests include distributed systems, software defined networking, data center networking, wireless and mobile systems, and interconnection networks.

Prof. Guo is a member of the ACM.

Denis C.-K. Wong received the B.Sc. and M.Sc. degree in mathematics and the Ph.D. degree in algebra/coding theory from Universiti Sains Malaysia, Gelugor, Malaysia, in 2001, 2004, and 2013, respectively.

He is currently an Assistant Professor and the Head of programme for the program, master's of Mathematics programme, Universiti Tunku Abdul Rahman, Kampar, Malaysia. His research interests include algebraic coding theory, algebraic combinatorics, and cryptography.

Geyao Cheng received the B.S. and M.S. degrees in management science and engineering from the National University of Defense Technology, Changsha, China, in 2017 and 2019, respectively, where she is currently pursuing the Ph.D. degree with the College of Systems Engineering.

Her research interests include edge computing and distributed system.

Bangbang Ren received the bachelor's, master's, and Ph.D. degrees from the National University of Defense Technology, Changsha, China, in 2015, 2017, and 2021, respectively.

He was also a Visiting Research Scholar with the University of Göttingen, Göttingen, Germany, in 2019. His research interests include software-defined network and network optimization.

Qianzhen Zhang received the Ph.D. degree from the National University of Defense Technology, Changsha, China, in 2022.

He is currently a Lecturer with the College of Systems Engineering, National University of Defense Technology. His research interests include continuous subgraph matching, graph data analytics, and knowledge graph.