Scalable Orchestration of Service Function Chains in NFV-enabled Networks: A Federated Reinforcement Learning Approach

Haojun Huang, Cheng Zeng, Yangmin Zhao, Geyong Min, YingYing Zhu, Wang Miao and Jia Hu

***Abstract*—Network function virtualization (NFV) is critical to the scalability and flexibility of various network services in the form of service function chains (SFCs), which refer to a set of Virtual Network Functions (VNFs) chained in a specific order. However, the NFV performance is hard to fulfill the ever-increasing requirements of network services mainly due to the static orchestrations of SFCs. To tackle this issue, a novel Scalable SFC Orchestration (SSCO) scheme is proposed in this paper for NFV-enabled networks via federated reinforcement learning. SSCO has three remarkable characteristics distinguishing from the previous work: (1) A federated-learning-based framework is designed to train a global learning model, with time-variant local model explorations, for scalable SFC orchestration, while avoiding data sharing among stakeholders; (2) SSCO allows for parameter update among local clients and the cloud server just at the first and last epochs of each episode to ensure that distributed clients can make model optimization at a low communication cost; (3) SSCO introduces an efficient deep reinforcement learning (DRL) approach, with the local learning knowledge of available resources and instantiation cost, to map VNFs into networks flexibly. Furthermore, a loss-weight-based mechanism is proposed to generate and exploit reference samples in replay buffers for future training, avoiding the strong relevance of samples. Simulation results obtained from different working scenarios demonstrate that SSCO can significantly reduce placement errors and improve resource utilization ratio to place time-variant VNFs compared with the state-of-the-art mechanisms. Furthermore, the results show that the proposed approach can achieve desirable scalability.**

***Index Terms*—Network function virtualization, service function chains, federated learning, deep reinforcement learning, resource allocation.**

I. Introduction

NETWORK Function Virtualization (NFV) has been considered as a promising network initiative to implement network functions and services in an efficient and effective manner [1], [2]. The main idea is to decouple network functions, e.g., firewall, load balancing and video caching, from dedicated hardware and run them on industry standard servers in software, known as Virtual Network Functions (VNFs) [3], [4]. Being an innovative step toward executing a lower-cost agile infrastructure, NFV has attracted tremendous interests from academia and industry. Today, global telecom companies and standardization groups are strongly supporting NFV [5], and furthermore, most of them such as AT&T, BT, Telecom Italia, and Telefonica, are releasing infrastructure to realize NFV-based networks.

To realise desired network services, VNFs should be orchestrated into the appropriate virtualized infrastructure in the form of software-based Service Function Chains (SFCs), which include a set of VNFs chained in a specific order [2], [6], under given resource constraints. Existing studies attempt to resolve this issue with different goals like expense reduction [7], network utility maximization [6], and performance acceleration [8]. However, in reality, the benefits of NFVs often come with considerable compromise mainly due to the static orchestrations of SFCs with these solutions. Specifically, most of them often instantiate SFCs in the underlaying infrastructure with the fixed allocated resources in terms of storage, computing, and bandwidths, etc. However, the real resources required by the running VNFs always vary with time [4], [9], [10], leading to a large-scale dimension of network states and actions for SFC orchestration, which refers to network resource, network-related factors, resource perception and VNF mapping [11], [12]. As a result, the performance of SFCs with static orchestration is greatly affected by the current and future network states and actions [6], [9], [13], and comes with the considerable compromise with ever-increasing NFV-based applications.

More recently, Federated Reinforcement Learning (FRL) has made significant breakthrough and impacted a variety of fields such as digital manufacturing, intelligent control, and energy management [14], [15], [16], which is expected to become a promising paradigm to tackle the above-mentioned challenges. However, the current FRL solutions suffer extreme time issue when the dimension of either network states or taken actions enlarges. Furthermore, it is becoming more complicated for current FRL to make decisions on SFC orchestrations with the ever-increasing continuous states and discrete actions simultaneously. All of these will further hinder the convergence of model training for SFC orchestration and thus motivate new FRL-based approaches to orchestrate each VNF into an appropriate infrastructure with the dynamically available resources in an efficient and rapid manner.

To this end, in this paper, we propose a Scalable Service Function Chain Orchestration (SSCO) solution via FRL, which

H. Huang, C. Zeng, and Y. Zhu are with the School of Electronic Information and Communications, Huazhong University of Science and Technology, Wuhan, 430074, China. Email: {hjhuang, c_zeng, yyzhu}@hust.edu.cn.

Y. Zhao is with the Department of Computer Science and Engineering, University at Buffalo, The State University of New York, USA. Email: yangming@buffalo.edu.

G. Min, W. Miao and J. Hu are with the Department of Computer Science, University of Exeter, Exeter, EX4 4QF, UK. Email: {g.min, wang.miao, j.hu}@exeter.ac.uk.

Manuscript received Jan 1, 2021; revised March. 10, 2021 (Corresponding author: Yingying Zhu).

introduces Federated Learning (FL) into the global model training and Deep Reinforcement Learning (DRL) into local model training for scalable SFC deployment with the available resources in a distributed and efficient manner. FL in our solution is used as the learning architecture and DRL as the scalable feature extractor to build the relationships among high-dimensional states, actions and rewards, related to network configurations, SFC requests, our objective optimization and constraints, for scalable SFC orchestration. SSCO works as follows. First, SSCO divides the whole network into a set of regions and assigns an agent in each region to train a local mode for SFC orchestration. Then, the cloud server specifies an initial pre-trained model and sends it to each local client. After that, each agent trains its local model and reports it to the cloud server for global model aggregation and improving via federated learning, without data sharing among them. This process will continue until a global model has been learned. Finally, each VNF is deployed in networks following the learned policy. The main contributions of this paper are summarized as follows.

- The FRL-based framework is proposed to train a global learning model for scalable SFC orchestration with the dynamically available resources associated with VNFs over time, through distributed learning in a set of clients. Both placement-error-rate-based and reward-based federated weighted strategies are designed for clients to participate in the global model training in an optimal iterative manner with quick convergence. Furthermore, the maximum-waiting-time and average-waiting-time explorations have been developed for the cloud server to train the global model, enabling to deploy SFCs in networks more flexibly than the traditional solutions.
- The DRL-based solution using DQN is introduced to reinforce distributed local learning models for the upcoming VNFs deployment with the available resources in an efficient and scalable manner. Furthermore, the loss-weight-based mechanism is proposed to generate and exploit reference samples in replay buffer for future training, avoiding the strong relevance of training samples.
- Extensive numerical analysis and simulation experiments are carried out under various network configurations to estimate the performance of the developed solution. The results show that SSCO can flexibly orchestrate SFCs in NFV-enabled networks.

The remainder of this paper is organized as follows. Section II provides an overview of the related work. Section III describes the system models and problem formulation. Then, Section IV introduces in detail the proposed SSCO for scalable deployment of SFCs in NFV-enabled networks. After that, specific implementations, results and performance comparisons are represented in Section V. Finally, we conclude the paper in Section VI.

II. Related Work

Over the past few years, many SFC orchestration approaches have been proposed with different goals, including expense reduction [7], network utility maximization [6], and performance acceleration [8], [17], [18], [19], [20], [21], [22], [23]. A comprehensive survey of those investigations is presented in [17], [18]. Generally speaking, these solutions with respect to SFC orchestration can be classified into two categories: (a) static and (b) dynamic orchestration.

Static orchestration: These approaches instantiate SFCs in given physical equipment with the allocated resources like storage, computing and bandwidths. All of these instantiated VNFs and SFCs will be quantitatively allocated the resources in their whole operations. Notice that the SFC orchestration is a NP-hard problem [7], many studies have suggested approximate or heuristic schemes to solve the static orchestration with different concerns, including the number of servers [24], network utility [25], and resource overhead [20]. In order to minimize the number of servers, a hybrid network function placement model has been proposed for small-scale networks [24]. To decrease CAPEX and OPEX, a scalable orchestration strategy has been proposed, while ignoring the changed processing capacity of VNFIs [19]. In addition, a VNF migration solution has been designed to increase network throughput during traffic steering [20]. Due to static SFC orchestration, all of these solutions cannot dynamically fulfill the service requirements of SFCs with time.

Dynamic orchestration: These solutions instantiate SFCs in the available network infrastructure, and dynamically allocate network resources for their operations [26]. Generally, such solutions target the SFC orchestration problem with the goal of providing enough resources (e.g., CPU, RAM and storage) [27], service nodes (e.g., routing path between VNFs) [28], [21], or a combination of them [22] for each VNF with time as far as possible. For example, a dynamic SFC deployment and readjustment solution has been proposed to decrease OPEX by Column Generation and Lyapunov optimization [27]. However, this work merely adapts to the hitherto traffic instead of subsequent requests. Thus, there are a few studies [29], [10], [30], [23] reported in the direction of proactive traffic forecasting. With proactively learning the traffic trend, such solutions complete the auto-scaling of VNFs and better utilize the re-aggregated SFCs to meet long-lasting demands. Unfortunately, these dynamic schemes are still heuristic with slow reactions and hard to train a large state space.

Currently, FRL and DRL [14], [31], [15], [16], [32], [33], [34], which combine static and dynamic ideas, are becoming attractive among the NFV community. Policy gradient has been used for auto-scaling policy judgments [9], and optimization models [13] have been developed to accelerate training for VNF management but with slow convergence. A specific Q-Learning [35] and DQN policy [11] have been presented, respectively, to improve the QoS in auto-scaling with an unsatisfactory reward. With this in mind, our proposed SSCO focuses on scalable SFC orchestration via FRL to fulfill the requirements of SFC with the available resources in a distributed and efficient manner. Different from the existing work, our DRL-based local models will be quickly aggregated and improved at the FL-based cloud server for efficient SFC deployment, without data sharing among local agents. Furthermore, the weight-based and time-variant federated training and weight-based sample exploration have been taken into consideration to ensure a more adaptive VNF scalability and

TABLE I: The Important Parameters and Notations

Notations	Descriptions
V/E	Set of nodes/links in the physical networks
F/L	Set of virtual nodes/links of all SFCs
λ_{f_i}	Setup cost to place VNF f_i
φ_{f_i}	Operation cost per unit time of VNF f_i
η_{uv}	Transport cost per unit over link uv
t/τ	Federated time/time interval
$x_{f_i}(t)$	The number of VNFs f_i in $G(V,E)$ at time t
$x_u^{f_i}(t)$	The i-th VNF deployment for node u
$y_{uv}^s(t)/y_{f_i}^s(t)$	SFC s deployment for links uv and VNF f_i
C_u^{cpu}/C_u^{mem}	CPU and memory capacity of u
r_u^{cpu}/r_u^{mem}	The available ratios of CPU and memory capacity of u
C_u^{bw}/r_u^{bw}	Bandwidth capacity/its available ratios of node u
$C(t)$	Total weighted cost of SFC deployment
σ/δ_i	Weight of different costs/clients
$\Theta(t)/\theta_t^i$	Global federated model/local Q-network weight
$\mathcal{S}/S_t$	State space /state in local neural networks training
$\mathcal{A}/A_t$	Action space/action in local training
R_t	Reward affected by action A_t
$Q/\hat{Q}$	Predictive/target neural networks
$\varepsilon_{rd}/\varepsilon_{\min}$	Exploration/minimum probability in $\varepsilon-greedy$ policy

higher service availability.

III. Problem Statement

In this section, we present the preliminary knowledge of network models and formulate the problem to solve. The important parameters used in this paper are listed in Table I.

A. Network Models

It is considered that the physical networks, which include a large number of physical nodes and links to host SFCs, are modeled as an undirected graph $G{=}(V,E)$. Both V and E denote the sets of nodes and links, respectively. The parameters u and $v \in V$ stand for two physical nodes, while uv denotes a link to connect them. The given CPU, memory, and bandwidth are considered indicators for nodes and links to host all VNFs. The bandwidth capacity of link uv is denoted as C_{uv}^{bw}, while the CPU and memory capacity of node u are defined as C_u^{cpu} and C_u^{mem}, respectively.

Similar to [36], each type of VNF is just deployed in virtualized server, which can host the corresponding network functions. Notice that each VNF may consume a specific number software/hardware resources, therefore, both C_{u,f_i}^{cpu} and C_{u,f_i}^{mem} are leveraged to respectively denote the number of CPU and memory resources required by VNF f_i on u. Let C_{uv,s_k}^{bw} denote the required bandwidth of the kth SFC s_k passing through link uv, and let $y_{uv}^s(t)$ indicate whether SFC s traverses link uv, defined as

$$y_{uv}^s(t) = \begin{cases} 1, & \text{if } s \text{ traverses } uv, \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

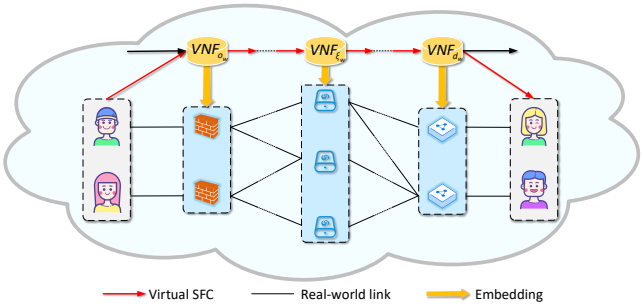


Fig. 1: SFC orchestration in NFV-enabled networks.

Let $x_u^{f_i}(t)$ denote whether VNF f_i is deployed in node u, given by

$$x_u^{f_i}(t) = \begin{cases} 1, & \text{if } f_i \text{ is deployed in } u, \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

Then, the available ratios of CPU, memory, and bandwidth of node u and uv, denoted by r_u^{cpu}, r_u^{mem} and r_{uv}^{bw}, respectively, can be defined as

$$\begin{cases} r_u^{cpu} = \dfrac{\sum_i x_u^{f_i}(t) C_{u,f_i}^{cpu}}{C_u^{cpu}}, \\ r_u^{mem} = \dfrac{\sum_i x_u^{f_i}(t) C_{u,f_i}^{mem}}{C_u^{mem}}, \\ r_{uv}^{bw} = \dfrac{\sum_i y_{uv}^{s_i}(t) C_{uv,s_i}^{bw}}{C_{uv}^{bw}}. \end{cases} \tag{3}$$

A traffic flow will traverse a series of VNFs hosted on different servers, scheduled in the form of a SFC illustrated in Fig. 1. All SFCs are described as a weighted directed graph $G_f{=}(F,L)$, where $F{=}\cup_{1\le i\le k}\{f_i\}$ and L denote two sets of VNFs $f_1, f_2, \cdots, f_k$ and virtual links connecting them, respectively. The parameters $\bar{u}$ and $\bar{v}$ represent two virtual nodes, while $\bar{u}\bar{v}$ denotes a virtual link to connect them. Similar to [2], the operation time of VNFs, T, i.e, the life cycle of VNF instance, is fallen in several episodes $T_1 \cdots T_i$, each of which consists of a series of epochs $\tau_1 \cdots \tau_k$.

B. Problem Formulation

Essentially, the VNF orchestration in NFV-based networks, as shown in Fig. 1, is closely related to the activation and deactivation of all VNFs with the available resources at the cost of setup, operation and communication, as described below.

The *setup cost* $C_{set}(t)$ is the total cost to activate all VNF instances through booting the VM image in the networks, defined as

$$C_{set}(t) = \sum_{u\in V}\sum_{f_i\in F} \lambda_{f_i} \max\{[x_u^{f_i}(t) - x_u^{f_i}(t-1)], 0\}, \tag{4}$$

where λ_{f_i} denotes the cost to setup VNF f_i in networks.

The *operation cost* $C_{op}(t)$ is referred to as the cost to run various types of VNFs in the whole operation time t,

depending on VNF type, server locations and operation time, and can be defined as follows:

$$C_{op}(t) = \sum_{u\in V}\sum_{f_i\in F}\varphi_{f_i}x_u^{f_i}(t) = \sum_{f_i\in F}\varphi_{f_i}x_{f_i}(t), \tag{5}$$

where $x_{f_i}(t)$ represents the number of VNFs $f_1, f_2, \cdots, f_n$ in physical networks at time t, and φ_{f_i} denotes the running cost per unit time of f_i.

The *communication cost* $C_{com}(t)$ is referred to as the expenditure on traffic transmission among all servers hosting VNFs, and can be given by

$$C_{com}(t) = \sum_{s\in G_f}\sum_{uv\in E}\eta_{uv}y_{uv}^s(t)\rho_{uv}^s(t), \tag{6}$$

where, $\rho_{uv}^s(t)$ stands for the number of flows of SFC s passing through link uv at time t, while η_{uv} denotes the transportation cost per unit flow of link $uv{\in}E$.

Then, the total weighted cost of VNF orchestration in networks can be expressed as

$$\begin{aligned} C(t) &= \sigma_1 C_{set}(t) + \sigma_2 C_{op}(t) + \sigma_3 C_{com}(t) \\ &= \sum_{f_i\in F}\sum_{u\in V}\sigma_1\lambda_{f_i}\max\{[x_u^{f_i}(t) - x_u^{f_i}(t-1)], 0\} + \\ &\sum_{f_i\in F}\sigma_2\varphi_{f_i}x_{f_i}(t) + \sum_{uv\in E}\sum_{s\in G_f}\sigma_3\eta_{uv}y_{uv}^s(t)\rho_{uv}^s(t), \end{aligned} \tag{7}$$

where σ_1, σ_2 and σ_3 are three weighted parameters to tune the tradeoff among $C_{set}(t)$, $C_{op}(t)$, and $C_{com}(t)$.

Built upon the above analysis, the scalable SFC orchestration problem in NFV-enabled networks can be formulated as follows:

$$\textbf{min:} \quad \sum_{t\in T}C(t), \tag{8}$$

$$\textbf{s.t.}\begin{cases} \sum_{u\in V}x_u^{f_i}(t) = x_{f_i}(t), & \forall t\in T, f_i\in F \\ \sum_{f_i\in F}x_u^{f_i}(t) = 1, & \forall t\in T, 0\le i\le x_f(t) \\ \sum_{f_i\in F}x_u^{f_i}(t)C_{u,f_i}^{cpu} \le C_u^{cpu}, & \forall u\in V, t\in T \\ \sum_{f_i\in F}x_u^{f_i}(t)C_{u,f_i}^{mem} \le C_u^{mem}, & \forall u\in V, t\in T \\ \sum_{s_i\in G_f}y_{uv}^{s_i}(t)C_{uv,s_i}^{bw} \le C_{uv}^{bw}, & \forall uv\in E, t\in T \\ \sum_{u\in V}y_{uv}^s(t)\rho_{uv}^s(t) = \sum_{w\in V}y_{vw}^s(t)\rho_{vw}^s(t), & \forall t\in T. \end{cases} \tag{9}$$

The first constraint in Eq. (9) ensures that one VNF can only be deployed on a node. The second constraint in Eq. (9) indicates that only VNF f_i can be deployed on node n. The following three constraints illustrate that the required resources C_{u,f_i}^{cpu}, C_{u,f_i}^{mem} and C_{uv,s_i}^{bw} of VNF f_i and link uv are no more than the maximum resource of node u and the bandwidth capacity of link uv, respectively. Besides, the last constraint indicates that any intermediate node, which a SFC s traverses through, needs to follow the flow-conservation principle except for its source node and destination node.

IV. SSCO: Scalable SFC Orchestration

This section presents in detail the proposed SSCO for scalable orchestration of SFCs in NFV-enabled networks. First, we present the essential 3-tuple for SSCO operations, and then introduce the framework of SSCO. Finally, we describe the local training process based on FRL in detail.

A. 3-Tuple Design

The specific learning framework of SSCO involves the close cooperations of DQN, including the neural networks and Q-learning networks, to implement iterative updates of a 3-tuple $<\mathcal{S},\mathcal{A},\mathcal{R}>$, which refers to the action, state and reward, respectively, for scalable SFC orchestration. The environments for the agents to take actions are referred to as the local clients. The agent is the VNF orchestrator and management, which mainly iteratively makes decisions to interact with the environments at each epoch. Once executing action A_t at state S_t, the agent steps into a new state S_{t+1} along with an updated reward R_t, which coincides with the optimization goal of SSCO, defined in Eqs. (8) and (9). The details of 3-tuple $<\mathcal{S},\mathcal{A},\mathcal{R}>$ associated with our scalable SFC orchestration are described as follows.

The **state** $S_t{\in}\mathcal{S}$ at time t is referred to as the information of all network resources and configurations, and can be denoted as

$$\begin{aligned} S_t = < & C_u^{cpu}(t), C_{u,f_i}^{cpu}(t), C_u^{mem}(t), r_u^{cpu}(t), r_u^{mem}(t), r_{uv}^{bw}(t), \\ & C_{u,f_i}^{mem}(t), C_{uv}^{bw}(t), C_{uv,s_i}^{bw}(t), \forall uv\in E, \forall u,v\in V, \\ & \forall s_i\in G_f > . \end{aligned} \tag{10}$$

Eq. (10) clearly states that network resources and configurations of node $u{\in}V$ and link $uv{\in}E$, in terms of CPU, memory and bandwidth, etc, will dynamically change with time.

The **action** $A_t{\in}\mathcal{A}$ at time t serves two different functions A_t^d and A_t^r, which specifies how to deploy VNFs flexibly and demonstrates how to apperceive and take advantage of available resources at the next epoch t, respectively. The action space $\mathcal{A}$ can be expressed as

$$A_t =< A_t^d, A_t^r >, A_t^d \cap A_t^r = \emptyset, \tag{11}$$

where $A_t^d{\in}\mathcal{A}$ refers to the actions of VNF deployment vectors, and $A_t^r{\in}\mathcal{A}$ consists of the actions of VNF resource perception and allocation. Each action a_t^d taken at the local client is responsible for an adaptive task, either VNF placement if $a_t^d{\in}A_t^d$ or resource allocation if $a_t^d{\in}A_t^r$.

The first action subspace A_t^d includes a discrete action set, and can be further divided into

$$A_t^d =< A_{f_i,t}^{cpu}, A_{f_i,t}^{mem}, A_{\bar{u}\bar{v},t}^{bw}, f_i\in F, \bar{u}\bar{v}\in L > . \tag{12}$$

Both $A_{f_i,t}^{cpu}$ and $A_{f_i,t}^{mem}$ denote the action to deploy VNF f_i in networks, while $A_{\bar{u}\bar{v},t}^{bw}$ stands for the action to instantiate virtual link $\bar{u}\bar{v}$ in networks. Essentially, each action $a_{s,t}^d$ for deploying SFC $s = \bar{s} - f_1 - f_2 - \cdots - f_k - \bar{d}$ in networks is a $1\times 2(k+1)$ matrix vectors, and can be derived as

$$a_{s,t}^d = \cup_{f_i\in s,\bar{u}\bar{v}\in s}\{a_{f_i,t}^{cpu}, a_{f_i,t}^{mem}, a_{\bar{u}\bar{v},t}^{bw}\}. \tag{13}$$

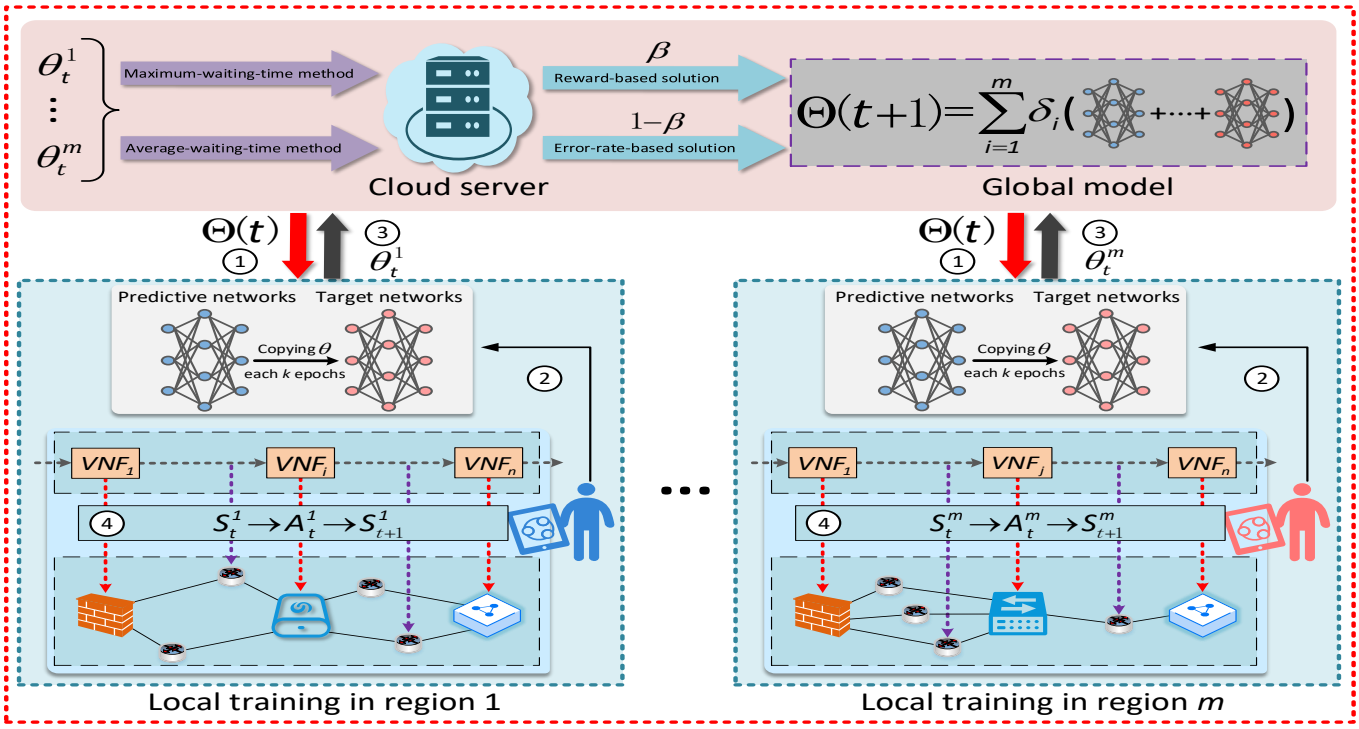


Fig. 2: The training architecture of SSCO, which introduces two categories of training: local training and aggregation training to realize scalable SFC orchestration.

The second action subspace A_t^r is mainly associated with the available CPU, memory, and bandwidth of nodes and links. It includes a set of actions of VNF resource perception $A_t^{r,p}$ and allocation $A_t^{r,a}$, given by

$$A_t^r =< A_t^{r,p}, A_t^{r,a} > . \tag{14}$$

Specifically, $A_t^{r,p}$ can be defined as

$$A_t^{r,p} =< A_{u,t}^{cpu}, A_{u,t}^{mem}, A_{uv,t}^{bw}, \forall u \in V, uv \in E >, \tag{15}$$

where $A_{u,t}^{cpu}$,$A_{u,t}^{mem}$ and $A_{uv,t}^{bw}$ stand for the actions to apperceive the remaining CPU, memory, and bandwidth of node $u \in V$ and link $uv \in E$. Another action $A_t^{r,a}$ can be denoted as

$$A_t^{r,a} =< A_{u,t}^{f_i,cpu}, A_{u,t}^{f_i,mem}, A_{uv,t}^{\bar{u}\bar{v},bw}, \forall f_i \in F, \bar{u}\bar{v} \in L > . \tag{16}$$

Both actions $A_{u,t}^{f_i,cpu}$ and $A_{u,t}^{f_i,mem}$ are taken at node u with the CPU and memory of $C_u^{cpu}[1-r_u^{cpu}(t-1)]$ and $C_u^{mem}[1-r_u^{mem}(t-1)]$, respectively, while action $A_{uv,t}^{\bar{u}\bar{v},bw}$ is taken in link uv with the bandwidth of $C_{uv}^{bw}[1-r_{uv}^{bw}(t-1)]$. Each action $a_{s,t}^r$ for resource perception and allocation of SFC $s = \bar{s}-f_1-f_2-\cdots-f_k-\bar{d}$ in networks is a $1\times 2(k+1)$ matrix vectors, denoted by

$$\begin{aligned} a_{s,t}^r = \cup_{f_i\in s,\bar{u}\bar{v}\in s} \{ & a_{f_i,t}^{cpu}, a_{f_i,t}^{mem}, a_{\bar{u}\bar{v},t}^{bw}, \\ & a_{u,t}^{f_i,cpu}, a_{u,t}^{f_i,mem}, a_{uv,t}^{\bar{u}\bar{v},bw} \}. \end{aligned} \tag{17}$$

The **reward** $R_t \in \mathcal{R}$, determined by action A_t and state S_t, is consistent with our optimization goal, defined in Eq. (7), i.e, to minimize the total weighted cost to deploy SFCs in NFV-enabled networks. By decomposing our goal into a series of local epoch decisions, the reward r_t at time t in local regions can be precisely modeled as a function of state s_t and action a_t, given by

$$r_t = -[\sigma_1 c_{set}(t) + \sigma_2 c_{op}(t) + \sigma_3 c_{com}(t)]. \tag{18}$$

Here, $c_{op}(t)$, $c_{com}(t)$ and $c_{set}(t)$, defined in Eqs. (4), (5) and (6), represent the cost of VNF operation, traffic transmission and VNF setup in local regions, respectively.

B. Training Framework Design

The main idea of SSCO is to design an intelligent framework to implement a scalable orchestration of SFCs, by utilizing FL to train a global SFC orchestration model with dynamically available resources and deep reinforcement learning to find the local cost-efficient model to deploy VNFs in networks.

Fig. 2 illustrates the training architecture of SSCO, which includes *local training* and *aggregation training*. There are two categories of nodes: agents and cloud server, who will perform local training and aggregation training, respectively. The whole network has been divided into m network regions, determined by network scale, node density, the locations of potential clients, and SFC requests, each of which has a client served as a function node to communicate with the cloud server. This means that there are m local clients c_1, c_2, $\cdots$, c_m as local agents to perform local training. The local training is online learning combined with Q-learning and neural networks to update local model θ_t^i $(1 \le i \le m)$, related to distributed resource perception and local VNFs deployment, within a time interval, and report it to the cloud server. Each client will perform the

local training in its region, and share its model with the cloud server rather than other clients. The aggregation training is to collect all local training models θ_t^1, θ_t^1, $\cdots$, θ_t^m to build a new global model $\Theta(t)$ and then send it to local clients for further cost-efficient VNFs deployment with the available resources at a quick convergence of learning rate.

There are two different timescales in our architecture for model training: federated episodes and epochs, as illustrated in Fig. 3. A federated episode with the duration of T is used to mark the time when the cloud server updates $\Theta(t)$, and consists of a series of epochs τ_1,τ_2,$\cdots$,τ_k with the duration of τ, i.e, T=$\cup_{1\leq i\leq k}\{\tau_i\}$ and $|\tau_i|$=$|\tau_{i+1}|$=τ. The epoch is designed for each client c_i to train local model θ_t^i for $\Theta(t)$ update. In order to reduce the communication cost, the data exchange between cloud server and clients is allowed just at the first time epoch τ_1 and the final epoch τ_k in each federated episode.

In order to rapidly fulfill the requirements of different applications, two delay-aware solutions, i.e, maximum-waiting-time and average-waiting-time explorations, have been developed to pursue a tradeoff between the waiting time of cloud server and the learning time of each local agent, for global model training for SFC deployment in various application scenes, which can tolerate the model update period of $\mathcal{T}$. The maximum-waiting-time solution is designed to collect all local models, in the maximum waiting time with the duration of [ζ, 1]T, for the global federated model update in delay-insensitive scenes. Meanwhile, the average-waiting-time solution is used to aggregate local models from a set of all clients in the average waiting time, with the duration of (0, ζ)T, for delay-sensitive scenes with enough accuracy.

The operation of SSCO is generally divided into four steps: ① model propagation, ② local iteration, ③ model aggregation and ④ VNFs allocation, as shown in Fig. 2. There are different clients conducting scalable orchestration tasks of SFCs in different environments which contains network resource states, VNF deployment actions and our designed rewards. All local clients share the same identical model structure, such that their models can be aggregated by the cloud server. In the first step, the cloud server specifies an initial pre-trained model and sends it to local clients. Each model, consisting of a Q-network with parameters θ_t^i for local clients, learns a policy to deploy VNFs as network state changes. In the second step, each participating client uses local data (e.g., available CPU, bandwidth, and memory in networks) to iteratively train its own model in each episode. It is noteworthy that only updated models, not the original data, will be sent to the cloud server for learning aggregation. In the third step, the cloud server aggregates updated models into a new model $\Theta(t+1)$ through reward-based federated methods with the probability of β or place-error-rate-based federated methods with the probability of $(1-\beta)$, described in Eqs. (19)-(21). The update of model aggregation is dependent on the total network costs and deployment efficiency. In the last step, VNFs are placed in networks according to the current local policy, as shown in Fig. 2. All the above is a repetitive process to promote the evolution of the shared model.

In our FRL-based framework, each client, for example, c_i,

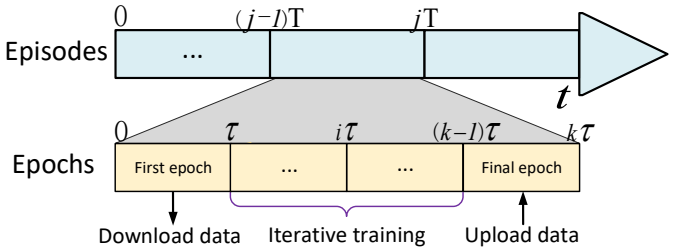


Fig. 3: Illustration of two different timescales.

Algorithm 1: FRL-based SFC Orchestration

Input: Initial model $\Theta(t)$
Output: Improved model $\Theta(t+1)$
Data: Initial local model θ_t^i, initial client weight δ_i

for $t \in [(j-1)T, jT]$ **do**
  Cloud Server does
    Send $\Theta(t)$ to c_i;
  Local client does
  for c_i ***in parallel*** **do**
    foreach τ **do**
      if τ_1 **then**
        $\theta_t^i \leftarrow \Theta(t)$;
      end
      $\nabla L(\theta) \leftarrow$ Network training;
      $\theta_t^i \leftarrow \theta_t^i + \alpha \nabla L(\theta)$; $\Rightarrow$ According to **Algorithm 2**
      if τ_k **then**
        Upload θ_t^i;
      end
    end
  end
  Cloud server does
  if $0 < \mathcal{T} < \zeta T$ **then**
    $m \leftarrow \zeta \times m$;
  end
    $i \in \{1, 2, ..., m\}$;
    Gather δ_i, θ_t^i;
    $\Theta(t+1) \leftarrow \sum_i \delta_i \theta_t^i$;
  return $\Theta(t+1)$.
end

where i is an index of participating clients [37], performs iterative training to obtain a local model θ_t^i and then sends this updated model to the cloud server for global training aggregation. Within a given federated time, client c_i will obtain a local data container, which will include a number of samples. The global updated federated model $\Theta(t+1)$ can be expressed as

$$\Theta(t+1) = \sum_{i=1}^{m} \delta_i \theta_t^i, \tag{19}$$

where δ_i represents the weight of local client c_i to participate in the global model training, and $\sum_i \delta_i$= 1. In order to obtain an optimal global model, there are two weighted strategies, i.e, placement-error-rate-based and reward-based, proposed to set the weights of all clients. The placement-error-rate-based

strategy takes the ratio of invalid placed SFCs to efficient place SFCs of clients as the essential factors to give their weights to participate in the global model training. Let χ_i denote the placement error of client c_i, then δ_i can be calculated as

$$\delta_i = \frac{1-\chi_i}{m - \sum_{i=1}^{m} \chi_i}. \tag{20}$$

The reward-based solution considers that the state-action reward is proportional to their weights to train the global model. Following this principle, δ_i, can be given by

$$\delta_i = \frac{\chi_i}{\sum_{i=1}^{m} \chi_i}, \tag{21}$$

where χ_i denotes the reward of client c_i. All hyperparameters of local training like the client replay buffer size and the number of client epochs will be tuned to optimize the federated model.

The pseudocode of our proposed intelligent framework, which includes local training and aggregation training, is described in **Algorithm 1**. Lines 2-3 and Lines 17-24 of **Algorithm 1** refer to the first step and the third step, respectively. Lines 5-16 of **Algorithm 1** describe the second and last steps of local learning. The local model also directly affects the accuracy and robustness of the predicted requests. We will introduce this model in detail in next section.

C. Local Training via Deep Q-Network

The local training in each region is built on the action-value function $Q(S_t, A_t)$, which is a pair of state S_t and action A_t taken from A_t^d or A_t^r, to find a scalable SFC orchestration solution with DQN. The action-value function $Q(S_t, A_t)$ can be described as

$$Q(S_t = s, A_t = a) = \mathbb{E}[\sum_{k=t}^{\infty} \gamma^{k-t} R_t | S_t = s, A_t = a], \tag{22}$$

where $\gamma(0<\gamma<1)$ stands for the discount factor to indicate the importance of all subsequent rewards in $Q(S_t, A_t)$. The optimal action-value function $Q^*(S_t, A_t)$ follows the rule as

$$\begin{aligned} Q^*(S_t = s, A_t = a) =& \mathbb{E}[R_t + \gamma \max_{a'} Q^*(S_{t+1}, a' \\ & | S_t = s, A_t = a)]. \end{aligned} \tag{23}$$

There are two neural networks, i.e, predictive neural networks and target networks, in DQN. Both networks are designed to train $Q(S_t, A_t; \theta)$ with the built-in parameters θ step by step such that this function can approximate the optimal action-value function $Q^*(S_t, A_t)$, i.e, $Q(S_t, A_t; \theta) \approx Q^*(S_t, A_t)$. Built on Eq. (23), the optimal action-value function can be evaluated following the temporal-difference iterative manner,

$$\begin{aligned} Q(S_t = s, A_t = a) \leftarrow & Q(S_t = s, A_t = a) + \alpha[R_t \\ & + \gamma \max_{a'} Q(S_{t+1}, a') - Q(S_t = s, A_t = a)], \end{aligned} \tag{24}$$

where α is the learning rate.

In order to facilitate understanding, let S_t^i, A_t^i and R_t^i denote the state, action and reward of local training in the i-th region. To precisely learn $Q^*(S_t, A_t)$, the loss function $L(\theta)$, used to estimate the deviation of the predicted value in each local training, has been introduced and defined as

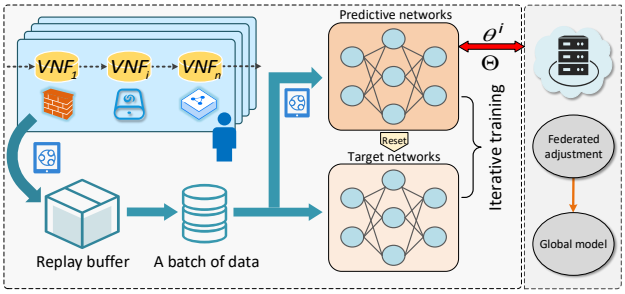


Fig. 4: Training predictive networks and target networks locally.

$$\begin{aligned} L(\theta) = & \mathbb{E}[R_t^i + \gamma \max_{a'} \hat{Q}(S_{t+1}^i, a'; \theta_{t+1}^{i-}) \\ & - Q(S_t^i, A_t^i; \theta_t^i)]^2. \end{aligned} \tag{25}$$

The value of $Q(S_t^i, A_t^i; \theta_t^i)$ is obtained from the predictive neural network, of which θ_t^i is the local training model according to the current state S_t^i and the triggered action (A_t^d, A_t^f). The target network $\hat{Q}$ is introduced to generate target reward for taking action a' in the state S_{t+1}^i, i.e., $G_t^i = R_t^i + \gamma \max_{a'} \hat{Q}(S_{t+1}^i, a'; \theta_{t+1}^{i-})$. In each federated episode, the local model θ_{t+1}^{i-} of target network will copy from θ_t^i at the final time epoch τ_k. In order to minimize the loss function defined in Eq. (25), gradient descent has been used to update the local training model as

$$\begin{aligned} \theta_t^i &= \theta_t^i + \alpha \frac{\partial L(\theta)}{\partial \theta}\Big|_{\theta=\theta_t^i} \\ &= \theta_t^i + \alpha \Big[R_t^i + \gamma \max_{a'} \hat{Q}(S_{t+1}^i, a'; \theta_{t+1}^{i-}) \\ &\quad - Q(S_t^i, A_t^i; \theta_t^i) \Big] \frac{\partial Q(S_t^i, A_t^i; \theta)}{\partial \theta}\Big|_{\theta=\theta_t^i}. \end{aligned} \tag{26}$$

In order to provide more transition samples for the whole training, a replay buffer $\mathcal{D}_i$ with capacity D_i has been introduced into our local model θ_t^i, as shown in Fig. 4, following the original training concept of DQN. Generally, all transition samples are stored in $\mathcal{D}_i$ in the form of $< S_t^i, A_t^i, R_t^i, S_{t+1}^i >$, updated on the first-come-first-served basis, and selected randomly by action space for neural networks ($<S_t^i, A_t^i>$) and target networks ($<S_{t+1}^i>$) to train $Q(S_t^i, A_t^i)$. Such an exploring manner is closely interrelated in the state and action, resulting in low convergence in local learning [8]. To address this issue, we design a weight-based sample update and selection solution for replay buffer exploration. The weight of samples is dependent on the loss function, defined in Eq. (25). The samples with small weights will be first replaced by the incoming samples due to data overflow of replay buffer, and later used for local learning to obtain $Q(S_t^i, A_t^i)$ than the samples with higher weights.

In addition, we adopt $\varepsilon-greedy$ policy for ensuring that all actions $<A_t^d, A_t^r> \in \mathcal{A}$ can be selected in two sequential steps. At the beginning of each epoch, the local host management

selects the action A_t^i through a random probability $\varepsilon \in [0,1]$, with a better exploration probability $\varepsilon_{rd} \in (0,1)$. Then, the local client makes a random choice from $\mathcal{A}$ with probability ε, or chooses the greedy policy if $\varepsilon_{rd} \leqslant \varepsilon \leqslant 1$. A small value of ε_{rd} indicates a greater probability to take optimal action $argmax_{a' \in \mathcal{A}} Q(S_t^i, a'; \theta_t^i)$. Therefore, the exploration parameter gradually decreases over time. At the end of each epoch, ε_{rd} is modified to $\varepsilon_{rd} \times \varepsilon_{decay}$ until a stable minimum value ε_{min}, where ε_{decay} denotes the decay rate of ε_{rd}.

The pseudocode of the local training procedure has been lised in **Algorithm 2**. Lines 4-5 of **Algorithm 2** randomly select an action A_t^i using $\varepsilon - greedy$ policy, and a temporary state-action value is calculated in Line 6 of **Algorithm 2**. Line 7 of **Algorithm 2** observes a reward and a new state from the environment. In Lines 8-9 of **Algorithm 2**, the current data tuple is stored in a replay buffer $\mathcal{D}_i$, while later, another data tuple is randomly selected from the same buffer. We obtain the loss function and a local training model in Lines 10-11 of **Algorithm 2** according to Eq. (26). If τ_i is the last epoch in federated time t, we upload training weights to the cloud server in Lines 12-15 of **Algorithm 2**. The new state and the ε parameter are updated for the next iteration in Lines 16-17 of **Algorithm 2**.

Algorithm 2: Local Online Training via DQN

Data: Initial predictive neural network Q, $< \mathcal{S}, \mathcal{A}, \mathcal{R} > \Rightarrow$ Eqs. (10)-(18), target network $\hat{Q} \leftarrow Q$, replay buffer $\mathcal{D}_i$, random probability ε_{rd}

1 **for** $t \in [(j-1)T, jT]$ **do**
2 **for** c_i ***in parallel*** **do**
3 **foreach** τ **do**
4 Get random probability ε;
5 Select action A_t^i; $\Rightarrow \varepsilon - greedy$
6 $Q(S_t^i, A_t^i; \theta_t^i) \leftarrow < S_t^i, A_t^i >$;
7 Obtain reward R_t^i and reach new state S_{t+1}^i;
8 Put $< S_t^i, A_t^i, R_t^i, S_{t+1}^i >$ into $\mathcal{D}_i$;
9 Choose a batch of training data from $\mathcal{D}_i$;
10 $G_t^i \leftarrow R_t^i + \gamma \max_{a'} \hat{Q}(S_t^i, a'; \theta_{t+1}^{i-})$;
11 $\theta_t^i \leftarrow \theta_t^i + \alpha[G_t^i - Q(S_t^i, A_t^i; \theta_t^i)] \nabla Q$;
12 **if** τ_k **then**
13 $\hat{Q} \leftarrow Q$;
14 Upload θ_t^i;
15 **end**
16 $S_t^i \leftarrow S_{t+1}^i$;
17 $\varepsilon_{rd} \leftarrow \varepsilon_{rd} \times \varepsilon_{decay}$ until ε_{min}.
18 **end**
19 **end**
20 **end**

V. Experimental Evaluation and Results

In this section, we establish and conduct extensive simulations on a universal server, equipped with an Intel(R) Core(TM) i5-8300 CPU 2.30 @ GHz and a Nvidia GeForce GTX 1080Ti GPU, to evaluate the performance of our proposed approach. First, we describe the experimental setup, and then present our evaluation metrics. Finally, we illustrate the results of performance comparison among SSCO, Neural Combinatorial Optimization (NCO) [12], and Branch and Bound (BAB) [38].

A. Simulation Setup

The networks used for simulation experiments are similar to Geant project [19], which includes 10×5 nodes and 45×5 links. The whole network topology is divided into 5 network regions, each of which has a client as a function node to communicate with the cloud server. This means that 5 nodes will be chosen as the function nodes to instantiate SFCs through FRL and the other 45 nodes are acted as the forwarding nodes. There is no data exchange between clients. The experiments are performed in two scenarios of different scales, i.e., a large and a small scenario. In the large scenario, the capabilities of CPU and memory of each node and bandwidth of each link are set to be [12, 16] Cores, [12, 16] GB and 6 Gbps [6], respectively. While in the small scenario, the capabilities of the same resources are set to [6, 10] Cores, [4, 8] GB and 3 Gbps, respectively.

TABLE II: Descriptions of the VNF Parameters

Resource / VNFs	CPU (Core)	Memory (Gb)	Bandwidth (Gbps)
VNF_1	1	1.0	[0.3,0.6]
VNF_2	1	1.5	[0.3,0.6]
VNF_3	2	1.0	[0.3,0.6]
VNF_4	2	1.5	[0.3,0.6]
VNF_5	2	2.0	[0.3,0.6]
VNF_6	3	1.5	[0.3,0.6]
VNF_7	3	2.0	[0.3,0.6]
VNF_8	4	2.0	[0.3,0.6]

In our simulation experiments, there will be 8 types of VNFs included in different SFCs [12]. Each VNF can be placed in one physical node only. The capacity of CPU and memory of each VNF is [1, 4] Core and [1.0, 2.0] GB, respectively. The bandwidth capacity of each virtual link is set between 0.3 Gbps and 0.6 Gbps. There are 128 SFCs, either come from pseudo-true dataset [39] or generated randomly, in the simulations, which is accompanied by a large number of resource consumptions used to schedule VNFs into various nodes [6]. In order to simulate the traffic fluctuation, the resource demands of a SFC are randomly generated from the sum of Gaussian functions with different parameters. Unless specially noted, the lifetime of each SFC is subject to the normal distribution with an average of 10 epoches, and the length of SFC $|s|$ is set between 12 and 18. The pre-defined VNF parameters are listed in Table II.

The experiments are executed in TensorFlow-gpu 1.10 version to implement the neural networks. In our neural networks, there are two fully connected layers with 800 neurons activated by ReLU function each layer, and the target Q-network is replaced each episode. The parameters of ε-greedy rule are set to $\varepsilon_{min} = 0.01$ and $\varepsilon_{decay} = 0.95$ [11]. The discount factor γ in Eq. (22) and learning rate α are set to be 0.99 and 0.0001, respectively. The capability of each replay buffer D_i is

TABLE III: Parameter Settings in Neural Networks

Hyperparameters	Descriptions	Value
α	the learning rate	0.0001
γ	the discount factor	0.99
ε_{min}	the minimal value of ε	0.01
ε_{decay}	the decay rate of ε	0.95
D_i	the capability of replay memory	10000
$-$	the capability of batch size	64

10000, and each update of predictive networks applies a batch of 64 samples. Following the training framework design, the 3-Tuple settings are as follows:

- The input state S_t at time episode t at each client can be given as a high-dimensional vector, which refers to all network resources and configurations as $S_t{=<} C_u^{cpu}(t)$, $C_{u,f_i}^{cpu}(t)$, $C_u^{mem}(t)$, $r_u^{cpu}(t)$, $r_u^{mem}(t)$, $r_{uv}^{bw}(t)$, $C_{u,f_i}^{mem}(t)$, $C_{uv}^{bw}(t)$, $C_{uv,s_i}^{bw}(t) >$ in the dimension of $10 \times 4 + 45 \times 2 = 130$.
- The output action $A_t{=<} A_t^d, A_t^r >$ at time episode t at each client consists of resource perception and allocation in the dimension of $10 + 45 = 55$.

Most parameter settings associated with neural networks are listed in Table III [11].

In order to qualitatively execute evaluations, in addition to SSCO, we have also implemented two related solutions: NCO and BAB, described below.

- NCO: NCO is a dynamic DRL-based VNF placement scheme built on neural combinatorial optimization theory. Considering the constraints that indicate the restrictions of network infrastructure and services, NCO formulates the VNF placement as a constrained combinatorial optimization problem with the aim of minimizing the overall power consumption, and addresses it by exploring the NFV infrastructure and learning optimal placement policy. The placement policy is trained through Long Short-term Memory (LSTM), which is further adopted to the RL agent architecture with policy gradients methods.
- BAB: BAB is a static NFV placement scheme based on the classical Breadth First Search. It considers all actions of SFC deployment satisfying Eqs. (8) and (9) as a solution space tree. At a given state, each action will be checked against the upper and lower estimated bounds on the optimal solution by a branch along a space tree. Finally, BAB explores the branches of candidate actions, which represent the subsets of the solutions, to seek a series of optimal actions for efficient NFV placement in networks.

B. Evaluation Metrics

There are four evaluation metrics, i.e., loss function, network cost, resource utilization ratio, and placement error ratio, considered in all scenarios to investigate the performance of SSCO.

- **Loss function:** The loss function $L(\theta)$ defined in Eq. (25) is used to indicate the convergence performance of a DQN model. The smaller the loss function is, the better estimation performance a DQN model will have.
- **Network cost:** The network cost refers to the total cost of VNF operations, setup and communications in networks in each training epoch according to local rewards, and can be defined as

$$C_t = \sum_{i=1}^{m} |R_t^i|. \tag{27}$$

With the federated framework employed in our approach, the environments will obtain an optimal reward, since it merely transmits learning models instead of client data.
- **Resource utilization ratio:** This metric includes $r_u^{cpu}(t)$, $r_u^{mem}(t)$, and $r_{uv}^{bw}(t)$, defined in Eq. (3), and can be used to measure the available resource of each node $u{\in}V$ and link $uv{\in}E$ to host SFCs.
- **Placement error ratio:** This metric P_e is referred to as the ratio of the number of invalid SFC deployment N_i to the number of efficient SFC deployment N_e in networks, and can be defined as

$$P_e = \frac{N_i}{N_e}. \tag{28}$$

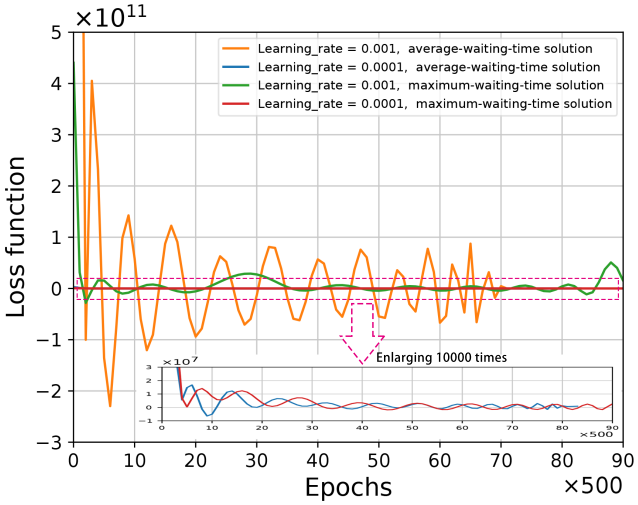
It is used to indicate the errors of SFC placement due to resource exhaustion of physical nodes and links to host all VNFs and their virtual links connecting them.

C. Simulation Results

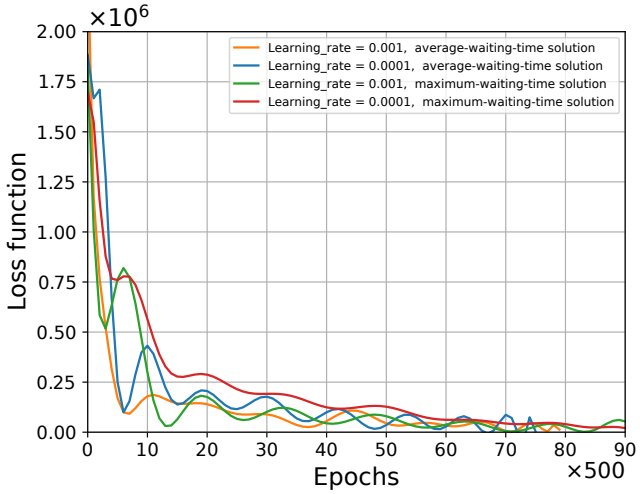
1) Convergence of SSCO in Local Training: Fig. 5 evaluates the convergence of SSCO, in the form of loss function, during local training process over 45000 epochs in different scenarios. The length of SFCs is set to 12. Both maximum-waiting-time and average-waiting-time explorations, with the duration of [0.6, 1]T and (0, 0.6)T, respectively, have been exploited to train the global model. The learning rates are set to either 0.001 or 0.0001.

Fig. 5 shows that the loss function of SSCO, which introduces two time-based training explorations to global model training, decreases closely to 0 when the learning rate is set to be a reasonable value, for example, 0.0001, meaning that all local training tends to converge. The final achieved convergence indicates that the local DQN model has learnt the hidden rules for evaluating actions. Fig. 5(a) illustrates that the local DQN model has obtained an optimal learning rate, which is equal to 0.0001, and thus is trained to be convergent after about 4000 training epochs in the large scenario with larger resources on nodes and links. In contrast, in the small scenario as depicted in Fig. 5(b), the local DQN model gradually converges after around 5500 epochs, where the convergence rate drops slightly under a tightly constrained resource. However, since the training time in both of the above scenarios is short enough to be ignored, SSCO can achieve efficient model training and update in the limited time.

2) Impact of Parameters of Federated Learning: Fig. 6 illustrates the network cost of SSCO with different federated learning cycle and federated learning weights. The federated learning cycle T, elaborated in Fig. 3, refers to a training

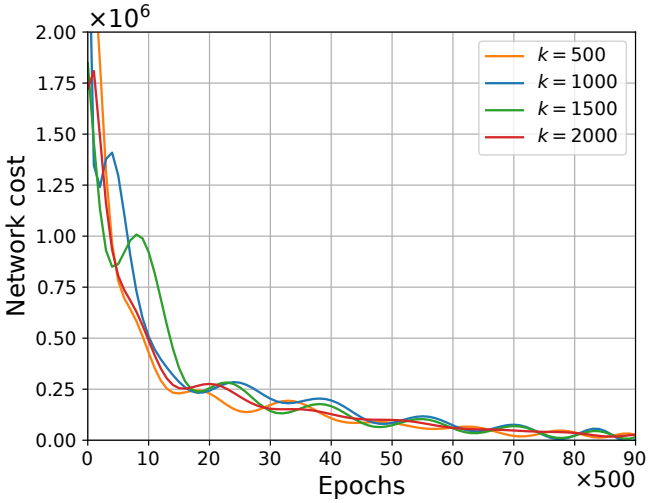


(a) Convergence of SSCO with different training round in large scenarios

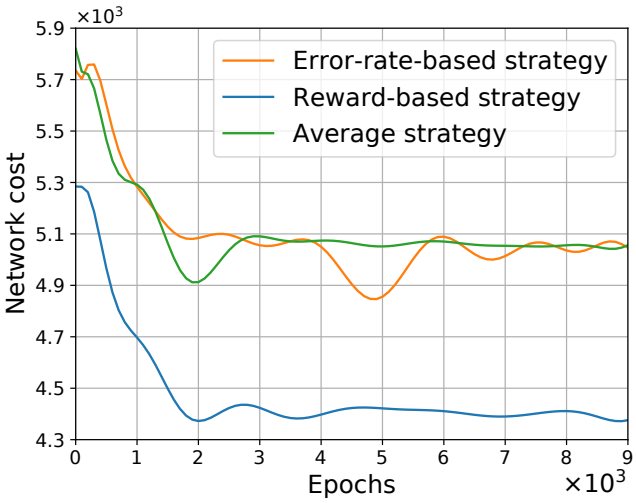


(b) Convergence of SSCO with different training round in small scenarios

Fig. 5: Convergence of SSCO in different scenarios.



(a) Impact of federated learning cycle



(b) Impact of federated learning weights

Fig. 6: Impact of parameters of federated learning.

episode, in which the cloud server needs to update a global training model for local model training. As stated, each episode includes k epochs, one of which denotes a round of local training. There are three basic strategies of federated learning weights, i.e., average, reward-based and error-rate-based strategies, designed for global model training in our simulations. The first strategy gives clients the same weights, while the second and last one tend to assign a higher weight to the clients owning better reward and lower placement error rate, respectively.

Fig. 6(a) demonstrates the network cost of our proposed SSCO with different federated learning cycles, in which k is set to be 500, 1000, 1500 and 2000, respectively. It is clear to see that the federated learning cycle, which is proportional to k, the number of epochs, has a great impact on network cost during the federated training of our scheme. This is because the duration of local model update cycle affects the convergence of the reward of local training. Large duration of cycles may ignore the global optimum of local learning, while small duration of cycles will cause federated aggregation ahead of local DQN convergence. It is noticeable in Fig. 6 (a) that SSCO has gradually achieved a better network cost after about 5000 training epochs. This means that such a learning speed is adequate and has reached the convergence range of local learning.

Fig. 6 (b) illustrates the network cost of our proposed SSCO with different federated learning weights of clients during local training process over 9000 epochs. Five local clients have been assigned different weights to train the global model, depending on their state-action reward, VNF placement error ratio and specific functions. It is clear to observe that the reward-based federated learning strategy can greatly reduce the network cost,

with up to 11.3% and 13.7% reductions, compared with the error-rate-based and average federated learning strategies. This is because the clients with more state-action rewards will be assigned more weights to train global model.

3) Comparison in Resource Utilization Ratio: Fig. 7 shows the resource utilization ratio of nodes and links in terms of CPU, memory, and bandwidth with different SFCs. Figs. 7 (a), (b), (d) and (e) demonstrate the utilization ratio of node resources of SSCO and NCO as the length and the number of SFCs varies from 12 to 18 and 2 to 8, respectively. The results illustrate that SSCO achieves the maximal utilization in most epochs as desired, meaning that a great federated model has been trained to well utilize the limited resources of nodes and links for SFC deployment in networks. For example, SSCO has up to 50% and 51% resource utilization ratio with respect to CPU and memory of nodes when the length of SFC is set to 14, compared with NCO. NCO almost always performs the worse, but occasionally with a better performance. This is because NCO always cannot take better actions to update the state-value function, with Monte Carlo policy which relies on all samples to work correctly. Figs. 7 (c) and (f) indicate the utilization ratio of bandwidth as the length and the number of SFCs increase, respectively. It shows that, SSCO leads NCO with an average gap of 5.58% in bandwidth as the SFC length increases from 12 to 18 and performs better when there are a mass of SFCs. This is because our well-trained SSCO can save more bandwidth for its effective adjusting on traffic resources by federated learning to make a more appropriate deployment, while NCO lacks consideration of this traffic feature. In addition, it's worth noting that SSCO finally achieves a better resource utilization with the increasing SFCs, which further confirms the scalability of SSCO in a wide range of different traffic distributions.

4) Comparison in Placement Error Ratio and Network Cost: Fig. 8 demonstrates the placement error ratio obtained by SSCO, NCO and BAB as the length and the number of SFCs increase in both large and small scenarios. It can be seen from Figs. 8(a) and (c) that SSCO and NCO vastly decrease placement errors than BAB when the length and the number of SFCs ranges from 12 to 18 and 2 to 8, respectively. This is due to that more feasible actions have been taken to orchestrate even-increasing SFCs. There is slight increase on placement error ratio for SSCO compared with NCO when the number of SFCs changes from 6 to 8. This is because there are ever-increasing SFCs to be instantiated, but more resources such as CPU, memory and bandwidth already have been used to host SFCs for SSCO, as shown in Fig. 7. Figs. 8 (b) and (d) indicates the network cost of BAB, NCO and SSCO for services which are successfully deployed as the length and the number of SFCs increase. Obviously, BAB holds the highest network cost at each length and each number of SFCs, which is according to the performance shown in Figs. 8 (b) and (d). In contrast, SSCO performs best with the lowest network cost while NCO holds the medium one. This is because SSCO takes federated learning into consideration and the return of aggregation at each federated epoch helps clients optimizing their local neural networks instead of using the original ones. Furthermore, it is apparent that the range between upper and lower bounds of SSCO is smaller than NCO and BAB, which indicates that SSCO shows stronger stability and scalability than others.

TABLE IV: The Complexity of SSCO and NCO

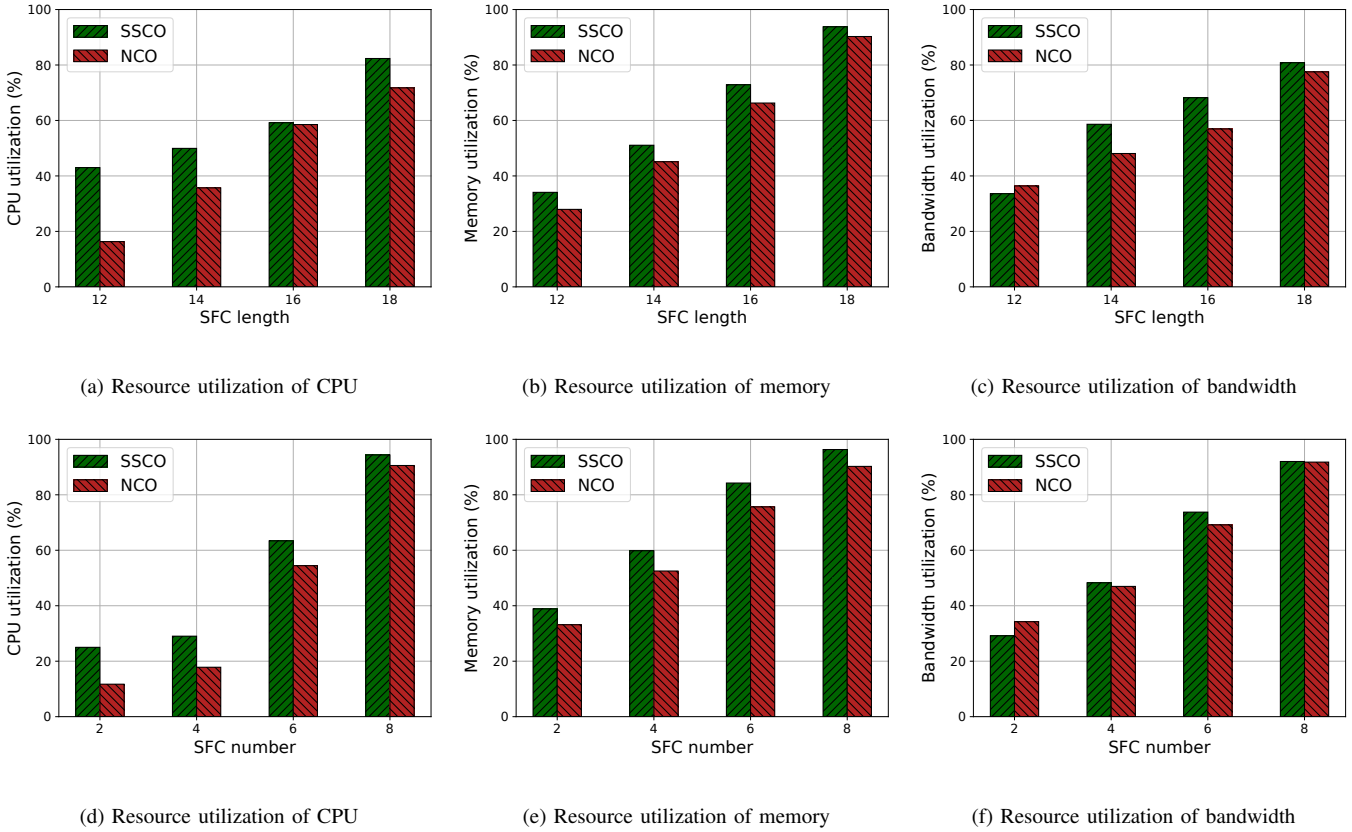
Solutions	Time Complexity
SSCO	$\mathcal{O}[IP(\|V\|+\|E\|)(\|V\|+\|E\|+P)DK+JM]$
NCO	$\mathcal{O}[I(P(\|V\|+\|E\|)(\|V\|+\|E\|+P)+P^2(\|V\|+\|E\|)^2)]$

5) Time Complexity: The time complexity of SSCO refers to aggregated learning and local learning. The time complexity of local learning is depended on the structure of neural networks and the size of state and action spaces. Let P denote the number of VNF requests at each epoch. Notice that the network resources provided by node $u \in V$ include CPU and memory while the configurations of link $uv \in E$ mainly refer to bandwidth resources, thus the number of neurons in the input layer, depending on the size of state space, can be expressed as $2|V|+|E|+P$. Because the output layer of neural networks is related to the action space, then the number of neurons in output layer can be given by $(P+1)(|V|+|E|)$. Let K and D be the number of middle layers in a DQN model and neurons in each hidden layer, respectively. Built on the operation principle of DQN, the time complexity of each local iteration can be expressed as $\mathcal{O}[P(|V|+|E|)(|V|+|E|+P)DK]$. In addition, let J and M stand for the federated aggregation times and the number of clients, respectively. Following the operation principle of FL, the time complexity of the aggregated process can be derived as $\mathcal{O}(JM)$. Thus, the total time complexity of SSCO is $\mathcal{O}[IP(|V|+|E|)(|V|+|E|+P)DK+JM]$, where I represents the local training rounds.

As for NCO, the agent is a sequence-to-sequence model, which is formed by an encoder-decoder design based on stacked LSTM cells [12]. The time complexity of NCO can be regarded as $\mathcal{O}[I(P(|V|+|E|)(|V|+|E|+P)+P^2(|V|+|E|)^2)]$, as shown in Table IV.

VI. Conclusions

In this paper, a novel scalable SFC orchestration scheme with the recent advances in FRL, namely SSCO, has been proposed to dynamically fulfill the ever-increasing requirements of different network applications. The proposed joint framework involves online federated training and deep reinforcement learning for scalable SFC deployment in NFV-enabled networks, which can realise desired network services as far as possible. Both placement-error-rate-based and reward-based federated weighed strategies have been designed for local agents to participate in global model training in an optimal iterative manner with quick convergence. Furthermore, the loss-weight-based mechanism has been proposed to generate and exploit reference samples in replay buffer for future training, and are devoted to advancing the convergence performance of the proposed scheme. Simulation results show that SSCO exhibits the better convergence performance, higher average reward, and smaller average resource consumption than other reference policies over a variety of network scenarios, which is coincident with our expected objective.



(a) Resource utilization of CPU (b) Resource utilization of memory (c) Resource utilization of bandwidth

(d) Resource utilization of CPU (e) Resource utilization of memory (f) Resource utilization of bandwidth

Fig. 7: Resource utilizations of SSCO with different SFCs.

As a part of our future investigation, we will focus on the implementation of SSCO on large-scale substrate networks and practical NFV platforms, for example, OPNFV, especially when constraints cannot be satisfied simultaneously, for further increasing the scalability and feasibility of our model. Considering that the full NFV-based networks have not been built in telecommunications, we hope to further extend our work into hybrid NFV-based networks, where the full NFV-based networks and traditional hardware-based networks will coexist. The combination of them brings new opportunities to fulfill the ever-increasing requirements of network applications with less CAPEX and OPEX. To this end, the potential directions include FRL acceleration, resource prediction and hybrid SFC recombining built on our current investigations.

REFERENCES

[1] R. Yu, G. Xue, V. T. Kilari, and X. Zhang, "Network function virtualization in the multi-tenant cloud," *IEEE Netw.*, vol. 29, no. 3, pp. 42–47, 2015.

[2] A. M. Medhat, T. Taleb, A. Elmangoush, G. A. Carella, S. Covaci, and T. Magedanz, "Service function chaining in next generation networks: State of the art and research challenges," *IEEE Commun. Mag.*, vol. 55, no. 2, pp. 216–223, 2017.

[3] A. Fischer, J. F. Botero, M. T. Beck, H. de Meer, and X. Hesselbach, "Virtual network embedding: A survey," *IEEE Commun. Surveys Tuts.*, vol. 15, no. 4, pp. 1888–1906, 2013.

[4] M. H. Gao, B. Addis, M. Bouet, and S. Secci, "Optimal orchestration of virtual network functions," *Comput. Netw.*, vol. 142, pp. 108–127, Jun. 2018.

[5] M. Mechtri, C. Ghribi, O. Soualah, and D. Zeghlache, "NFV orchestration framework addressing SFC challenges," *IEEE Commun. Mag.*, vol. 55, no. 6, pp. 16–23, 2017.

[6] J. Pei, P. Hong, M. Pan, J. Liu, and J. Zhou, "Optimal VNF placement via deep reinforcement learning in SDN/NFV-enabled networks," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 2, pp. 263–278, 2020.

[7] S. Sahhaf *et al.*, "Network service chaining with optimized network function embedding supporting service decompositions," *Comput. Netw.*, vol. 93, pp. 492–505, Oct. 2015.

[8] A. Leivadeas, G. Kesidis, M. Falkner, and I. Lambadaris, "A graph partitioning game theoretical approach for the VNF service chaining problem," *IEEE Trans. Netw. Serv. Manag.*, vol. 14, no. 4, pp. 890–903, 2017.

[9] Y. Xiao *et al.*, "NFVdeep: Adaptive online service function chain deployment with deep reinforcement learning," in *Proc. Int. Symp. Qual. Service (IWQoS)*, 2019, pp. 1–10.

[10] S. Rahman, T. Ahmed, M. Huynh, M. Tornatore, and B. Mukherjee, "Auto-scaling VNFs using machine learning to improve QoS and reduce cost," in *Proc. IEEE Int. Conf. Commun. (ICC)*, 2018, pp. 1–6.

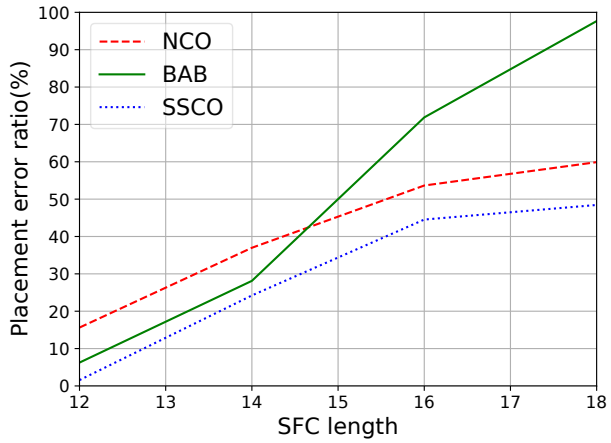
[11] X. Xiong, K. Zheng, L. Lei, and L. Hou, "Resource allocation based on deep reinforcement learning in IoT edge computing," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 6, pp. 1133–1146, 2020.

[12] R. Solozabal, J. Ceberio, A. Sanchoyerto, L. Zabala, B. Blanco, and F. Liberal, "Virtual network function placement optimization with deep reinforcement learning," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 2, pp. 292–303, 2020.

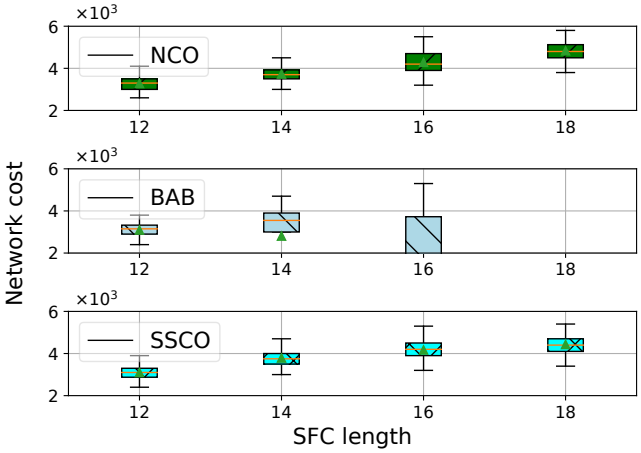
[13] L. Gu, D. Zeng, W. Li, S. Guo, A. Zomaya, and H. Jin, "Deep reinforcement learning based VNF management in geo-distributed edge computing," in *Proc. IEEE Int. Conf. Distrib. Comput. Syst. (ICDCS)*, 2019, pp. 934–943.

[14] S. Troia, R. Alvizu, and G. Maier, "Reinforcement learning for service function chain reconfiguration in NFV-SDN metro-core optical networks," *IEEE Access*, vol. 7, pp. 167 944–167 957, 2019.

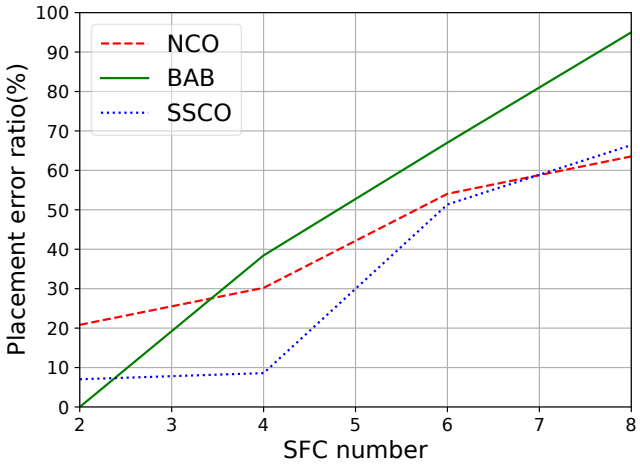
[15] S. Troia *et al.*, "Machine learning-assisted planning and provisioning for SDN/NFV-enabled metropolitan networks," in *Proc. Eur. Conf. Netw. Commun. (EuCNC)*, 2019, pp. 438–442.



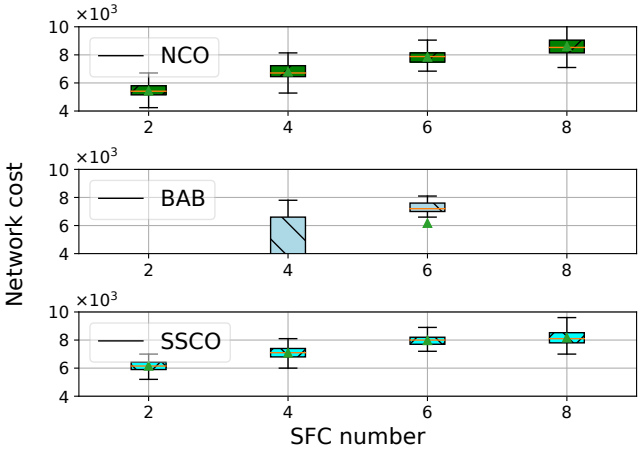
(a) Placement error ratio with different SFC length



(b) Network cost with different SFC length



(c) Placement error ratio with different SFC number



(d) Network cost with different SFC number

Fig. 8: Placement error and network cost with different SFCs

[16] S. Lee and D. H. Choi, "Federated reinforcement learning for energy management of multiple smart homes with distributed energy resources," *IEEE Trans. Ind. Informat.*, pp. 1–1, 2020.

[17] B. Yi and X. Wang and K. Li and S. K. Das and M. Huang, "A comprehensive survey of network function virtualization," *Comput. Netw.*, vol. 133, pp. 212–262, Mar. 2018.

[18] A. Laghrissi and T. Taleb, "A survey on the placement of virtual resources and virtual network functions," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 2, pp. 1409–1434, 2019.

[19] F. Bari, S. R. Chowdhury, R. Ahmed, R. Boutaba, and O. C. M. B. Duarte, "Orchestrating virtualized network functions," *IEEE Trans. Netw. Serv. Manag.*, vol. 13, no. 4, pp. 725–739, 2016.

[20] V. Eramo, E. Miucci, M. Ammar, and F. G. Lavacca, "An approach for service function chain routing and virtual function network instance migration in network function virtualization architectures," *IEEE/ACM Trans. Netw.*, vol. 25, no. 4, pp. 2008–2025, 2017.

[21] B. Zhang, J. Hwang, and T. Wood, "Toward online virtual network function placement in software defined networks," in *Proc. Int. Symp. Qual. Service (IWQoS)*, 2016, pp. 1–6.

[22] R. Zhou, "An online placement scheme for VNF chains in geo-distributed clouds," in *Proc. Int. Symp. Qual. Service (IWQoS)*, 2018, pp. 1–2.

[23] H. Tang, D. Zhou, and D. Chen, "Dynamic network function instance scaling based on traffic forecasting and VNF placement in operator data centers," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 3, pp. 530–543, 2019.

[24] H. Moens and F. De Turck, "VNF-P: A model for efficient placement of virtualized network functions," in *Proc. 10th Int. Conf. Netw. Service Manage. (CNSM)*, 2014, pp. 418–423.

[25] L. Gu *et al.*, "Fairness-aware dynamic rate control and flow scheduling for network utility maximization in network service chain," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 5, pp. 1059–1071, 2019.

[26] H. B. McMahan, "A survey of algorithms and analysis for adaptive online learning," *J. Mach. Learn. Res.*, vol. 18, no. 1, pp. 3117–3166, Jan. 2017.

[27] J. Liu, W. Lu, F. Zhou, P. Lu, and Z. Zhu, "On dynamic service function chain deployment and readjustment," *IEEE Trans. Netw. Serv. Manag.*, vol. 14, no. 3, pp. 543–553, 2017.

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

[29] X. Zhang, C. Wu, Z. Li, and F. C. M. Lau, "Proactive VNF provisioning with multi-timescale cloud resources: Fusing online learning and online optimization," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, 2017, pp. 1–9.

[30] X. Fei, F. Liu, H. Xu, and H. Jin, "Adaptive VNF scaling and flow routing with proactive demand prediction," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, 2018, pp. 486–494.

[31] J. Wang, J. Hu, G. Min, W. Zhan, Q. Ni, and N. Georgalas, "Computation offloading in multi-access edge computing using a deep sequential model based on reinforcement learning," *IEEE Commun. Mag.*, vol. 57, no. 5, pp. 64–69, 2019.

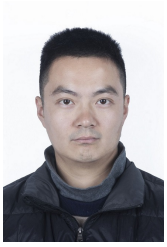
[32] V. Mnih et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–33, 2015.

[33] B. Kiumarsi, K. G. Vamvoudakis, H. Modares, and F. L. Lewis, "Optimal

and autonomous control using reinforcement learning: A survey," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 6, pp. 2042–2062, 2018.
[34] J. Wang, J. Hu, G. Min, A. Y. Zomaya, and N. Georgalas, "Fast adaptive task offloading in edge computing based on meta reinforcement learning," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 1, pp. 242–253, 2021.
[35] P. Tang, F. Li, W. Zhou, W. Hu, and L. Yang, "Efficient auto-scaling approach in the telco cloud using self-learning algorithm," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, 2015, pp. 1–6.
[36] S. Draxler, H. Karl, and Z. A. Mann, "Joint optimization of scaling and placement of virtual network services," in *Proc. IEEE/ACM Int. Symp. Cluster, Cloud Grid Comput. (CCGrid)*, 2017, pp. 365–370.
[37] A. Hard *et al.*, "Federated learning for mobile keyboard prediction," 2018, *arXiv:1811.03604*. [Online]. Available: http://arxiv.org/abs/1811.03604
[38] G. T. C. Schulte, M. Lagerkvist, "Gecode," [Online], http://www.gecode.org.
[39] S. Orlowski, R. Wessäly, M. Pióro, and A. Tomaszewski, "Sndlib 1.0—Survivable network design library," *Networks*, vol. 55, no. 3, pp. 276–286, May 2010.

Haojun Huang is an Associate Professor in the School of Electronic Information and Communications at Huazhong University of Science and Technology, China. He received his PhD degree in Communication and Information Engineering from the University of Electronic Science and Technology of China in 2012, and the BS degree in Computer Science from Wuhan University of Technology in 2005. His current research interests include Internet of Things, Network Function Virtualization, Software-Defined Networking, and Artificial Intelligence for networking.

Cheng Zeng is currently a Master student in Information and Communication Engineering at Huazhong University of Science and Technology, China. He received the BS degree in Electronic Information Engineering from Wuhan University of Technology, China, in 2019. His research interests include Network Function Virtualization and Federated Learning.



Yangming Zhao is a research scientist with University at Buffalo. He received the BS degree in Communication Engineering and the PhD degree in Communication and Information System from University of Electronic Science and Technology of China in 2008 and 2015, respectively. His research interests include network optimization, data center networks, edge computing and transportation systems.

Geyong Min is a Professor of High Performance Computing and Networking in the Department of Computer Science within the College of Engineering, Mathematics and Physical Sciences at the University of Exeter, United Kingdom. He received the PhD degree in Computing Science from the University of Glasgow, United Kingdom, in 2003, and the BS degree in Computer Science from Huazhong University of Science and Technology, China, in 1995. His research interests include Computer Networks, Wireless Communications, Parallel and Distributed Computing, Ubiquitous Computing, Multimedia Systems, Modelling and Performance Engineering.

Yingying Zhu is currently a Postdoctoral Researcher at Huazhong University of Science and Technology, Wuhan, China (HUST). She received the PhD degree in Communication and Information Engineering and the BS degree in Communication Engineering from HUST in 2018 and 2011, respectively. Her research interests include Computer Vision and Machine Learning.

Wang Miao is currently a Postdoctoral Research Associate in the Department of Computer Science at the University of Exeter, United Kingdom. He received his PhD degree in Computer Science from the University of Exeter, United Kingdom in 2017. His research interests focus on Network Function Virtualization, Software-Defined Networking, Unmanned Aerial Networks, Wireless Communication Networks, Wireless Sensor Networks, and Edge Artifical Intelligence.

Jia Hu is a Senior Lecturer in Computer Science at the University of Exeter. He received his PhD degree in Computer Science from the University of Bradford, UK, in 2010, and MS and BS degrees in Electronic Engineering from Huazhong University of Science and Technology, China, in 2006 and 2004, respectively. His research interests include Edge-cloud Computing, Resource Optimization, Applied Machine Learning, and Network Security.