Parallel Placement of Virtualized Network Functions via Federated Deep Reinforcement Learning

Haojun Huang, Jialin Tian, Geyong Min, Hao Yin, Cheng Zeng, Yangming Zhao, and Dapeng Wu

***Abstract*—Network Function Virtualization (NFV) introduces a new network architecture that offers different network services flexibly and dynamically in the form of Service Function Chains (SFCs), which refer to a set of Virtualization Network Functions (VNFs) chained in a specific order. However, the service latency often increases linearly with the length of SFCs due to the sequential execution of VNFs, resulting in sub-optimal performance for most delay-sensitive applications. In this paper, a novel Parallel VNF Placement (PVFP) approach is proposed for real-world networks via Federated Deep Reinforcement Learning (FDRL). PVFP has three remarkable characteristics distinguishing from previous work: (1) PVFP designs a specific parallel principle, with three parallelism identification rules, to reasonably decide partial VNF parallelism; (2) PVFP considers SFC partition in multi-domains built on their remaining resources and potential parallel VNFs to ensure that VNFs can be reasonably distributed for resource balancing among domains; (3) An FDRL-based framework of parallel VNF placement is designed to train a global intelligent model, with time-variant local autonomy explorations, for cross-domain SFC deployment, avoiding data sharing among domains. Simulation results in different scenarios demonstrate that PVFP can significantly reduce the end-to-end latency of SFCs at the medium resource expenditures to place VNFs in multiple administrative domains, compared with the state-of-the-art mechanisms.**

***Index Terms*—Network function virtualization, parallel placement, federated learning, deep reinforcement learning, multiple domains.**

I. Introduction

NOWADAYS, networks are greatly dependent on a number of specialized network functions (NFs) or middleboxes [1], [2], [3] like Network Address Translation (NAT), Intrusion Detection System (IDS), Firewall (FW), and Load Balancer (LB) to provide diversified services. Because the initial instantiation of such NFs is implemented in the dedicated hardware, it is more costly for telecom operators to launch network services due to the high Capital Expenditure (CAPEX) and Operational Expenses (OPEX). These have seriously impeded the roll-out of new network services and constrained network innovations to fulfill the various requirements of the ever-emerging applications, such as augmented reality, autonomous driving, and smart city. To alleviate this problem, Network Function Virtualization (NFV) [3], [4], [5] has been proposed as a promising initiative by leveraging virtualization technologies. The main goal of NFV is to decouple NFs from the dedicated hardware and implement them on the general-purpose infrastructures in software, known as Virtualized Network Functions (VNFs) [6], to provide network services with lower expenditure and enriched flexibility. Moreover, such a paradigm will greatly benefit vendors to improve the network resilience and availability [2].

In spite of such superiorities, NFV still faces some key issues that should be tackled when the Service Function Chains (SFCs), which refer to a series of VNFs connected in a specific order, are orchestrated in networks. A critical concern is how to efficiently deploy each component of SFCs into appropriate infrastructures with the guarantee of satisfied end-to-end latency. A number of studies have been devoted to traffic scheduling [7], [8], VNF acceleration [9], VNF resource utilisation [10], [11], and VNF instantiation [12] to minimize the end-to-end latency of SFCs. Although preliminary progresses have been made, all these efforts target to reduce the NFV latency in a horizontal scope, i.e., assuming the sequential composition of VNFs and accelerating each component of SFCs horizontally. Such assumptions make the VNF orchestration more tractable, because only the linear objectives and constraints, e.g., the Quality-of-Service (QoS) requirements of a sequential SFC for an NFV service, are taken into consideration and thus the associated optimization problems are typically deterministic. Nevertheless, these may be impractical for specific VNFs like the Caching and NAT, which can be executed in a parallel manner, as shown in Fig. 1, to reduce the NFV latency [13], [14]. In reality, approximately half of the VNF pairs can work in parallel [15] because there is no strict execution sequence among them. So far, only a few traditional heuristics [13], [14], [15] have been proposed to realize parallel placement of VNFs, and only achieved low NFV performance gains as the number of SFCs increases. All of these efforts rely on the prior knowledge to deploy VNFs, but cannot jointly capture the features behind the VNF pairs and the time-variant network states, including network resources and configurations, to take the best actions for parallel SFC orchestration in networks with more NFV latency rewards in an intelligent manner.

Today, Federated Deep Reinforcement Learning (FDRL),

Haojun Huang and Cheng Zeng are with the School of Electronic Information and Communications, Huazhong University of Science and Technology, Wuhan, 430074, China. H. Huang is also with the Department of Computer Science, University of Exeter, Exeter, EX4 4QF, UK. Email: {hjhuang, c_zeng}@hust.edu.cn

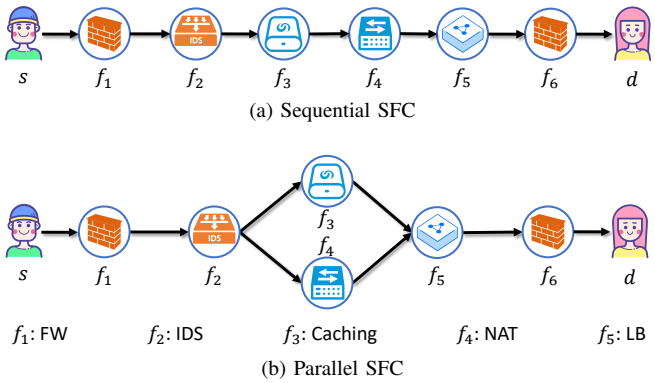
Jialin Tian and Geyong Min are with the Department of Computer Science, University of Exeter, Exeter, EX4 4QF, UK. Email: {jt816, g.min}@exeter.ac.uk.

Hao Yin is with Tsinghua University, Beijing 100084, China (E-mail:h-yin@mail.tsinghua.edu.cn).

Yangming Zhao is with the School of Computer Science and Technology, University of Science and Technology of China, 230026, Heifei, China. Email: zhaoym@ustc.edu.cn.

D. Wu is with the Department of Computer Science, City University of Hong Kong, 999077, Hong Kong. Email: dpwu@ieee.org.

Corresponding author: Haojun Huang and Geyong Min.



(a) Sequential SFC

(b) Parallel SFC

Fig. 1: Sequential SFC v.s. Parallel SFC.

which closely integrates Federated Learning (FL) with Deep Reinforcement Learning (DRL), has achieved notable breakthrough and performance gains in a variety of applications, like autonomous driving, strategic game, routing and network slicing [10], [16], [17], [18], [19]. Because deep neural networks installed in DRL and FL can distill the close relationships among VNFs, network states, taken actions and their performance rewards for SFC orchestration, FDRL has become a promising paradigm to alleviate the above-mentioned issues. With the powerful exploration and exploitation of the historical experiences and the hidden network-related factors, FDRL can well assess the performance obtained from SFC instance actions under different network states, and learn to take better actions for the optimal sequential/parallel deployment of SFCs with greater performance rewards from networks. However, the current applications show that it always suffers from the learning inefficiency problem due to the ever-increasing dimensions of network states and taken actions as the number of SFCs increases, which will hinder the convergence of model training and consume more resources. For instance, the federated cloud server will take a long time to learn a global model of VNF placement, since the ever-increasing SFCs have not been reasonably assigned to the swarm domains built on their available resources or it fails to take into consideration the time-variant autonomy explorations. Furthermore, there are no available parallelism principles integrated with FDRL to identify the independence of all VNFs in their parallel placement. These motivate us to develop novel FDRL-based strategies for VNF orchestration in parallel with the end-to-end latency guarantee under resource constraints.

Therefore, in this paper, we focus on FDRL-based PVFP, a partial VNF parallelism framework that explores the optimal parallel placement of VNFs via FL in global model training and DRL in local model training, respectively, to reduce the end-to-end latency of SFCs. To the best of our knowledge, PVFP is the first work to realize parallel VNF placement via FDRL. The main goal of PVFP is to identify which independent NFs should be parallelized and where VNFs should be instantiated in parallel in networks with explicit consideration of their instance costs. To this end, PVFP first designs three parallelism rules to enumerate all the feasible configurations of partial VNF parallelism based on the dependency of each VNF pair. Then, PVFP allocates the whole SFCs into all domains, considering the available resources of each domain and potential parallel VNF pairs that should be assigned to the same domain to reduce inter-domain information exchange. Finally, a parallel VNF placement model that identifies the minimum end-to-end latency with the designed parallelism principles, has been trained by a set of domain orchestrators and the cloud server, through latency-based and reward-based federated aggregation.

The main contributions of this paper are summarized as follows:

- An FDRL-based framework is proposed to train a global model for parallel VNF placement executed by a set of domain orchestrators and the cloud server, through latency-based and reward-based federated aggregation. As there are different amounts of resources available in each domain, the whole SFCs have been decomposed into all domains built on their remaining resources and potential parallel VNF pairs for global model training with quick convergence. Furthermore, the VNF parallelism principles, referring to three types of parallelism rules, have been designed to guarantee the processing correctness and reduce the SFC latency.
- The latency-based aggregation weights are designed to aggregate all local models with explicit consideration of VNF instance latency and costs. In order to accelerate the convergence of PVFP, a reward-based adaptive ϵ-greedy strategy is adopted to ensure that all actions at each state can be faster explored. Furthermore, a soft target update solution, which slowly tracks the predictive networks, is introduced to update the parameters of target networks to improve the stability of local learning and accelerate its convergence. Besides, a flexible replay buffer, which is scaled to the available resources of each domain, is proposed to store reference experiences for further training.
- Extensive simulation experiments and numerical analyses are carried out under various network configurations to estimate the effectiveness and efficiency of our proposed solution. Simulation results demonstrate that PVFP can significantly reduce the end-to-end latency of SFCs at the medium resource expenditure compared with the state-of-the-art mechanisms.

The remainder of this paper is organized as follows. Firstly, an overview of the related work is provided in Section II. Then, Section III formulates the system models and problems. Section IV introduces in detail the proposed PVFP for parallel VNF placement in networks. After that, Section V presents the specific implementations, results and performance comparisons. Finally, we conclude the paper in Section VI.

II. Related Work

During the past years, numerous efforts have been devoted to efficient SFC placement in networks [1], [2]. Two comprehensive surveys of it are given in [2], [20], involving traffic scheduling, individual VNF acceleration, VNF resource utilisation, VNF instantiation, etc. Considering the logic of VNF execution, these previous efforts can be categorized as sequential and parallel placement of SFCs.

Sequential placement: These schemes mainly focus on diverse VNF service provisioning in a horizontal scope, i.e., considering the sequential composition of VNFs and accelerating each component of SFCs horizontally. Note that the VNF placement is an NP-hard problem, many studies with different concerns (e.g., the number of servers [21], [22], [23], resource overhead [4], [5], and network traffic utilization [24]) have proposed heuristics or intelligent schemes to solve the sequential deployment. To minimize the total costs for serving a set of requests, two logarithmic factor approximation approaches have been suggested for the placement of VNFs with ordering constraints [25]. Considering the traffic periodicity of workload, a two-stage heuristic solution has been designed to minimize the number of physical machines and improve the network resource utilization [11]. In addition, a heuristic solution based on Hidden Markov Chain has been designed to jointly minimize the cost and delay for a set of given flows [26]. Indeed, all of these schemes cannot reasonably fulfill the service requirements as the SFC length grows, especially for delay-sensitive applications, due to sequential SFC orchestration.

Parallel placement: These approaches are devoted to parallel VNF deployment from a vertical scope [13], [14], where some specific VNFs, for example, the virtualized FW and IDS, are executed in a parallel manner to enhance the NFV performance. A typical example includes the network function parallel (NFP) framework [15], which parallelizes NFs to create new service graphs based on the type of operations between NFs. To reduce the SFC latency, a novel hybrid packet processing architecture, named ParaBox, dynamically distributes packets to VNFs in parallel and merges their outputs intelligently to ensure the preservation of correct sequential processing semantics [27]. However, these efforts only present specific deployment schemes instead of basic frameworks. Thus, there are a few studies [28], [29], [30] reported in the direction of parallel VNF deployment details. For example, a multiple duplicate parallel solution in [28] has been used to improve end-to-end service reliability in Data Centre Networks (DCNs). The work in [13] and [14] is closely related to PVFP to address how to identify the optimal partial parallelism that minimizes the latency. Unfortunately, these heuristics cannot jointly capture the features behind the VNF pairs and the time-variant network states to take better actions for parallel SFC orchestration, thereby achieving less NFV performance gains. With this in mind, PVFP focuses on FDRL-based parallel orchestration of VNFs to fulfill the QoS requirements of SFCs, by introducing parallelism rules, latency/reward-based federated aggregation, and the flexible replay buffer in a distributed and efficient manner.

III. System Model and Problem Formulation

In this section, we provide a foundational introduction referring to the system models, SFC parallelization, and problem formulation, to facilitate the understanding of our proposed PVFP. The key parameters and notations are listed in Table I.

TABLE I: The Important Parameters and Notations

Notations	Descriptions
$\mathcal{V}/\mathcal{E}$	Set of nodes/links in networks
$\mathcal{V}_i/\mathcal{E}_i$	Set of nodes/links in domain G_i
$C^{cpu}_{v\|\mathcal{V}_i}/C^{bw}_{e\|\mathcal{E}_i}$	CPU capacity of node v/bandwidth capacity of link e in domain G_i
$\mathcal{F}=\{f_1,...,f_{\|\mathcal{F}\|}\}$	Set of VNF types
$C^{cpu}_{f_i}/C^{bw}_{f_i}$	CPU/bandwidth resources required by VNF f_i
$\mathbb{C}$	Set of all SFC requests
$x^{l_c}_{e\|\mathcal{E}_i}/x^{f_{j\|c}}_{v\|\mathcal{V}_i}$	Binary variable indicating if SFC c traverses link e/VNF $f_{j\|c}$ is placed in node v of domain G_i
$y_{f_{j\|c}}$	Binary variable of a function required parallel operations
$r^{cpu}_{v\|\mathcal{V}_i}/r^{bw}_{e\|\mathcal{E}_i}$	Available ratios of CPU/bandwidth of node v/e in domain G_i
t^w_c	Waiting time of SFC c passing through all nodes
$t^a_{f_{j\|c}}/t^p_{f_{j\|c}}/t^r_{f_{j\|c}}$	Activation/parallel operation latency of $f_{j\|c}$/ processing delay via $f_{j\|c}$
$T^{ac}_c/T^{pe}_c/T^{co}_c$	Activation /parallel execution/communication latency

A. System Models

The whole physical network, which includes a large number of nodes and links to host SFCs, is modeled as an undirected graph G=($\mathcal{V}$, $\mathcal{E}$). Both $\mathcal{V}$ and $\mathcal{E}$ stand for the sets of nodes and links, respectively. The element $v \in \mathcal{V}$ denotes a physical node, while $e \in \mathcal{E}$ indicates a link connecting two nodes. The whole physical networks are divided into $\mathcal{K}$ federated domains, i.e., G=$G_1 \cup ...G_i... \cup G_{\mathcal{K}}$, depending on their service providers and their cooperations. There are no the same shared elements between G_i and G_{i+1} except for the federated models received from the cloud server. Domain G_i is managed by its own service orchestrator to mobilize the available resources, and can be further expressed as G_i= $(\mathcal{V}_i, \mathcal{E}_i)$. The bandwidth capacity of link e∈$\mathcal{E}_i$ is denoted as $C^{bw}_{e|\mathcal{E}_i}$, while the CPU capacity of node v∈$\mathcal{V}_i$ is defined as $C^{cpu}_{v|\mathcal{V}_i}$. The total bandwidth and CPU capacities of all links and nodes included in G_i are considered as its indicators to host VNFs. Each node $v \in \mathcal{V}$ can transmit incoming data to one or multiple outgoing ports toward its destination. We use f_i and $\mathcal{F} = \{f_1, f_2, ..., f_{|\mathcal{F}|}\}$ to denote the $i-th$ VNF type and a set of service-required $|\mathcal{F}|$ VNF types, respectively. Each VNF is just deployed in a physical node that can host the corresponding NFs [24]. A pair of VNFs f_i, f_j∈$\mathcal{F}(i \neq j)$ can work at the same time only if they are parallelized, following the parallelism rules discussed with more details in Section IV. Each VNF consumes a specific amount of software/hardware resources. The sequence of VNFs can be switched if they are parallelizable. The CPU and bandwidth resources, required by f_i, are denoted as $C^{cpu}_{f_i}$ and $C^{bw}_{f_i}$, respectively.

It is considered that all SFCs are represented as a set $\mathbb{C}$, and $|\mathbb{C}|$ is the number of SFCs. A given SFC c =$s - f_1 - f_2, \cdots, -d$ of length $|\mathbb{F}|$ starts from source node s ∈$\mathcal{V}$ and destines to destination node d ∈$\mathcal{V}$ with the initial VNF sequence $\{f_{j|c}, 1 \leq j \leq |\mathbb{F}|\}$, where $f_{j|c}$ indicate the $j-th$ VNF of c. Similarly, the set of virtual links of c is defined as l_c. Let $x^{f_{j|c}}_{v|\mathcal{V}_i}$ indicate whether or not VNF $f_{j|c}$ is deployed in

node v of G_i, expressed as

$$x_{v|\mathcal{V}_i}^{f_{j|c}} = \begin{cases} 1, & \text{if } f_{j|c} \text{ is deployed in } v \text{ of } \mathcal{V}_i, \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

Similarly, let $x_{e|\mathcal{E}_i}^{l_c}$ denote whether or not SFC c traverses link e of G_i, given by

$$x_{e|\mathcal{E}_i}^{l_c} = \begin{cases} 1, & \text{if } l_c \text{ traverses } e \text{ of } \mathcal{E}_i, \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

Similar to [13], it is considered that there is a center server, which has full knowledge of the resource utilization inside domains and flow states between domains. Suppose the total amount of each type of resources (i.e. CPU and bandwidth) is a constant in each domain. With the aforementioned variables, the utilization ratios of CPU and bandwidth of node v and e in domain G_i, denoted by $r_{v|\mathcal{V}_i}^{cpu}$ and $r_{e|\mathcal{E}_i}^{bw}$, respectively, can be defined as

$$\begin{cases} r_{v|\mathcal{V}_i}^{cpu} = \dfrac{\sum_{f_{j|c} \in \mathbb{C}} x_{v|\mathcal{V}_i}^{f_{j|c}} C_{f_{j|c}}^{cpu}}{C_{v|\mathcal{V}_i}^{cpu}}, \forall v \in \mathcal{V}_i, \\ r_{e|\mathcal{E}_i}^{bw} = \dfrac{\sum_{l_c \in \mathbb{C}} x_{e|\mathcal{E}_i}^{l_c} C_{f_{j|c}}^{bw}}{C_{e|\mathcal{E}_i}^{bw}}, \forall e \in \mathcal{E}_i. \end{cases} \tag{3}$$

The available network resources and configurations of nodes and links defined in Eq. (3), in terms of CPU and bandwidth, will dynamically change with time.

B. SFC Parallelization

The main work of VNF orchestrators in networks is to process and place the SFC requests. Usually, an SFC request from users is a conventional chain, which includes the necessary VNFs ordered in sequence. To reduce the latency of SFCs, the VNF orchestrators can make SFCs partial parallelization, where a number of VNFs can be executed in parallel while others in sequence.

In order to facilitate the understanding, Fig. 1 illustrates a motivating example of an SFC $s-f_1-f_2-f_3-f_4-f_5-f_6-d$, which includes sequential VNF execution and parallel VNF execution. Each VNF included in $s-f_1-f_2-f_3-f_4-f_5-f_6-d$ shown in Fig. 1(a) has been deployed horizontally, following the sequential composition of VNFs. However, some VNFs, for example, f_3 and f_4, standing for Caching and NAT, respectively, have no dependency and can work in parallel. This is because the Caching only maintains packet statistics without modifying packets. Therefore, as shown in Fig. 1(b), the possible parallelism of sequential SFC $s-f_1-f_2-f_3-f_4-f_5-f_6-d$ is designed as $s-f_1-f_2-\{f_3,f_4\}--f_5-f_6-d$, which will achieve significantly lower delay than sequential SFC. It is appropriate to send traffic into the f_3 and f_4 simultaneously, which can still achieve the same result as sequential composition. By parallelizing VNFs f_3 and f_4, the computational time and communication latency of $s-f_1-f_2-f_3-f_4-f_5-f_6-d$ can be reduced with a feasible and flexible process [14].

In reality, each node in networks has limited capacity and can only support a limited number of VNF instances simultaneously. Once independent VNF pairs in a chain have been deployed in parallel, packets should be dispatched to different nodes and combined somewhere. In this case, these nodes can serve as the parallel function nodes to host VNFs with the cost of packet duplication/merging, but the end-to-end latency of SFC can be reduced, as intuitively illustrated in Fig. 1.

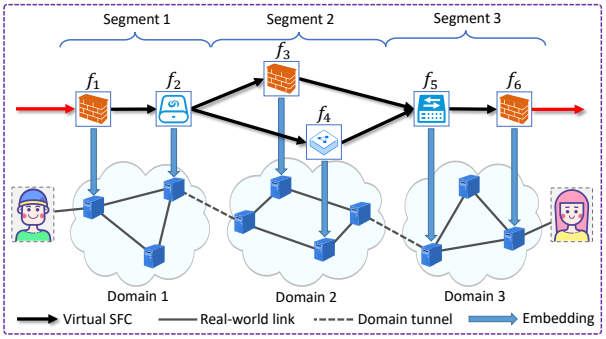


Fig. 2: Multi-domain network model.

C. Problem Formulation

Essentially, the end-to-end latency of a parallel SFC in multi-domain networks, as shown in Fig. 2, mainly includes activation latency, parallel execution latency, and communication latency, described below.

The *activation latency* T_c^{ac} is the total delay to activate all VNF instances through booting the VM images in networks and can be defined as follows:

$$T_c^{ac} = \sum_{v \in \mathcal{V}} \sum_{f_{j|c} \in c} x_{v|\mathcal{V}}^{f_{j|c}} t_{f_{j|c}}^{a}, \tag{4}$$

where $t_{f_{j|c}}^{a}$ stands for the time of $f_{j|c}$ to be activated in a node, and $x_{v|\mathcal{V}}^{f_{j|c}}$ is a binary variable defined in Eq. (1).

The *parallel execution latency* T_c^{pe} is the total delay to execute sequential SFCs into parallel SFCs with VNF parallelism principles, and can be defined as

$$T_c^{pe} = \sum_{f_{j|c} \in c} (\sum_{\eta_k} t_{f_{j|c}}^{\eta_k} + y_{f_{j|c}}^{p} t_{f_{j|c}}^{p}), 1 \le j \le |\mathbb{F}|, 1 \le k \le |\eta|, \tag{5}$$

where η_k and $|\eta|$ are the $k-th$ parallelism rule and the number of rules, respectively. $t_{f_{j|c}}^{\eta_k}$ is denoted as the execution time of VNF $f_{j|c}$ with principle η_k. The parameter $t_{f_{j|c}}^{p}$ stands for the parallelization latency caused by packet duplication/merging processing, in which the packets need to be dispatched to different nodes and combined somewhere, respectively, when their corresponding independent VNF pairs are deployed in networks in parallel. Let $t_{f_{j|c}}^{dup}$ and $t_{f_{j|c}}^{mer}$ denote the packet duplication/merging latency, respectively. Thus, $t_{f_{j|c}}^{p}$ can be further represented as $t_{f_{j|c}}^{p} = t_{f_{j|c}}^{dup} + t_{f_{j|c}}^{mer}$. The binary variable $y_{f_{j|c}}^{p}$ is used to represent whether or not a function requires parallel data processing. If so, $y_{f_{j|c}}^{p}$ equals 1, and 0 otherwise. The time $t_{f_{j|c}}^{dup}/t_{f_{j|c}}^{mer}$ required by node v to duplicate/merge the packet for its n parallel instances can be calculated by $t_{f_{j|c}}^{dup} = (n-1)t^{dup}$ and $t_{f_{j|c}}^{mer} = (n-1)t^{mer}$, where t^{dup} and

t^{mer} indicate the time of packet duplication/merging for each parallel instance, respectively.

The *communication latency* T_c^{co} is referred to as the time expenditure on traffic communications among all servers hosting VNFs, including data processing delay, transmission delay, propagation delay, and waiting delay, and can be given by

$$T_c^{co} = \max_{m_i} \big[\sum_{f_{j|c} \in m_i} x_{v|\mathcal{V}}^{f_{j|c}} t_{f_{j|c}}^{r} + \sum_{l_c \in m_i} x_{e|\mathcal{E}}^{l_c} t_{l_c}^{uv} + t_c^{w} \big], \tag{6}$$

where $t_{f_{j|c}}^{r}$ represents the data processing delay via VNF $f_{j|c}$, exclusive of data packet duplication/merging due to VNF parallelization, while $t_{l_c}^{uv}$ stands for the traffic propagation delay and transmission time via two adjacent physical nodes u and v, respectively. t_c^w represents the waiting time of all nodes in each end-to-end path, which SFC c passes through, to transmit data to the next hop. The parameter $m_i (i{=}1, 2, ...)$ is referred to as the $i-th$ end-to-end path of the parallelized SFC c. To facilitate the understanding, two such paths, i.e., $m_1{:}s-f_1-f_2-f_3-f_5-f_6-d$ and $m_2{:}s-f_1-f_2-f_4-f_5-f_6-d$, are shown in Fig. 1(b). Obviously, the actual communication delay is bounded by the maximum delay among all end-to-end paths.

With those decision variables, the end-to-end latency of partially parallel placement of SFC c in networks, denoted by T_c, can be expressed as

$$\begin{aligned} T_c &= T_c^{pe} + T_c^{co} + T_c^{ac} \\ &= \sum_{f_{j|c} \in c} \big(\sum_{\eta_k} t_{f_{j|c}}^{\eta_k} + y_{f_{j|c}}^{p} t_{f_{j|c}}^{p} \big) \\ &+ \max_{m_i} \big[\sum_{f_{j|c} \in m_i} x_{v|\mathcal{V}}^{f_{j|c}} t_{f_{j|c}}^{r} + \sum_{l_c \in m_i} x_{e|\mathcal{E}}^{l_c} t_{l_c}^{uv} + t_c^{w} \big] \\ &+ \sum_{v \in \mathcal{V}} \sum_{f_{j|c} \in c} x_{v|\mathcal{V}}^{f_{j|c}} t_{f_{j|c}}^{a}. \end{aligned} \tag{7}$$

Therefore, the parallel VNF placement of SFC set $\mathbb{C}$ in networks under resource constraints can be formulated as follows:

$$\textbf{min:} \quad T = \frac{\sum_{c \in \mathbb{C}} T_c}{|\mathbb{C}|}, \tag{8}$$

subject to

$$\textbf{s.t.} \begin{cases} \sum\limits_{v \in \mathcal{V}_i} x_{v|\mathcal{V}_i}^{f_{j|c}} = 1, \forall f_{j|c} \in \mathbb{C}, i \in [1, \mathcal{K}], \\ \sum\limits_{f_{j|c} \in \mathbb{C}} x_{v|\mathcal{V}_i}^{f_{j|c}} C_{f_{j|c}}^{cpu} \le C_{v|\mathcal{V}_i}^{cpu}, \forall v \in \mathcal{V}_i, i \in [1, \mathcal{K}], \\ \sum\limits_{l_c \in \mathbb{C}} x_{e|\mathcal{E}_i}^{l_c} C_{f_{j|c}}^{bw} \le C_{e|\mathcal{E}_i}^{bw}, \forall e \in \mathcal{E}_i, i \in [1, \mathcal{K}], \\ \sum\limits_{f_{j|c} \in \mathbb{C}} x_{v|\mathcal{V}_i}^{f_{j|c}} x_{v'|\mathcal{V}_i}^{f_{j+1|c}} = \sum\limits_{f_{j|c} \in c} x_{v|\mathcal{V}_i}^{f_{j+1|c}} x_{v'|\mathcal{V}_i}^{f_{j|c}}, \\ \qquad\qquad \forall v, v' \in \mathcal{V}_i, i \in [1, \mathcal{K}]. \end{cases} \tag{9}$$

This will be achieved by applying FDRL to learn a global SFC deployment model such that $\mathcal{K}$ domain orchestrators can execute parallel VNF instances in an efficient manner with resource constraints. The first constraint in Eq. (9) ensures that each VNF can be deployed in one and only one node. The following two constraints illustrate that the required resources $C_{f_{j|c}}^{cpu}$ and $C_{f_{j|c}}^{bw}$ of VNF $f_{i|c}$ and virtual link l_c can not exceed the maximum resource of node v and the bandwidth capacity of link e, respectively. The final constraint indicates that any intermediate node, which an SFC c traverses through, needs to follow the flow-conservation principle except for its source destination nodes.

IV. PVFP: Parallel VNF Placement

This section presents in detail the proposed PVFP for parallel placement of SFCs in NFV-based networks. First, we present the framework of parallel VNF placement and then introduce the VNF parallelism principles. Thirdly, we develop an SFC decomposition scheme in multiple domains and design a 3-tuple required by FDRL. Finally, we describe the local training process based on FDRL in detail.

A. Framework of Parallel VNF Placement

The main idea of PVFP is to design an intelligent framework of parallel VNF placement with the goal of minimizing end-to-end SFC latency. By utilizing FDRL to train a global SFC deployment model, a set of domain orchestrators can implement optimal parallel VNF instances with dynamically available resources.

The framework of PVFP is illustrated in Fig. 3, which consists of *SFC parallelism and decomposition*, *local training*, and *aggregation training*. There are two kinds of servers, i.e., the cloud server and domain orchestrators, to be assigned to jointly perform such tasks. The cloud server is responsible for SFC decomposition and aggregation training, while the domain orchestrators are in charge of local training. The timescale is made of a series of aggregation episodes $\mathbb{T}{=}\{\mathbb{T}_1, \mathbb{T}_2, ...\}$, each of which consists of a set of the same local epochs $\tau_1, ..., \tau_j, ...\tau_{\mathbb{N}}$. The whole network has been divided into $\mathcal{K}$ domains, each of which has a domain orchestrator installed in deep Q-networks (DQN) to iteratively learn its local model $\theta_i(t) (i \in [1, \mathcal{K}])$ via DRL and serves as a function node to communicate with the cloud server. The model $\theta_i(t)$ is structured based on neural networks of DQN, guiding resource allocation, delay perception, and parallel VNF deployment within each time interval, and will be utilized by its domain orchestrator for decision-making regarding parallel VNF deployments. The *SFC parallelism and decomposition* aims to design a parallelism principle and factorize the whole SFC into $\mathcal{K}$ segments $\mathbb{S}_1, ..., \mathbb{S}_{\mathcal{K}}$ with the available resources of different domains. The initial network models and SFC segment $\mathbb{S}_i$ will be delivered to domain G_i from the cloud server. The *local training* executed by each domain orchestrator is to learn a locally deployed model $\theta_i(t)$ via DRL. To preserve the privacy of data, each local $\theta_i(t)$ is not shared with other orchestrators but with the cloud server. The *aggregation training* is to collect all local models $\theta_1(t), ..., \theta_{\mathcal{K}}(t)$ executed by the cloud server to learn a global model $\Theta(t+1)$ and then send it to domain orchestrators for further VNF deployment with the learned intelligent parallel policy. Essentially, $\Theta(t)$ is a DQN-based model with a number of built-in parameters learned from all domain orchestrators, through the collaborative efforts among

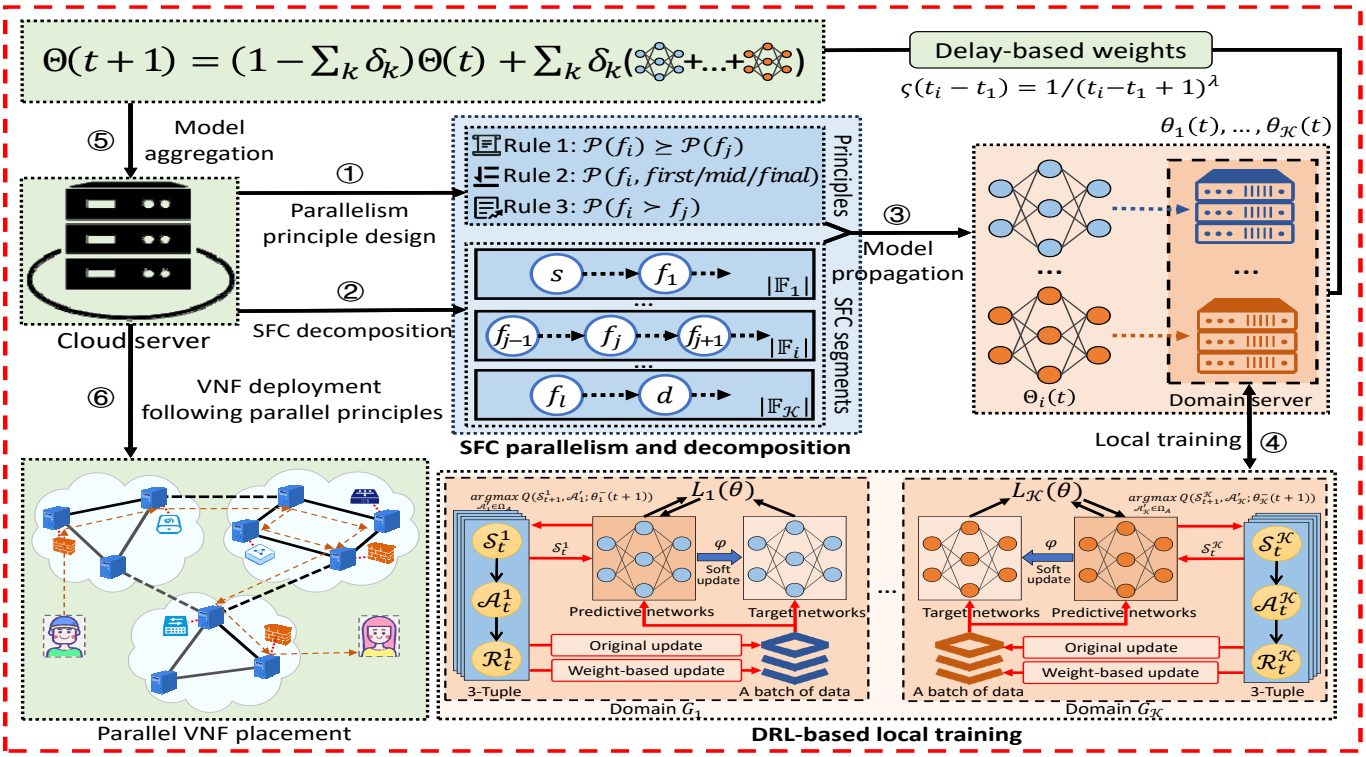


Fig. 3: The whole architecture of PVFP, which mainly SFC parallelism and decomposition, local training, and aggregation training to realize parallel VNF placement.

them without centralized data aggregation, to train a global optimal SFC placement solution. It is worth noting that the parallel structure of the SFC segment in each domain will be redesigned by its local orchestrator to form a more optimized action.

The operation of PVFP is generally divided into six steps: ① parallelism principle design, ② SFC decomposition, ③ model propagation, ④ local parallel training, ⑤ model aggregation, and ⑥ VNF deployment. In step ①, a parallelism principle, including three types of rules, has been designed to identify dependencies of VNFs and ensure the processing correctness. In step ②, the given SFCs are divided into $\mathcal{K}$ segments, built on the remaining resources of each domain and potential parallelization of VNFs. Potential parallel VNFs should be assigned to the same domain to reduce inter-domain information exchange. In step ③, an initial centralized network model and the initial SFC segments, which are specified by the cloud server, are delivered to different domains. Each model consists of a Q-network with local client parameters, guiding the domain orchestrator to learn a model $\theta_i(t)$ to allocate parallelized VNFs as the network state changes. Then, each domain orchestrator uses the local data to iteratively train its own model in each epoch of step ④, and then sends updated models to the cloud server at each time slot of FL training. In step ⑤, the cloud server aggregates updated local models received from all domains into a new model $\Theta(t+1)$. Note that the parallelisation structure of the SFC segment will be redesigned by the local orchestrator to form a better action. In step ⑥, the partial parallel VNFs are deployed in separate domains according to the current intelligent policy. The above steps will be repeated to promote the evolution of the shared models.

In our FDRL-based framework, each domain orchestrator will send its local learning model at different time slots to the cloud server. This means that our cloud server will spend different amounts of time to collect all local models $\theta_i(t)$,....,$\theta_\mathcal{K}(t)$. Note that the time to receive each local model is related to the training efficiency of each domain, we introduce a delay-based aggregation weight to update $\Theta(t+1)$ to accelerate the global convergence. The mixed weight is set adaptively as a function of the time staleness $\varsigma(t_i - t_1)$ [7], defined as

$$\varsigma(t_i - t_1) = \frac{1}{(t_i - t_1 + 1)^\lambda}, \tag{10}$$

where $(t_i - t_1)$ is expressed as the time difference from the cloud server receiving the first model to receiving the model of G_i, and the parameter λ is set between 1 and 5. Let the delay-based aggregation weight $\delta_i = \varsigma(t_i - t_1) * \delta, \delta \in (0,1)$ represent the weight of G_i when participating in the global model training. training. It adaptively controls the trade-off according to the staleness $\varsigma(t_i - t_1)$. Then, the updated global federated model $\Theta(t+1)$ can be expressed as

$$\Theta(t+1) = (1 - \sum_i \delta_i)\Theta(t) + \sum_i \delta_i \theta_i(t), \tag{11}$$

where larger staleness $\varsigma(t_i - t_1)$ results in lower weight δ_i when updating $\Theta(t+1)$.

B. VNF Parallelism Principle Design

To enhance VNF performance in parallel, particularly in terms of end-to-end latency, an intuitive way is required to describe both sequential and parallel VNF composition intents to host VNFs. Thus, a VNF parallelism principle, which includes three types of rules, has been designed as follows.

Rule 1:$\mathcal{P}(f_i) \succeq \mathcal{P}(f_j)$. This rule expresses the parallel processing order principle of several VNFs, where $\mathcal{P}(f_i)$ indicates the execution order of VNF f_i in sequential SFCs, while " $\succ$ " and " $=$ " indicate that f_i can be parallelly executed before f_j and they can work in parallel, respectively. For example, as shown in Fig. 1(a), the network operator can first send the traffic to the IDS and then the Caching and NAT in the specified order, while there is no strict execution sequence between Caching and NAT, meaning that $\mathcal{P}(IDS) \succ \mathcal{P}(Caching)$, $\mathcal{P}(IDS) \succ \mathcal{P}(NAT)$, and $\mathcal{P}(Caching) = \mathcal{P}(NAT)$. This order rule can be used to describe a continuous sequence. To guarantee the parallel processing equivalency, an optimal parallel SFC will be designed to process the Caching and NAT at the same time for better performance in Fig. 1(b).

The most important principle of VNF parallelism is ensuring the equivalence in parallel processing between a SFC SFC and the corresponding original sequence. *Rule 1* can simply provide the desired execution order of several VNFs. The ordering relationship between VNFs can be identified by inspecting related documentation and utilizing VNF actions on packets derived from other studies [15]. The outcome of addressing any request with parallel SFCs should be identical to processing the same request with the original sequential SFC.

Rule 2:$\mathcal{P}(f_i, first/mid/final)$. This rule indicates the VNF position dependency principle, where particular VNFs should be placed in specific locations in networks. However, it is difficult to foreknow the final optimized SFC structure. Thus, a VNF can only be assigned as the first, middle or final one in *Rule 2*. For instance, the SFC request in Fig. 1(a) requires all flows to be processed by a firewall first. A specific position is specified to ensure packets traverse a firewall first (e.g., $\mathcal{P}(FW, first)$). In addition, the position of each VNF can be determined to make the critical path of the optimized SFC shortest.

Rule 3:$\mathcal{P}(f_i \succ f_j)$. This rule introduces a priority principle of several VNFs, where the data passing through f_i will be retained if f_i and f_j are in parallel. The first and second principles can describe the intent to execute several VNFs in parallel. However, the taken actions of several VNFs may conflict. For instance, the Caching and NAT may disagree on whether to drop a packet or not. Therefore, it is requisite to specify a priority rule to process two parallel VNFs, which further inspects their dependency to see whether they are parallelizable. Then, the coordinator in *Rule 3* will automatically identify conflicting actions between VNFs and parallelise them accordingly.

C. SFC Decomposition in Multi-domain

Note that there are different available resources in each domain and potential VNF parallelism; it is necessary to decompose the whole SFC into different domains. The cloud server, as a multi-domain orchestrator, will be responsible for such a task, and also for the coordination of available resources in all domains. To this end, it will store the available resource information adaptively reported from $\mathcal{K}$ domain orchestrators, and decide the whole strategy among domains to keep the balance between latency and resource constraints, as shown in Fig. 2.

Algorithm 1: SFC Decomposition

Input: SFC length $|\mathbb{F}|$
Output: SFC length of each domain $|\mathbb{F}_i|$
Data: Initial CPU capacities of nodes in each domain $C^{cpu}_{v|\mathcal{V}_i}$, CPU ratios of each domain $r^{cpu}_{v|\mathcal{V}_i}$

```
Cloud server does
for i ∈ [1, K − 1] do
    |F_i| = ⌊ (Σ_{v∈V_i} (1 − r^cpu_{v|V_i}) C^cpu_{v|V_i}) / (Σ_{j=i}^{K} Σ_{v∈V_j} (1 − r^cpu_{v|V_j}) C^cpu_{v|V_j}) × |F| ⌋;
    |F| = |F| − |F_i|;
    i++;
|F_i| = |F|;
return |F_i|, i ∈ [1, K].
```

A specific SFC $s-f_1,...,f_j,...-d$ will be divided into $\mathcal{K}$ SFC segments $\mathbb{S}_1,...,\mathbb{S}_\mathcal{K}$, built on the available resources of all domains G_1,...,$G_\mathcal{K}$. Here, $\mathbb{S}_i= f_l-,...,-f_m$ and $\mathbb{S}_{i+1}= f_{m+1}-,...,-f_n$, which satisfy $\mathbb{S}_i \bigcap \mathbb{S}_{i+1}=\phi$. Denote the length of f_j be 1 and equal to $f_k(j \neq k)$. Thus, the number of VNFs assigned to G_i can be given by

$$|\mathbb{F}_i| = \lfloor \frac{\sum_{v\in\mathcal{V}_i}(1 - r^{cpu}_{v|\mathcal{V}_i})C^{cpu}_{v|\mathcal{V}_i}}{\sum_{j=i}^{\mathcal{K}}\sum_{v\in\mathcal{V}_j}(1 - r^{cpu}_{v|\mathcal{V}_j})C^{cpu}_{v|\mathcal{V}_j}} \times |\mathbb{F}| \rfloor, \tag{12}$$

where $|\mathbb{F}_i|(1 \leq i \leq \mathcal{K})$ represents the length of SFC segment $\mathbb{S}_i$ assigned to domain G_i, and the symbol $\lfloor . \rfloor$ stands for a floor function (i.e., $\lfloor 14.7 \rfloor = 14$). All $f_1,...,f_{|\mathbb{F}_i|} \in \mathbb{S}_j(1 \leq j \leq \mathcal{K})$ will be assigned to domain G_j in sequence.

Following this rule, the SFC $s - f_1 - ... - f_{j-1} - f_j - f_{j+1} - ... - f_{|\mathbb{F}|} - d$ illustrated in Fig. 3 has been decomposed into a series of segments $\mathbb{S}_1, ..., \mathbb{S}_i, ..., \mathbb{S}_\mathcal{K}$ with the lengths of $|\mathbb{F}_1|, ..., |\mathbb{F}_i|, ..., |\mathbb{F}_\mathcal{K}|$, respectively. Note that PVFP utilises CPU rather than bandwidth as the indicator of remaining resources for SFC decomposition in Eq. (12), because the CPU utilization of nodes is closely related to the processing latency of service requests, which significantly affects the local training feedback. The pseudocode of this decomposition process is illustrated in Algorithm 1.

D. State, Action and Reward Spaces

The agent in each domain orchestrator works closely with a 3-tuple $< \Omega_S, \Omega_A, \Omega_R >$ to instantiate VNFs parallelly, where Ω_S, Ω_A and Ω_R represent the state space, action space and reward space, respectively. The details of them can be described as follows.

The state $\mathcal{S}_t^i \in \Omega_S$ at time t is referred to as the data of all network resources and configurations in domain G_i and can be denoted as

$$\begin{aligned}\mathcal{S}_t^i = < C_{v|\mathcal{V}_i}^{cpu}, C_{e|\mathcal{E}_i}^{bw}, r_{v|\mathcal{V}_i}^{cpu}, r_{e|\mathcal{E}_i}^{bw}, C_{f_j}^{cpu}, C_{f_j}^{bw}, \\ \forall v \in \mathcal{V}_i, \forall e \in \mathcal{E}_i, \forall f_j \in \mathbb{S}_i > .\end{aligned} \tag{13}$$

It clearly indicates that in domain G_i, the available network resources of node $v \in \mathcal{V}_i$ and link $e \in \mathcal{E}_i$, in terms of CPU and bandwidth, will be continually changing with time.

The action $\mathcal{A}_t^i \in \Omega_A$ at time t serves two different functions, i.e., one for choosing possible parallel actions, and the other for appointing current deployment schemes. The action space Ω_A is divided into Ω_A^p and Ω_A^d, defined as

$$\Omega_A = \Omega_A^p \cup \Omega_A^d, \Omega_A^p \cap \Omega_A^d = \emptyset, \tag{14}$$

where Ω_A^p refers to the action subspace of parallel VNFs and Ω_A^d represents the action subspace of VNF deployment vectors, e.g., the segment of SFC IDS-{Caching, NAT}-LB in Fig. 1(b). As a consequence, each unique task is handled by separate actions in these two spaces, including selecting the parallel chain $\mathcal{A}_t^{i,p} \in \Omega_A^p$ and deploying the selected chains $\mathcal{A}_t^{i,d} \in \Omega_A^d$. The details of subspaces Ω_A^p and Ω_A^d are described as follows.

The parallelization action subspace Ω_A^p includes a discrete action set and can be further denoted as

$$\Omega_A^p = \{(\alpha_{mn})_{|\mathbb{F}_i|\times|\mathbb{F}_i|} | 1 \le m \le |\mathbb{F}_i|, 1 \le n \le |\mathbb{F}_i|\}, \tag{15}$$

where each action $\mathcal{A}_t^{i,p} = (\alpha_{mn})$ is designed as a square matrix of $|\mathbb{F}_i|\times|\mathbb{F}_i|$, i.e., the allocated SFC length in domain G_i. The element α_{mn} is given by

$$\alpha_{mn} = \begin{cases} 1, & \text{if } f_m \text{ connects to } f_n, \forall f_m, f_n \in \mathbb{F}_m, \\ 0, & \text{otherwise.} \end{cases} \tag{16}$$

In other words, the action $\mathcal{A}_t^{i,p} = (\alpha_{mn})$ records possible parallel actions in space Ω_A^p. However, not all actions in space Ω_A^p are valid due to VNF parallelism principles. The selected actions must satisfy *Rules 1∼3*, which depends on the dependencies of VNFs. Note that the length of the parallel SFC segments is no longer than a constant $\mathcal{L}$, as the resource consumption can not exceed the limits of the local orchestrator.

The deploying action subspace Ω_A^d is also described as a set of matrices for uniformity and can be expressed as

$$\Omega_A^d = \{(\beta_{mv})_{|\mathbb{F}_i|\times|\mathcal{V}_i|} | 1 \le m \le |\mathbb{F}_i|, 1 \le v \le |\mathcal{V}_i|\}, \tag{17}$$

where the action $\mathcal{A}_t^{i,d} = (\beta_{mv})$ stands for deploying VNFs $f_m \in \mathbb{F}_i$ of the selected SFC to node $v \in \mathcal{V}_i$ in G_i. If f_m is deployed in node v, $\beta_{mv} = 1$, otherwise $\beta_{mv} = 0$.

The reward $\mathcal{R}_t^i \in \Omega_R$, decided by state $\mathcal{S}_t^i$ and action $\mathcal{A}_t^i$, is consistent with our optimization objective, defined in Eq. (8), i.e., to minimize the end-to-end latency of SFC segments in G_i. Following the maximized principle of DQN, the reward $\mathcal{R}_t^i$ can be modeled as a function of state $\mathcal{S}_t^i$ and action $\mathcal{A}_t^i$, defined as

$$\mathcal{R}_t^i = -[T_{c,i}^{ac} + T_{c,i}^{pe} + T_{c,i}^{co}], \forall i \in [1, \mathcal{K}], \tag{18}$$

where $T_{c,i}^{ac}$, $T_{c,i}^{pe}$ and $T_{c,i}^{co}$ are defined as the activation latency, parallel execution latency and communication latency in G_i, respectively.

E. DRL-based Local Training for Parallel VNF Placement

In order to facilitate the understanding, let $Q_i(\mathcal{S}_t^i, \mathcal{A}_t^i)$ be the action-value function in domain G_i, which is a pair of state $\mathcal{S}_t^i$ and action taken from Ω_A^p and Ω_A^d. The local training of PVFP in domain G_i is built on $Q_i(\mathcal{S}_t^i, \mathcal{A}_t^i)$, defined as

$$Q_i(\mathcal{S}_t^i = s, \mathcal{A}_t^i = a) = \mathbb{E}[\sum_{k=t}^{T} \gamma^{k-t} \mathcal{R}_t^i | S_t = s, A_t = a], \tag{19}$$

to execute a parallel VNF placement with DQN. Here, $\gamma(0<\gamma<1)$ stands for the discount factor to indicate the importance of all subsequent rewards in $Q_i(\mathcal{S}_t^i, \mathcal{A}_t^i)$.

There are two neural networks, i.e., predictive networks Q and target networks $\hat{Q}$, in DQN. Both networks are designed to train $Q_i(\mathcal{S}_t^i, \mathcal{A}_t^i)$ with the built-in parameter $\hat{\theta}$ step by step. To precisely learn the optimal action-value function, the loss function $L_i(\hat{\theta})$ has been introduced and defined as

$$\begin{aligned}L_i(\hat{\theta}) = \mathbb{E}[\mathcal{R}_t^i + \gamma \max_{\mathcal{A}'} \hat{Q}(\mathcal{S}_{t+1}^i, \mathcal{A}_i'; \hat{\theta}_i^-(t+1)) \\ - Q(\mathcal{S}_t^i, \mathcal{A}_t^i; \hat{\theta}_i(t))]^2,\end{aligned} \tag{20}$$

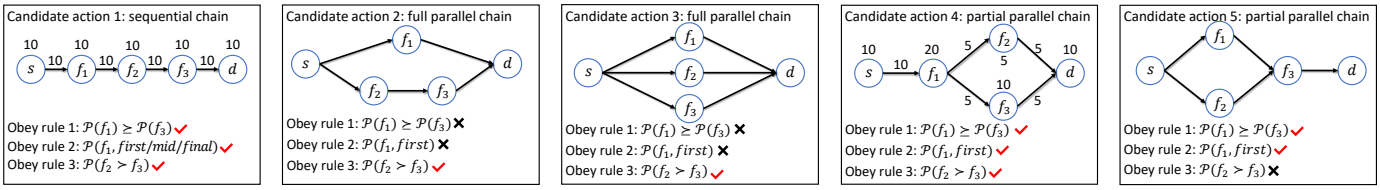
where $L_i(\hat{\theta})$ estimates the deviation of the predicted value in domain G_i, while $\hat{\theta}_i(t)$ and $\hat{\theta}_i^-(t+1)$ represent the built-in parameters of predictive networks and target networks at times t and $t+1$, respectively. The target network $\hat{Q}_i$ is introduced to generate the target reward for taking action $\mathcal{A}_i'$ at state $\mathcal{S}_{t+1}^i$.

Following the designed VNF parallelism principles, each domain orchestrator will select actions satisfying Eq. (20) from several candidate actions to identify the optimal action-value function. Fig. 4 illustrates an example of action-value function learning in G_i, which refers to 3 VNFs. The traditional actions for sequential VNF instance are shown in Fig. 4(a), and their potential parallel actions for VNF deployment are demonstrated in Figs. 4(b)-(e). With $\mathcal{P}(f_i) \succeq \mathcal{P}(f_j)$, $\mathcal{P}(f_i, first)$ and $\mathcal{P}(f_i \succ f_j)$ in mind, only action 4 in Fig. 4(d) can meet the filtering requirements with the given operations and transmission delays. The total delays in Figs. 4(a) and 4(d) are $10 \times 9 = 90$ and $10 + 10 + 20 + 5 + 10 + 5 + 10 = 70$, respectively, derived from the real-world applications and included in the action-value function in the form of rewards. Obviously, compared to sequential actions, the parallel actions can reduce the end-to-end latency of VNF instances. Built on Eq. (19), the optimal action-value function can be evaluated with these two candidate actions and the corresponding action-value function, in the temporal-difference iterative manner. Finally, the orchestrator will give the optimal action-value function to the corresponding candidate action 4 for this training epoch.

In order to improve the stability of local learning and accelerate the convergence process, a soft target update, instead of standardly copying neural networks, is introduced to change the model parameter $\hat{\theta}_i^-(t)$ of the target network $\hat{Q}$ by each $\mathbb{N}$ epochs [31]. Specifically, the model update of target networks, which slowly tracks the predictive networks, follows the rule as

$$\hat{\theta}_i^-(t+1) = \varphi \hat{\theta}_i(t) + (1-\varphi)\hat{\theta}_i^-(t), \tag{21}$$

where $\varphi(0 \le \varphi \ll 1)$ is a neural network discount factor.



(a) (b) (c) (d) (e)

Fig. 4: The potential parallel VNF placement with candidate actions at given states, following the designed VNF parallelism principles.

Note that there are different amounts of resources available in each domain, therefore, a resource-oriented sample buffer for data exploration has been introduced, as shown in Fig. 3. The capacity of the experience buffer can be variable with the different sizes, depending on the whole available resources in each domain, defined in Eq. (12). The samples in the smaller buffer will be replaced faster by the incoming samples due to data overflow, and used for local learning to obtain $Q_i(\mathcal{S}_t^i, \mathcal{A}_t^i)$.

In addition, a reward-based adaptive ϵ-greedy strategy is adopted for ensuring that all actions $< \mathcal{A}_t^{i,p}, \mathcal{A}_t^{i,d} > \in \Omega_A$ in different domains can be selected faster to complete possible action exploration [32]. Considering that different SFC deployment leads to different reward values, the exploration factor ϵ_i in G_i is updated as follows:

$$\epsilon_i = \frac{\mathcal{R}_t^i}{\mathcal{R}_t^i - 1}, \tag{22}$$

where $\mathcal{R}_t^i \in \Omega_R$ is the current reward of domain G_i, built on Eq. (18). The domain orchestrator makes a random exploration from Ω_A with probability ϵ_i or chooses the optimal policy with probability $1 - \epsilon_i$.

To facilitate understanding, the pseudocode of FDRL-based training framework has been listed in **Algorithm 2**. Line 3 of **Algorithm 2** introduces ① parallelism principle, which includes three types of rules. Line 4 of **Algorithm 2** gives ② preliminary decomposition of each request. Lines 5-6 and Lines 11-14 of **Algorithm 2** refer to ③ model propagation and ⑤ model aggregation, respectively. Lines 7-10 of **Algorithm 2** describe the process of ④ local parallel training. VNF allocation in ⑥ will be performed after each epoch of FL training in Line 15 of **Algorithm 2**.

Algorithm 2: FDRL-based Training Framework

Input: Initial model $\Theta(t)$
Output: Improved model $\Theta(t+1)$
Data: Initial local model $\theta_i(t)$, initial weight λ
1 **for** *epoch* $t \in \mathbb{T}$ **do**
2 **Cloud Server does**
3 $\mathcal{P}(f_i) \succeq \mathcal{P}(f_j)$, $\mathcal{P}(f_i, \text{first/mid/final})$, $\mathcal{P}(f_i \succ f_j)$;
4 Decompose $|\mathbb{F}|$ to $|\mathbb{F}_i|$;
5 $\delta_i \leftarrow \delta$;
6 Send $\Theta(t)$ to G_i ;
7 **Local orchestrator does**
8 **foreach** G_i ***in parallel*** **do**
9 Accept and update $\theta_i(t)$;
10 Upload $\varsigma(t_i - t_1), \theta_i(t)$;
11 **Cloud server does**
12 Gather $\varsigma(t_i - t_1), \theta_i(t)$;
13 $\delta_i \leftarrow \varsigma(t_i - t_0) * \delta$;
14 $\Theta(t+1) \leftarrow (1 - \sum_i \delta_i)\Theta(t) + \sum_i \delta_i \theta_i(t)$;
15 **return** $\Theta(t+1)$.

V. Experimental Evaluation and Results

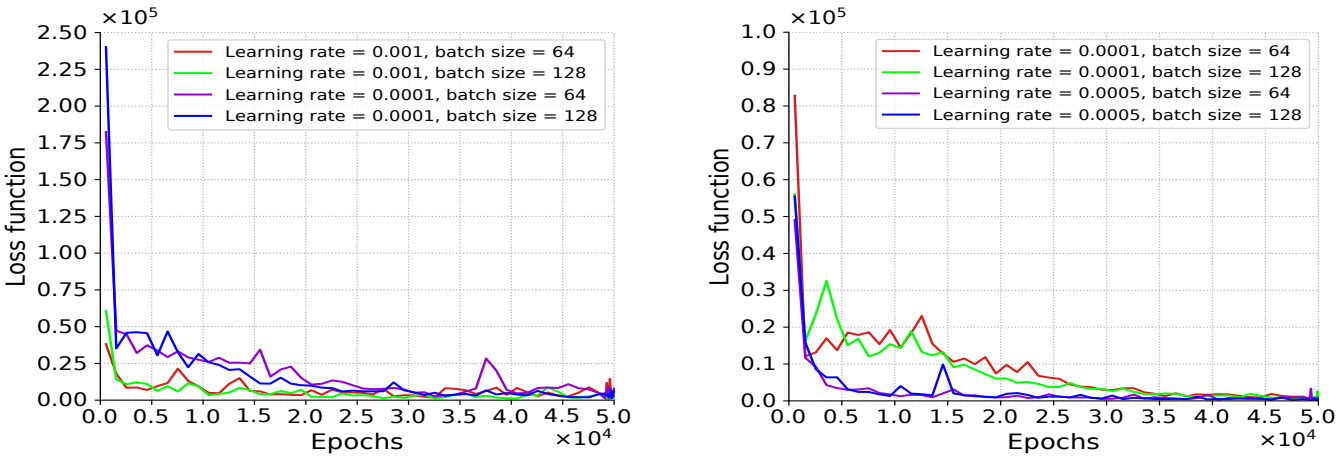
In this section, we conduct extensive simulations on three universal servers, equipped with an Intel(R) Core(TM) i5-8300 CPU 2.30@GHz and an Nvidia GeForce GTX 1080Ti GPU, to evaluate the performance of PVFP. First, we describe the experiment setup and then present our evaluation metrics. Finally, we illustrate the results of performance comparisons among PVFP and four typical related approaches [13], [16], [33], [34].

A. Simulation Setup

The simulation experiments are executed using TensorFlow-gpu 1.10 for neural network implementation. These networks comprise three fully connected layers, each with 600 neurons activated by the ReLU function. The target Q-network is replaced at the end of each episode. For FDRL, we set the time staleness parameter λ to 5 and the basal aggregation weight δ to 0.9. The main parameter settings associated with FDRL and neural networks are listed in Table II [1].

Two network topologies used for simulation experiments stem from SNDlib [35], including a small-scale network with 12 nodes and 15 links as well as a large-scale network with 35 nodes and 79 links, representing two simulation scenarios. Built on their available resources, these networks are divided into three domains, each containing an orchestrator as a function node for communication with the cloud server. Data exchange between orchestrators occurs only upon completion of a full SFC request. In small-scale and large-scale scenarios, the CPU capabilities of nodes are set to 30 and 20 Cores [8], and the link bandwidth capabilities are set to 4 and 2 Mbps, respectively.

Unless specified otherwise, our simulations employ 7 different VNF types and a maximum SFC length of 10. The VNFs include NAT, load balancer, NIDS, Gateway, VPN, FW, and Caching. SFCs are randomly generated between various source-destination host pairs, comprising VNFs selected from



(a) Convergence of PVFP in the small-scale network

(b) Convergence of PVFP in the large-scale network

Fig. 5: Convergence of PVFP with the varying learning rate and batch size in different networks.

TABLE II: Parameter Settings in PVFP

Hyperparameters	**Value**
Learning rate α	0.0001-0.001
Discount factor γ	0.99
Parameter ε /its decay rate	0.01/0.95
Number of domains $\mathcal{K}$	3
Capability of replay memory D_s	10000
Capability of batch size	64-128
Number of layers	3
Number of neurons each layer	600
Time staleness parameter λ	5
Basal aggregation weight δ	0.9

the aforementioned 7 types. The default packet size is 500 bytes, with a 60-byte header, and the read-write time per bit is 0.08 ms, two of which jointly determine packet processing times. Function execution times are set randomly between 5 and 10 ms. To align transmission time with function execution time, we set transmission delays at 20 ms. Each VNF has CPU capacity ranging from 1 to 3 Cores, and virtual link bandwidth capacity varying between 0.1-0.3 Mbps. The average performance results are collected from 30 simulation runs in each scenario, presented with confidence intervals.

To qualitatively execute evaluations, in addition to PVFP, we have also implemented four typical solutions: Greedy Sequential Scheme (GSS) [34], ParaSFC [13], Neural Combinatorial Optimization (NCO) [16], and Gecode [33] described below.

- GSS: GSS is a parallel heuristic scheme built on a Greedy strategy. To instantiate the parallel SFC, GSS calculates the shortest path from the ingress to the egress of each sequential SFC and selects nodes closer to the shortest path as its candidates.
- ParaSFC: ParaSFC is a parallel VNF deployment scheme based on Viterbi dynamic programming. It estimates each parallelized SFC occupation of the bottleneck resources and adjusts the processing order of parallelized SFCs. To approximate optimal solutions in resource-limited networks, ParaSFC regulates the parallel processing order through the most abusive p-SFC first drop policy and the least competitive node first allocate policy.
- NCO: NCO is a sequential DRL-based solution of VNF instance based on neural combinatorial optimization theory. By introducing Long-Short Term Memory (LSTM) into DRL, NCO has learned effective placement policies, aimed at minimizing the whole resource consumption.
- Gecode: Gecode, a professional programming solver with state-of-the-art performance in solving constraint-based optimization problems, is utilized to optimize SFC placement according to the objective and constraints defined in Eq. (8) and Eq. (9), representing a solution that approximates the optimal placement the most.

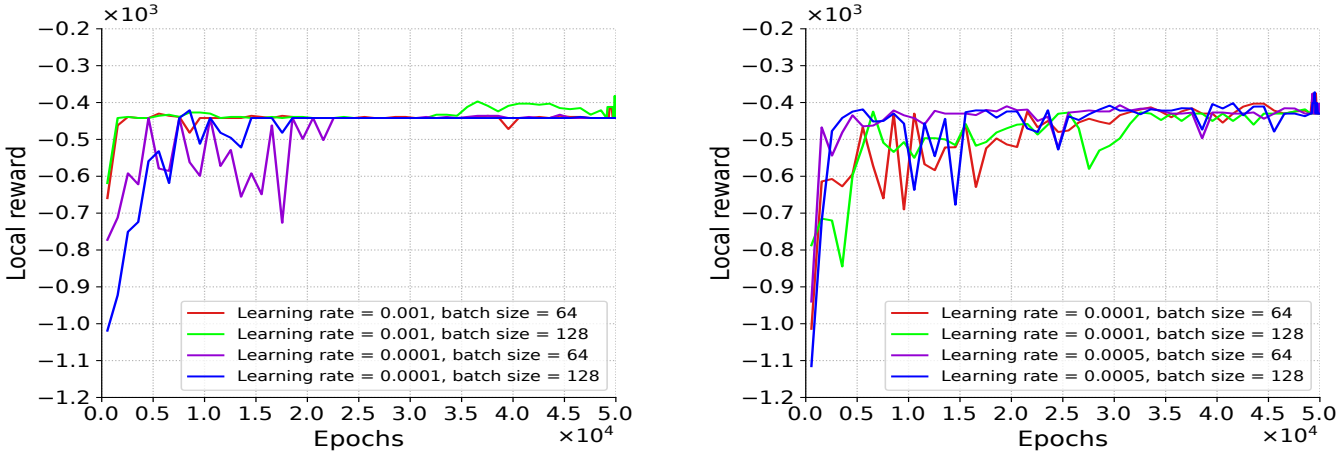
B. Evaluation Metrics

There are four evaluation metrics, i.e., loss function, local reward, average end-to-end latency, and resource overhead, to investigate the performance of PVFP.

- **Loss function:** The loss function $L_i(\hat{\theta})$ defined in Eq. (20) indicates the convergence performance of a DQN model in PVFP. The smaller the loss function is, the better estimation performance a DQN model will have.
- **Local reward:** The local reward $\mathcal{R}_t^i$ is referred to as the end-to-end latency of SFC segments $\mathbb{F}_i$ in G_i, defined in Eq. (18).
- **Average end-to-end latency:** The average end-to-end latency T refers to the average latency of deploying parallel SFCs in networks, which further includes activation latency T_c^{ac}, parallel execution latency T_c^{pe} and communication latency T_c^{co} as defined in Eqs. (4), (5), and (6), respectively.
- **Resource overhead:** This metric is referred to as the total consumed resources in each node $v \in \mathcal{V}_i$ and link $e \in \mathcal{E}_i$ to host all VNFs, with $r_{v|\mathcal{V}_i}^{cpu}$, $r_{e|\mathcal{E}_i}^{bw}$, defined in Eq. (3).

C. Simulation Results

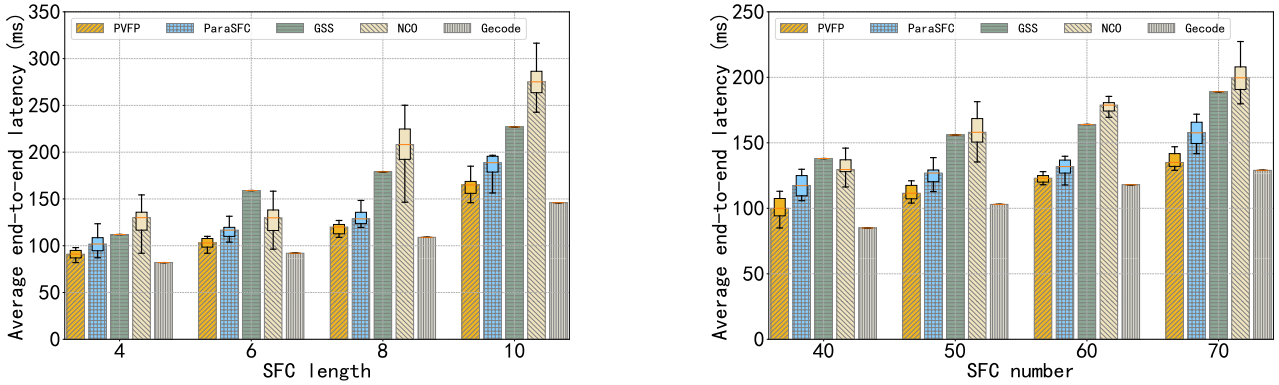
1) Convergence of PVFP in Local Training: Fig. 5 illustrates the convergence of PVFP, in the form of loss function in different networks. The learning rate α is set to be either



(a) Local rewards of PVFP in the small-scale network

(b) Local rewards of PVFP in the large-scale network

Fig. 6: Local rewards of PVFP with the varying learning rate and batch size in different networks.



(a) Latency of SFCs with varing lengths in the small-scale networks

(b) Latency of SFCs with varing amounts in the large-scale networks

Fig. 7: Comparisons of the average end-to-end latency among PVFP, ParaSFC, GSS, NCO and Gecode with varying SFCs in different networks.

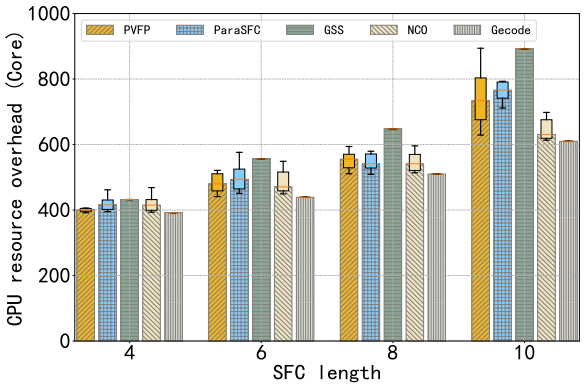
0.0001 or 0.001 in the small-scale networks, while 0.0001 or 0.0005 in the large-scale networks. The capacity of the experience buffer is set to 64 or 128. At the beginning of the learning, domain agents have a higher probability of exploring actions randomly, which are subjected to a higher $L_i(\hat{\theta})$. Along with the learning progress, agents correct the probability of exploring actions via Eq. (22) to select the optimal policy. Repeating the above process, the loss function of PVFP decreases closely to 0, in other words, all local training tends to converge.

In Fig. 5(a), the local DQN model has generally improved its optimal policy, and thus is trained to be convergent after about 6500 training epochs in the small-scale networks with more resources. It can be observed in Fig. 5(b) that PVFP has gradually reduced more network costs after about 1100 training epochs in the large-scale networks. This means that such a learning speed is adequate and has reached the convergence range of local learning. The time spent in model learning in both of the above networks is short enough to be ignored. Thus, PVFP can achieve efficient model training to evaluate actions under current states in a limited time.

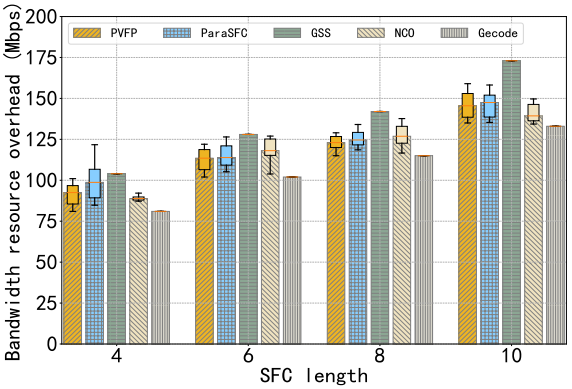
2) Comparison in Local Reward: Fig. 6(a) illustrates the local rewards of our proposed PVFP in the small-scale networks. It is clear to see that the local rewards have a great convergence performance after about 6500 and 2500 training epochs when $\alpha = 10^{-4}$ and 10^{-3}, respectively, which means that all of them have learned the hidden policies to choose the optimal actions. Furthermore, when the batch size increases from 64 to 128, local reward converges faster with $\alpha = 10^{-4}$. This is because the larger batch size mitigates the gradient oscillation of the local models, which leads to faster learning of the optimal policy.

Fig. 6(b) demonstrates the performance of local rewards in the large-scale networks. The results confirm that all local rewards converge after around 12500 epochs. Moreover, the convergence epochs decrease from 17000 to 7500 as α changes from 10^{-4} to 5×10^{-4} when the batch size is set to 64. Therefore, to better train the VNF placement model for PVFP, our simulations are executed with α=5×10^{-4} and batch size=128 hereafter, to compare PVFP with other schemes.

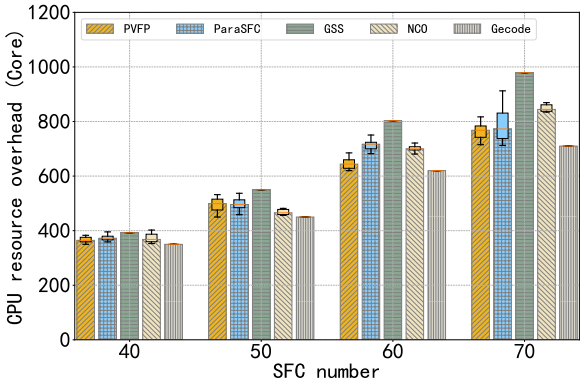
3) Comparison in Average End-to-end Latency: Figs. 7(a) and 7(b) show the average end-to-end latency obtained by



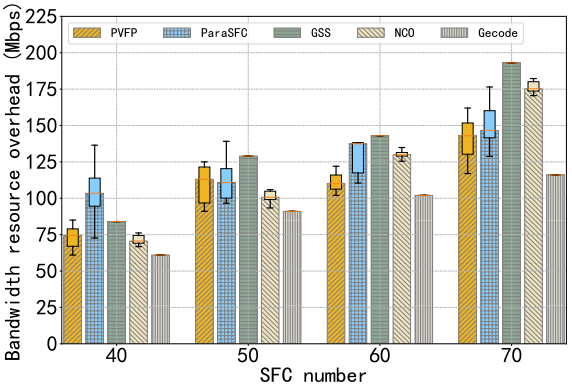
(a) CPU resource overhead of varing SFCs in the small-scale networks



(b) Bandwidth resource overhead of varing SFCs in the small-scale networks



(c) CPU resource overhead of varing SFCs in the large-scale networks



(d) Bandwidth resource overhead of varing SFCs in the large-scale networks

Fig. 8: Comparisons of the resource overhead among PVFP, ParaSFC, GSS, NCO and Gecode with varying SFCs in different networks.

PVFP, ParaSFC, GSS, NCO and Gecode as the SFC length varies from 4 to 10 in the small-scale networks and the SFC amount varies from 40 to 70 in the large-scale networks, respectively. The bars represent the average performance of these methods, with standard deviation bars set at a confidence interval of 50%.

In Fig. 7, Gecode always achieves the best performance compared to the other four approaches, regardless of the SFC requests and network scales, demonstrating its effectiveness as an optimal solution for SFC deployment with the end-to-end latency as the optimization objective. However, the tremendous computational overhead and running time of Gecode prevent it from being applied in real-world scenarios. Even so, Gecode can be suited as a benchmark to evaluate the gap between other methods and the optimal solution. NCO, on the other hand, consistently performs the worst across various SFC lengths and amounts. This can be attributed to NCO's sequential instantiation of all VNFs with a focus on minimizing overall resource consumption, without explicit consideration of end-to-end latency in its LSTM-based DRL approach. In contrast, PVFP consistently exhibits the closest performance to the optimal solution in terms of average end-to-end latency, regardless of SFC requests and network scales. For instance, in Fig. 7(b), as the number of SFCs increases from 40 to 70, PVFP outperforms ParaSFC, GSS, and NCO by average margins of 11.81%, 27.09%, and 28.90%, respectively. PVFP leverages SFC decomposition and FDRL to intelligently learn parallel VNF placement policies and effectively utilize available resources across domains, resulting in the lowest average end-to-end latency. Besides, feasible actions have been taken by PVFP to respond to the changing states. Furthermore, the superior performance of PVFP, ParaSFC, and GSS compared to NCO underscores the significant advantages of VNF parallelization in speeding up SFC processing and reducing end-to-end latency.

In addition to the average performance on end-to-end latency, Fig. 7 also provides insights into the stability of five approaches through standard deviation bars and confidence intervals. While PVFP and NCO exhibit slightly more fluctuations compared to ParaSFC, GSS, and Gecode, the average fluctuation of PVFP remains within an acceptable range. Notably, even PVFP's worst performance, i.e., the upper limit of end-to-end latency, surpasses that of ParaSFC, GSS, and NCO, bringing it closer to the optimal solution.

4) Comparison in Resource Overhead: Fig. 8 illustrates the resource overheads, including CPU and bandwidth, of PVFP in comparison to ParaSFC, GSS, NCO, and Gecode for different SFCs in various networks.

In Figs. 8(a) and 8(b), resource overheads for all methods increase as SFC length grows in the small-scale networks.

It can be seen that Gecode consistently exhibits superior performance in terms of both average CPU and bandwidth resource overheads. Conversely, GSS incurs the highest average resource overheads, while NCO, aimed at minimizing overall resource consumption, maintains slightly lower CPU and bandwidth overheads across various SFC lengths. Compared with them, PVFP and ParaSFC strike a balance with intermediate resource overheads. For instance, with the SFC length of $|\mathbb{F}|$=10, PVFP's CPU and bandwidth resource overheads are approximately 16.62% and 15.55% lower than GSS, around 7.41% and 5.49% higher than ParaSFC, as well as 17.51% and 8.71% higher than NCO, respectively. This indicates that parallelization methods, like PVFP and ParaSFC, slightly increase resource overheads in exchange for reducing execution and transmission latency. Additionally, for shorter SFCs (e.g., length 4), PVFP's resource overheads align closely with the other methods, emphasizing the role of parallelization in complex scenarios with longer requests. In Figs. 8(c) and 8(d), where the number of SFCs varies from 40 to 70 in the large-scale networks, the CPU and bandwidth overhead patterns are similar to those in Figs. 8(a) and 8(b).

In summary, PVFP maintains a moderate resource overhead while still offering advantages in end-to-end latency and practical applicability in real-world scenarios. It effectively balances resource consumption and latency optimization through its VNF relationships and local DRL optimization, outperforming ParaSFC, GSS, NCO and Gecode.

VI. Conclusions

In this paper, PVFP, a novel FDRL-based parallel VNF placement strategy, has been presented to execute optimal VNF orchestration in networks under resource constraints. The proposed PVFP involves online FL and DRL for parallel VNF placement in networks, which can realize desired network services with the guarantee of end-to-end latency in parallel as far as possible. First, PVFP designs a specifical parallel principle, including three parallelism identification rules, to reasonably decide partial VNF parallelism. Then, PVFP allocates the whole SFCs into all domains built on their remaining resources and potential parallel VNF pairs. After that, both delay-based and reward-based strategies have been designed for domain orchestrators to participate in global model training in an optimal iterative manner with quick convergence. Furthermore, by selecting potential parallelization actions, the improved DQN has been introduced to reinforce distributed local learning models for the parallel VNF placement in an efficient and scalable manner. Extensive simulation results have shown that PVFP can significantly reduce the end-to-end latency of SFCs at the medium resource expenditures.

Acknowledgment

This work is partially supported by the National Key Research and Development Program of China (No.2022YFB2702801), the Natural Science Foundation of China (No.62372192 and No.92067206), the Hong Kong RGC GRF (No.11203523), and the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie Grant Agreement (No.101030505). This article reflects only the authors' view. The European Union Commission is not responsible for any use that may be made of the information it contains.

References

[1] S. Yang, F. Li, S. Trajanovski, R. Yahyapour, and X. Fu, "Recent Advances of Resource Allocation in Network Function Virtualization," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 2, pp. 295–314, 2021.

[2] A. J. Gonzalez, G. Nencioni, A. Kamisiński, B. E. Helvik, and P. E. Heegaard, "Dependability of the NFV Orchestrator: State of the Art and Research Challenges," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 4, pp. 3307–3329, 2018.

[3] X. Shen, J. Gao, W. Wu, M. Li, C. Zhou, and W. Zhuang, "Holistic Network Virtualization and Pervasive Network Intelligence for 6G," *IEEE Commun. Surveys Tuts.*, vol. 24, no. 1, pp. 1–30, 2022.

[4] 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.

[5] Q.-T. Luu, S. Kerboeuf, A. Mouradian, and M. Kieffer, "A Coverage-Aware Resource Provisioning Method for Network Slicing," *IEEE/ACM Trans. Netw.*, vol. 28, no. 6, pp. 2393–2406, 2020.

[6] L. Yang, J. Jia, H. Lin, and J. Cao, "Reliable Dynamic Service Chain Scheduling in 5G Networks," *IEEE Trans. Mob. Comp.*, vol. 22, no. 8, pp. 4898–4911, 2023.

[7] C. Xie, S. Koyejo, and I. Gupta, "Asynchronous Federated Optimization," 2019, *arXiv:1903.03934*. [Online]. Available: https://arxiv.org/abs/1903.03934

[8] 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.

[9] B. Li *et al.*, "ClickNP: Highly Flexible and High Performance Network Processing with Reconfigurable Hardware," in *ACM SIGCOMM Comput. Commun. Rev.*, 2016, pp. 1–14.

[10] N. He, S. Yang, F. Li, S. Trajanovski, L. Zhu, Y. Wang, and X. Fu, "Leveraging Deep Reinforcement Learning With Attention Mechanism for Virtual Network Function Placement and Routing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 34, no. 4, pp. 1186–1201, 2023.

[11] D. Li, P. Hong, K. Xue, and j. Pei, "Virtual Network Function Placement Considering Resource Optimization and SFC Requests in Cloud Datacenter," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 7, pp. 1664–1677, 2018.

[12] H. K. Thakkar, C. K. Dehury, and P. K. Sahoo, "MUVINE: Multi-Stage Virtual Network Embedding in Cloud Data Centers Using Reinforcement Learning-Based Predictions," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 6, pp. 1058–1074, 2020.

[13] J. Luo, J. Li, L. Jiao, and J. Cai, "On the Effective Parallelization and Near-Optimal Deployment of Service Function Chains," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 5, pp. 1238–1255, 2021.

[14] I.-C. Lin, Y.-H. Yeh, and K. C.-J. Lin, "Toward Optimal Partial Parallelization for Service Function Chaining," *IEEE/ACM Trans. Netw.*, vol. 29, no. 5, pp. 2033–2044, 2021.

[15] C. Sun, J. Bi, Z. Zheng, H. Yu, and H. Hu, "NFP: Enabling Network Function Parallelism in NFV," in *ACM SIGCOMM Comput. Commun. Rev.*, 08 2017, pp. 43–56.

[16] 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.

[17] M. Akbari, M. R. Abedi, R. Joda, M. Pourghasemian, N. Mokari, and M. Erol-Kantarci, "Age of Information Aware VNF Scheduling in Industrial IoT Using Deep Reinforcement Learning," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 8, pp. 2487–2500, 2021.

[18] R. A. Addad, D. L. C. Dutra, T. Taleb, and H. Flinck, "Toward Using Reinforcement Learning for Trigger Selection in Network Slice Mobility," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 7, pp. 2241–2253, 2021.

[19] G. Li, H. Zhou, B. Feng, Y. Zhang, and S. Yu, "Efficient Provision of Service Function Chains in Overlay Networks Using Reinforcement Learning," *IEEE Trans. Cloud Comput.*, vol. 10, no. 1, pp. 383–395, 2022.

[20] 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.

[21] B. Farkiani, B. Bakhshi, S. A. MirHassani, T. Wauters, B. Volckaert, and F. De Turck, "Prioritized Deployment of Dynamic Service Function Chains," *IEEE/ACM Trans. Netw.*, vol. 29, no. 3, pp. 979–993, 2021.
[22] H. D. Chantre and N. L. Saldanha da Fonseca, "The Location Problem for the Provisioning of Protected Slices in NFV-Based MEC Infrastructure," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 7, pp. 1505–1514, 2020.
[23] 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.
[24] 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.
[25] A. Tomassilli, F. Giroire, N. Huin, and S. Pérennes, "Provably Efficient Algorithms for Placement of Service Function Chains with Ordering Constraints," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, 2018, pp. 774–782.
[26] H. Chen *et al.*, "MOSC: A Method to Assign the Outsourcing of Service Function Chain across Multiple Clouds," *Computer Networks*, vol. 133, pp. 166–182, 03 2018.
[27] Y. Zhang *et al.*, "Parabox: Exploiting Parallelism for Virtual Network Functions in Service Chaining," in *Proceedings of the Symposium on SDN Research*, 2017, pp. 143–149.
[28] A. Engelmann and A. Jukan, "A Reliability Study of Parallelized VNF Chaining," in *Proc. IEEE Int. Conf. Commun. (ICC)*, 2018, pp. 1–6.
[29] N. Promwongsa *et al.*, "Ensuring Reliability and Low Cost When Using a Parallel VNF Processing Approach to Embed Delay-Constrained Slices," *IEEE Trans. Netw. Services Manag.*, vol. 17, no. 4, pp. 2226–2241, 2020.
[30] D. Zheng, G. Shen, X. Cao, and B. Mukherjee, "Towards Optimal Parallelism-Aware Service Chaining and Embedding," *IEEE Trans. Netw. Serv. Manag.*, vol. 19, no. 3, pp. 2063–2077, 2022.
[31] T. P. Lillicrap *et al.*, "Continuous Control with Deep Reinforcement Learning," 2015, *arXiv:1509.02971*. [Online]. Available: https://arxiv.org/abs/1509.02971
[32] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep Reinforcement Learning: A Brief Survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.
[33] G. T. C. Schulte, M. Lagerkvist, "Gecode," [Online], http://www.gecode.org.
[34] N. Promwongsa, A. Ebrahimzadeh, R. H. Glitho, and N. Crespi, "Joint VNF Placement and Scheduling for Latency-Sensitive Services," *IEEE Trans. Netw. Sci. Eng.*, vol. 9, no. 4, pp. 2432–2449, 2022.
[35] S. Orlowski, R. Wessäly, M. Pióro, and A. Tomaszewski, "SNDlib 1.0-Survivable Network Design Library," *Networks*, vol. 55, no. 3, pp. 276–286, 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 Artificial Intelligence, Wireless Communications, Network Function Virtualization, Software-Defined Networking and Internet of Things.

Jialin Tian is currently a PhD student in Computer Science at the University of Exeter. She received the ME degree in Information and Communication Engineering from Huazhong University of Science and Technology, China in 2022, and the BS degree in Communication Engineering from Northeastern University, China, in 2019. Her research interests include Network Function Virtualization and Artificial Intelligence for networks.

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.

Hao Yin is a Professor with the Research Institute of Information Technology, Tsinghua University, Beijing, China. His research interests span broad aspects of Multimedia Communications and Computer Networks. He received the BS, ME and PhD degrees in Electrical Engineering from Huazhong University of Science and Technology, Wuhan, China, in 1996, 1999 and 2002, respectively. Prof. Yin was awarded the State Natural Science Award (Second Prize) in 2011 and the State Technological Invention Award (Second Prize) in 2012, and was appointed as a Yangtze River Scholar in 2021.

Cheng Zeng is 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 Professor with School of Computer Science and Technology, University of Science and Technology of China. Before that, he was 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 Quantum Networks.

Dapeng Oliver Wu is currently a Chair Professor at the Department of Computer Science, City University of Hong Kong. He received the PhD degree in Electrical and Computer Engineering from Carnegie Mellon University, Pittsburgh, PA, in 2003, and BE degree in Electrical Engineering from Huazhong University of Science and Technology, Wuhan, China, in 1990. His research interests are in the areas of Networking, Communications, Signal Processing, Computer Vision, Machine Learning, Smart Grid, and Information and Network Security. He has served as an Editor in Chief of IEEE TNSE, Associate Editor for IEEE TCC, ToC, TWC and TVT, and a Guest-Editor for IEEE JSAC. He is an IEEE Fellow.