# Leveraging Deep Reinforcement Learning with Attention Mechanism for Virtual Network Function Placement and Routing

Nan He, Song Yang, *Senior Member, IEEE,* Fan Li, *Member, IEEE,* Stojan Trajanovski, *Member, IEEE,* Liehuang Zhu, *Senior Member, IEEE,* Yu Wang, *Fellow, IEEE* and Xiaoming Fu, *Fellow, IEEE*

**Abstract**—The efficacy of Network Function Virtualization (NFV) depends critically on (1) where the virtual network functions (VNFs) are placed and (2) how the traffic is routed. Unfortunately, these aspects are not easily optimized, especially under time-varying network states with different QoS requirements. Given the importance of NFV, many approaches have been proposed to solve the VNF placement and Service Function Chaining (SFC) routing problem. However, those prior approaches mainly assume that the network state is static and known, disregarding dynamic network variations. To bridge that gap, we leverage Markov Decision Process (MDP) to model the dynamic network state transitions. To jointly minimize the delay and cost of NFV providers and maximize the revenue, we first devise a customized Deep Reinforcement Learning (DRL) algorithm for the VNF placement problem. The algorithm uses the attention mechanism to ascertain smooth network behavior within the general framework of network utility maximization (NUM). We then propose attention mechanism-based DRL algorithm for the SFC routing problem, which is to find the path to deliver traffic for the VNFs placed on different nodes. The simulation results show that our proposed algorithms outperform the state-of-the-art algorithms in terms of network utility, delay, cost, and acceptance ratio.

**Index Terms**—Network function virtualization, deep reinforcement learning, placement, routing.

## 1 Introduction

Network functions (NFs) such as firewalls and load balancers have been implemented on physical devices, which are called middleboxes. Although such middleboxes are designed to effectively improve network security and performance, these devices are costly, lack flexibility, and are difficult to operate. Network Functions Virtualization (NFV) has emerged as an innovative technology, which can address these challenges by decoupling network functions from dedicated hardware and implementing them as Virtual Network Functions (VNFs) [2]. Recently, NFV attracts a great deal of interest from the networking community as this technology shows great potential to promote openness, innovation, flexibility, and scalability in the network [3], [4]. To build more complex services, the concept of Service Function Chaining (SFC) can be used, where a series of VNFs must be processed in a predefined order to collectively deliver a specific service. Therefore, one important issue is to determine where to place the VNFs in a predefined order to meet the service requirements. Considering about the capacitated network resources, the main goal is to maximize the network utility. Network Utility Maximization (NUM) is therefore a mathematical framework for allocating network resources so that the total utility is maximized subject to network resource constraints [5]. While successful VNF placement provides a simple route for the SFC, new issues arise with it. For example, in an NFV-enabled environment, multiple VNFs will be placed in different locations. Therefore, another important issue is the SFC routing problem, which is defined as how to find a path from the source to the destination with the goal of maximizing the number of accepted requests. The path during an SFC must pass through all the required VNFs in order to meet the bandwidth and delay requirements. In this paper, we refer to the VNF placement and SFC routing problem as the VNF-PR problem. The overarching objective of the VNF-PR problem is to achieve NUM. Therefore, cost and QoS (e.g., delay) schemes need to be jointly considered, which will lead to better user experience and higher profitability.

To solve the VNF placement and SFC routing problem, existing work resorts to linear programming [6] or proposes heuristic or approximate methods [7] to transform the problem into some well-known NP-hard problems [8] such as the knapsack problem. Although these approaches can solve the VNF placement and SFC routing problem to some extent, they usually use prior knowledge (e.g. delay, bandwidth, and demand changes) to develop effective VNF placement and SFC routing strategies. For this reason, these approaches are not feasible in an NFV-enabled environment with dynamic network states, as the VNF placement and SFC routing problem needs to be rehandled once the network states have varied significantly. For example, existing work formulates a one-shot optimization problem in


- *Nan He, Song Yang, and Fan Li are with the School of Computer Science and Technology, Beijing Institute of Technology, Beijing, 100081, China. Email: {henan, S.Yang, fli}@bit.edu.cn*
- *Stojan Trajanovski is with Microsoft, London, W2 6BD, United Kingdom. E-mail: sttrajan@microsoft.com*
- *Liehuang Zhu is with the School of Cyberspace Security, Beijing Institute of Technology, Beijing 100081, China. E-mail: liehuangz@bit.edu.cn*
- *Yu Wang is with Department of Computer and Information Sciences, Temple University, Philadelphia, Pennsylvania 19122, USA. Email: wangyu@temple.edu*
- *Xiaoming Fu is with the Institute of Computer Science, University of Göttingen, Göttingen, 37073, Germany. E-mail: Fu@cs.uni-goettingen.de*

- *A preliminary version of this work appeared in [1].*

a dynamic environment [9]. However, the existing work does not have the potential for positive long-term returns in response to dynamic network environments and leads to excessive waste of resources and performance degradation due to irrational allocation. Besides, existing work requires multiple iterations to solve the VNF placement and SFC routing problem, which leads to high computational complexity and resource overheads. Thus existing efforts cannot effectively solve the complex VNF placement and SFC routing problems in dynamic environment.

Another line of work applies (Deep) Reinforcement Learning (DRL) [10] to solve the VNF placement and SFC routing problem, since DRL can capture the dynamic network state transitions and user demands. More specifically, an RL agent interacts with the dynamic NFV-enabled environment by implementing placement and routing strategies. The RL agent then continuously optimizes the strategies based on the reward values (e.g., delay, capacity, and bandwidth) fed back from the environment. Nonetheless, RL methods still suffer from the impractical and inefficient issues brought by large networks. On the contrary, deep neural networks can be applied to high-dimensional state space. Different from these works, in this paper we adopt Deep Deterministic Policy Gradient (DDPG) [11] algorithm to deal with the high-dimensional and time-varying network state and complex networks environment. The reason is that the DDPG algorithm replaces the value function with a deep network that can cope with high-dimensional input states. Moreover, the DDPG algorithm contains a policy network to generate actions, which can be applied to the output of continuous actions and large action space. We are motivated to leverage the feature of deep neural networks and introduce a Markov Decision Process (MDP) to capture the dynamic network state transitions and process them within a DRL architecture. A DRL agent usually does not pay equal attention to all available placement nodes. For example, an agent typically selects the current action based on information with a higher level of cognitive skill and ignores other perceptible information.

In this paper, we first study the VNF placement problem assuming that a complete graph is adopted as a problem input. Then, we introduce the concept of attention mechanism, which is widely used in the image processing problem [12] and machine translation [13] to model the agent's action for DDPG. We find that during the training process of DDPG, the attention mechanism will automatically focus on the feasible neighbor node that may affect the agent's selection behavior. It ultimately helps to reduce the attention to other unnecessary nodes and improve the training efficiency of the model. With this motivation, we design a customized attention mechanism-based DDPG to train our DRL model as in our previous work [1]. Subsequently, we additionally study the SFC routing problem, which is to find a path from a source to a destination to maximize the number of accepted requests. In particular, we adopt a pointer network, which is a simplified structure of an attention mechanism that uses a *softmax* probability distribution as a "pointer" to the node (i.e., the next-hop routing policy) with the highest weight. In reality, the service provider can first solve the SFC routing problem via the proposed algorithm for different node pairs in the network. Those returned solutions serve as the input for the VNF placement problem. At last, the service provider can solve the VNF placement problem by using the proposed solutions. This is one possible scenario of how a practitioner can jointly solve the VNF placement and SFC routing problem and apply our proposed solutions in sequential order. The main contributions of this paper are:

- We first formulate the VNF placement problem as an optimization model and establish a utility function aiming to solve a trade-off between revenue and cost.
- We propose a novel Attention mechanism-based Deep Deterministic Policy Gradients (A-DDPG) framework to solve the VNF placement problem, using the Actor-Critic network structure, in which both the Actor and Critic networks adopt double networks (namely the main network and the target network).
- We then propose an Actor-Critic-based learning algorithm called P-AC to solve the SFC routing problem using the Pointer network structure. P-AC uses attention as a "pointer" to select the node with the highest probability as the next-hop node for path selection at each time.
- Through extensive simulations, we show that our proposed algorithms outperform the state-of-the-art algorithms in terms of network utility, delay, cost, and acceptance ratio.

The remainder of this paper is organized as follows. Section 2 describes the related work. Section 3 defines the VNF placement problem and we devise the A-DDPG algorithm to solve the VNF placement problem in Section 4. The proposed A-DDPG algorithm is evaluated via simulations in Section 5. Then, we define the SFC routing problem and devise the P-AC algorithm to solve the SFC routing problem in Section 6. We also provide the simulation results in this section. Finally, we conclude in Section 7.

## 2 Related Work

### 2.1 Combinatorial optimization theory for VNF placement and/or SFC routing

**VNF Placement**: The VNF placement problem has been studied with the aim of cost minimization [14], performance improvement [15], and utility maximization [16]. For instance, Li *et al.* [17] leverage a correlation-based greedy algorithm to solve the VNF placement problem in the cloud data center. A Minimum-Residue heuristic is presented in [18] for VNF placement in a multi-cloud scenario with constraints of deployment cost. Furthermore, Cziva *et al.* [19] study how to use the optimal stopping theory to place VNFs under the edge cloud to minimize the total delay expectation. However, the authors in [19] assume that one VNF is sufficient to meet the users' requirements.

**SFC routing**: Feng *et al.* [20] formulate the routing problem as an Integer Linear Programming (ILP) and then devise an approximate algorithm to effectively consolidate flows into a limited number of active resources. By using the resource-aware algorithm, Hong *et al.* [21] formulate the SFC routing problem as a Binary Integer Programming

(BIP) model aiming to minimize the resource consumption costs of flows with SFC requests. Besides, Wang *et al.* [22] formalize the SFC routing problem by jointly optimizing the update delay and the load of VNF. Then they design an algorithm based on randomized rounding to solve it with a bounded approximate ratio.

**VNF placement and SFC routing**: Ma *et al.* [23] solve the placement and routing problem based on the heuristics algorithm for the general scenario of a non-ordered or ordered middlebox set. By using a primal-dual-based algorithm, a system model is designed in [24] for accomplishing the VNF placement and SFC routing in practical operator Data Center Networks (DCNs). Yang *et al.* [25] propose an efficient randomized rounding approximation algorithm for VNF placement and SFC routing, to minimize the maximum link load ratio. Gao *et al.* [14] propose a cost-efficient scheme to address the VNF placement and SFC routing problem in public cloud networks with the goal of low cost and delay. Nevertheless, the above work utilizes combinatorial optimization theory that is often slow and impractical with the dynamic network variations.

### 2.2 DRL for VNF placement and/or SFC routing

**VNF placement**: There are also some studies that solve the VNF placement problem by using DRL. For instance, Manabu *et al.* [26] propose an accelerated reinforcement learning method to shorten the delivery time of services. According to [26], the reinforcement learning agent learns the optimal placement strategy of VNFs according to the state value function and simulates the model in various environments. Su *et al.* [27] combine the DRL and GNN to solve the VNF placement problem with the minimum deployment cost. Akbari *et al.* [28] minimize cost and age of information in terms of network resources and solve the VNF placement problem. However, the methods in [26]–[28] ignore the end-to-end delay, especially the processing delay, as we do in this paper.

**SFC routing**: Pei *et al.* [29] formulate the SFC routing problem as a BIP model aiming to minimize the end-to-end delay for each SFC request. Then, a novel two-phase algorithm based on deep learning technology is proposed to handle the SFC routing problem. Gu *et al.* [30] adopt a deep deterministic policy gradients based algorithm for SFC routing, aiming to minimize the deployment cost based on geographic location and the processing delay. However, the authors in [30] suppose that all VNF instances have already been placed on network nodes, so the SFC routing problem is simplified to the deployment of paths and allocation of traffic load on the links.

**VNF placement and SFC routing**: By using the DQN algorithm, Xiao *et al.* [31] solve VNF placement and SFC routing problem for the optimization of throughput and operation cost. The DDPG algorithm is adopted in [32] to solve the VNF placement and SFC routing problem, aiming to improve the inter-domain load balancing capabilities in multiple non-cooperative domains. Moreover, Tong *et al.* [33] propose a Gated Recurrent Units (GRU)-based traffic prediction model and place VNF and routing in advance based on the prediction result. They apply a DRL algorithm called Asynchronous Advantage Actor-Critic (A3C) to train the agent and then obtain the optimal strategy.

Nevertheless, none of the aforementioned works consider the impact of surrounding nodes' resources on network states. In fact, the importance of neighbors to the learning agent is distinguishable according to their remaining resources in the DRL model. The attention mechanism enables to focus on neighbor nodes with sufficient resources and contributes to the generation of neighbor interaction behaviors. Our proposed A-DDPG and P-AC are verified to be able to efficiently and jointly solve the VNF placement and SFC routing problem by applying the attention mechanism to the DRL architecture and using the Actor-Critic network structure.

## 3 Network Model and the VNF Placement Problem Analysis

In this paper, our goal it to jointly consider the VNF placement and SFC routing problem. Due to the complexity of this problem, we propose to solve it sequentially. Our rationale is to first solve the SFC routing problem and then use the returned solutions as VNF placement problem input to solve the VNF placement problem eventually. Due to the reason that the VNF placement problem is more representative and complex than the SFC routing problem, we first present to study the VNF placement problem, assuming the routing information is already calculated by using the solutions for solving the SFC routing problem proposed in Section 6. As a result, in this section we begin with the VNF placement problem by describing the network model, network utility model, and cost model, respectively, in Section 3.1, 3.2 and 3.3. Subsequently, we formulate the VNF placement problem with the objective and constraints in Section 3.4. For the convenience of reading, we summarize the notations used in Table 1.

### 3.1 Network model

Firstly, we consider a physical network which is presented as a graph $\mathcal{G} = \{\mathcal{N}, \mathcal{E}\}$, where $\mathcal{N}$ is the set of nodes, and $\mathcal{E}$ is the link set connecting each of two nodes[1]. We mainly consider two kinds of resource constraints, including node and link resources constraints. Each node $n \in \mathcal{N}$ has a computing resource capacity (i.e., CPU cycles per second), which is denoted as $\delta_n$, $d_n$ represent the delay of node $n \in \mathcal{N}$. Meanwhile, $\eta_e$ represent the capacity of link $e \in \mathcal{E}$ and $d_{u,v}$ indicates the delay between nodes $u$ and $v$.

We denote the set of requests by $\mathcal{R} = \{r_1, r_2, ...r_l\}$, and each request by $r_l(s_l, d_l, \xi, \mathcal{F}_l, D_l) \in \mathcal{R}$, each $r_l$ transmits its packets from a source node $s_l$ to a destination node $d_l$ at a flow rate $\xi$ in required delay $D_l$. Meanwhile, $\mathcal{F}_l = \{f_{l1}, f_{l2}, ..., f_{lk}\}$ is used to indicate the VNFs set, where $f_{lk}$ is the $k$th virtual network functions in service function chain $\mathcal{F}_l$. And we use the high-order matrix $k * N$ to represent the deployment status of VNF on a physical server. Fig. 1 illustrates an example of service function chains. A service function chain is composed of several network functions. In a network function virtualization environment, these

1. We first assume that a complete graph is used in the VNF placement problem. Later, in Section 6, we will demonstrate how to solve the SFC routing problem to find the path and serve as the input for the VNF placement problem.

TABLE 1
Notations.

| Variable | Definition |
|---|---|
| $\mathcal{G}$ | Physical network |
| $\mathcal{N}, \mathcal{E}$ | The set of nodes and links of the network |
| $\mathcal{R}$ | The set of requests. For each $r(\xi, \mathcal{F}, D) \in \mathcal{R}$, $\xi$ indicates the flow rate, $\mathcal{F}$ represents a set of requested VNFs, $D$ denotes the requested delay |
| $\mathcal{F}_r$ | The set of requested VNFs of $r \in \mathcal{R}$ |
| $u^{re}_{s,a}, u^{c}_{s,a}$ | The revenue function and cost function |
| $T^r$ | The total delay of network request $r \in \mathcal{R}$ |
| $C^f_n$ | The required processing capacity for $f \in \mathcal{F}_r$ on node $n \in \mathcal{N}$ |
| $C_{u,v}$ | The link capacity between $u$ and $v$ |
| $s_k, d_k$ | Source and destination of request $r \in \mathcal{R}$ |
| $d^f_n$ | Processing delay for $f \in \mathcal{F}_r$ on node $n$ |
| $\delta_n,\ \eta_e$ | Capacity of node $n \in \mathcal{N}$ and link $e \in \mathcal{E}$ |
| $\Psi$ | The expected service payment from consumers |
| $S, A, R$ | The state space, action space, and reward |
| $x^{r,f}_n$ | A Boolean variable. It is 1 if $r$'s requested VNF $f$ is placed on $n$; and 0 otherwise |
| $y^r_{u,v}$ | A Boolean variable. It is 1 if path between $u$ and $v$ is used for delivering the requested task of $r$; and 0 otherwise |
| $k_i, v_i, q_i$ | The key, value and query for node $i \in \mathcal{N}$ |
| $sc^j_i$ | The compatibility of query $q_i$ with key $k_j$ |
| $\theta^\mu$ and $\theta^Q$ | The weights of actor and critic networks |
| $l^o_{u,v}, b^o_{u,v}, t^o_{u,v}$ | The distance, bandwidth and traversing delay from $u$ to $v$ |

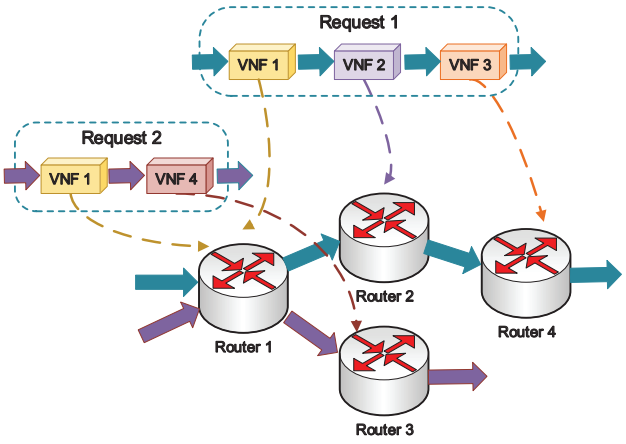



Fig. 1. System Framework.

network functions are virtualized as VNFs. Then, an ordered combination of several VNFs comprises a service function chain.

### 3.2 Network Utility Model

In this section, we formulate the network utility model which refers to a function of revenue and cost. More specially, we define the utility function $U$ as:

$$U = u^{re}_{s,a} - u^{c}_{s,a} \tag{1}$$

where $u^{re}_{s,a}$ is the revenue function, and $u^{c}_{s,a}$ is the total cost. We use the concept of Shannon's entropy [34] and define the revenue function as:

$$u^{re}_{s,a} = \sum_{r \in \mathcal{R}} y^r_{u,v} \cdot \xi \cdot \Psi - \sum_{r \in \mathcal{R}} (-\frac{1}{T^r} \log \frac{1}{T^r}) \tag{2}$$

where $\xi$ represents the traffic of the request $r$, $y^r_{u,v}$ It is 1 if path between $u$ and $v$ is used for delivering the requested task of $r$, and 0 otherwise. $\Psi$ is the expected service revenue from consumers according to the Service Level Agreement (SLA) [35]. $T^r$ represents the total delay of network request $r$, and it is the sum of the processing delay of all nodes. The purpose of using information entropy is to unify service revenue and delay into an order of magnitude to ensure the additivity between data. The transmission delay in the SFC is defined as:

$$T^r = \sum_{f \in \mathcal{F}_r} x^{r,f}_n \cdot d^f_n + \sum_{e^{u,v} \in \mathcal{E}} y^r_{u,v} \cdot d_{u,v} \quad \forall u, v, n \in \mathcal{N} \tag{3}$$

where $d^f_n$ represents the processing delay for $f \in \mathcal{F}_r$ on node $n$, $d_{u,v}$ indicates the delay between nodes $u$ and $v$.

### 3.3 Cost Model

The cost function $u^c_{s,a}$ includes three parts: operation cost, deployment cost, and transmission cost.

#### 3.3.1 Operation cost

Each physical node needs to complete the preparatory work before deploying VNFs, such as the pre-configuration of different types of VNFs. We define the unit operating cost as $\Phi^{op}$, then the total operation cost is defined as:

$$u^{op}_{s,a} = \sum_{n \in \mathcal{N}} x^{r,f}_n \cdot \Phi^{op} \quad \forall r \in \mathcal{R}, f \in \mathcal{F}_r \tag{4}$$

#### 3.3.2 Deployment cost

The deployment cost of a server is directly proportional to the resources consumed. Therefore, we stipulate that VNF deployment cost is mainly generated by the server of the deployed function. If no VNF is placed on the server, the deployment cost is not considered. We define the unit deployment cost as $\Phi^{de}$. The total deployment cost is defined as:

$$u^{de}_{s,a} = \sum_{n \in \mathcal{N}} x^{r,f}_n \cdot \Phi^{de} \quad \forall r \in \mathcal{R}, f \in \mathcal{F}_r \tag{5}$$

#### 3.3.3 Transmission cost

The transmission cost is the communication cost for transferring traffic between nodes. In practice, the deployment cost is negatively correlated with the transmission cost [36]. We define the transmission unit cost as $\Phi^{tr}$. The total transmission cost is defined as:

$$u^{tr}_{s,a} = \sum_{e \in \mathcal{E}} y^r_{u,v} \cdot \xi^e_r \cdot \Phi^{tr} \quad \forall r \in \mathcal{R}, u, v \in \mathcal{N} \tag{6}$$

Finally, $u^c_{s,a}$ is a combination of above mentioned three kinds of cost:

$$u^c_{s,a} = u^{op}_{s,a} + u^{de}_{s,a} + u^{tr}_{s,a} \tag{7}$$

### 3.4 Problem Definition and Formulation

Formally, the VNF placement problem can be defined as follows:

***Definition 1.*** Given are a network $\mathcal{G} = \{\mathcal{N}, \mathcal{E}\}$ and a set of requests $\mathcal{R}$, for each request $r(\xi, \mathcal{F}, D) \in \mathcal{R}$, the VNF placement problem is to place the VNFs on $N$, such that the network utility $U$ is maximized.

The VNF placement problem is known to be NP-hard [8]. We first formally present the VNF placement problem with objectives and constraints. We begin with some necessary variables.

**Boolean Variables:**

$x_n^{r,f}$: It is 1 if $r$'s requested VNF $f$ is placed on $n$; and 0 otherwise.

$y_{u,v}^r$: It is 1 if the path between $u$ and $v$ is used for delivering the requested task of $r$; and 0 otherwise.

**Placement Objective:**

$$\max\ U \tag{8}$$

**Placement Constraint:**

$$\sum_{n\in\mathcal{N}} x_n^{r,f} = 1\ \ \forall r \in \mathcal{R}, f \in \mathcal{F}_r \tag{9}$$

**Node Capacity Constraint:**

$$\sum_{r\in\mathcal{R}} \sum_{f\in r} x_n^{r,f} \cdot C_n^f \leq \delta_n\ \ \forall n \in \mathcal{N} \tag{10}$$

**Delay Constraint:**

$$\sum_{f\in r} \sum_{n\in\mathcal{N}} x_n^{r,f_i} \cdot d_n^f \leq D\ \ \forall r \in \mathcal{R} \tag{11}$$

Eq. (8) maximizes the network utility. Eq. (9) ensures that for each requested VNF $f$, it must be placed on one node in the network. Eq. (10) indicates that each node's capacity is not violated. Eq. (11) ensures that for each request the total delay does not exceed $D$.

Without considering the variability of network states, this optimization problem can be solved by using ILP or heuristic algorithms [8]. However, it is non-trivial to use those techniques to model dynamic metrics. DRL can capture the dynamic states of the networks. Therefore, in Section 4, we exploit DRL to solve the VNF placement problem.

## 4 DRL-based VNF Placement Algorithm

In this section, we begin with the DRL model design in Section 4.1. In fact, this is a Markov decision process including state, action, and reward. Then we demonstrate our proposed A-DDPG algorithm to solve the VNF placement problem in Section 4.2, 4.3 and 4.4.

### 4.1 DRL Model Design

In the DRL model, we first model the VNF placement problem using a Markov decision process. The Markov decision process can be represented as a tuple $(S, A, R)$ with the state space, action space and reward composition. In particular, we consider a discrete-time period $T$ for dynamic changes in the state of the network resources. At each time step $t \in T$, the DRL agent chooses an action for placing the VNF and receives the state and reward by interacting with the environment. Accordingly, the 3-tuple $(S, A, R)$ in DRL for the VNF placement problem are defined as follows:

**State**: The state space can be described by a vector $S = \{s_1, s_2, s_3, ..., s_T\}$, where each term $s_t \in S$ represents the remaining resources of the virtual links and nodes at time $t$. $T$ represents a periodic discrete time.

**Action**: The action of any agent is a vector $A$ with each term $a \in A$ representing VNF placement. Therefore, we define an action as $a = \{x_n^{r,f},\ \xi_r^f\}\ \ \forall r \in \mathcal{R}, f \in \mathcal{F}_r,\ n \in \mathcal{N}$.

**Reward**: The reward is a value indicating correct action. Whether the action can bring profit and whether the user's demand is met is taken as the criteria to affect the reward value. The reward received at time-slot $t$ is set as the objective of our utility function, defined as $R = \sum_{r\in\mathcal{R}} y_{u,v}^r \cdot \xi \cdot \Psi - \sum_{r\in\mathcal{R}}(-\frac{1}{T^r}\log\frac{1}{T^r}) - u_{s,a}^{op} + u_{s,a}^{de} + u_{s,a}^{tr}$ according to Eq. (1). We put the constraints in the reward function as a penalty factor. If the action brings benefits to the network and saves cost, the reward will be a positive value to encourage the operation. However, if the cost increases or the constraint is violated, a negative reward is returned. Consequently, this method affects the action decision to make sure it does not violate these constraints.

### 4.2 Attention Model

We argue that the neighbor nodes of each server node are of great significance to the performance of VNF placement. Therefore, we introduce an attention mechanism into the neural network, which allows the network to better obtain neighbor node information. The attention mechanism assumes that the weights of nodes measure the matching degree between neighbors in the attention layer.

Formally, we define $h_i$ as the state of the node, and the key $k_i$, value $v_i$, and $q_i$ can be calculated as follows:

$$k_i = W^K \cdot h_i, v_i = W^V \cdot h_i, q_i = W^Q \cdot h_i\ \ \forall i \in \mathcal{N} \tag{12}$$

where the $W^Q$, $W^K$ and $W^V$ are the parameter matrices that can be learned, and $h_i$ is equal to $s_i$ defined in section 4. The compatibility $sc_i^j$ of the query $q_i$ of node $i$ with the key $k_j$ of node $j$ is calculated as the active function (e.g., dot-product):

$$sc_i^j = active(q_i, k_j)\ \ \forall i, j \in \mathcal{N} \tag{13}$$

According to Eq. (13), we compute the weights $a_i^j$ using a *softmax* function:

$$a_i^j = softmax(sc_i^j) = \frac{e^{sc_i^j}}{\sum_{i=1}^N e^{sc_i^j}}\ \ \forall i, j \in \mathcal{N} \tag{14}$$

Then the attention value is equal to:

$$at((k_i, v_i), q_i) = \sum_{i=1}^{\mathcal{N}} a_i^j \cdot v_i = \sum_{i=1}^{\mathcal{N}} \frac{e^{sc_i^j}}{\sum_{i=1}^{\mathcal{N}} e^{sc_i^j}} \cdot v_i\ \ \forall i, j \in \mathcal{N} \tag{15}$$

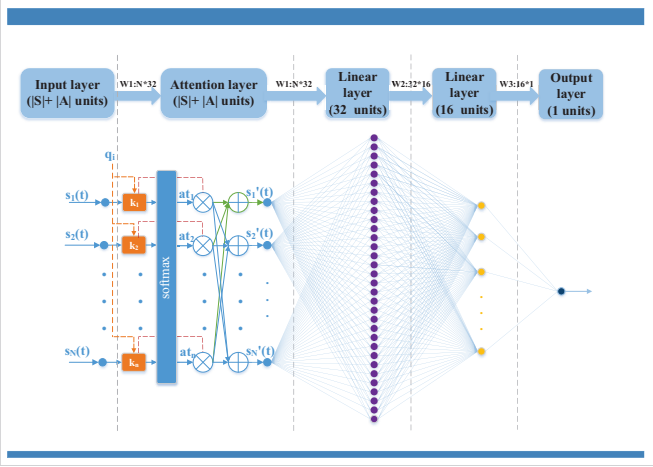



Fig. 2. Actor-Critic Network Design of A-DDPG Framework.

### 4.3 Placement Network Model

We propose the A-DDPG algorithm based on the Actor-Critic network structure, which can solve the VNF placement problem defined in Eq. (8)-(11). The Actor-Critic network structure involves two neural networks: actor network and critic network. They are two different networks in the A-DDPG algorithm. Actor network is a policy network for making good decisions on VNF placement, while critic network is a value function network used to estimate the current Actor's policy. Specifically, the Actor and Critic networks both construct two neural networks with the same structure but different parameters, namely the evaluation and target networks. As demonstrated in Fig. 2,the Actor-Critic network structure is represented as an attention layer and a multi-layer fullyconnected neural network with two hidden layers. We include the attention mechanism in the A-DDPG since it can learn the hidden states of inputs more effectively.

The cumulative reward is used as the target value, while the expected cumulative reward is used as the projected value by the network. The goal of training is to get the predicted value as near to the target value as possible. The loss function is defined by the following equation:

$$L(\theta) = \frac{1}{B}\sum_{t}(y_t - Q(s_t, a_t|\theta^Q)^2) \tag{16}$$

where $\theta$ represents the parameter of actor for sampling and $B$ indicates the size of the replay buffer. Then, the partial derivative of the loss function to the weight of the neural network can be calculated as:

$$\frac{\partial L(\theta)}{\partial \theta} = \frac{1}{B}\sum_{t}(y_t - Q(s_t, a_t|\theta^Q)^2)\frac{\partial Q(s_t, a_t|\theta^Q)}{\partial \theta} \tag{17}$$

where $Q(s_t, a_t|\theta^Q)$ refers to the long-term return of an action, taking a specific $a_t$ under a specific policy from the current state $s_t$, and $y_t$ denotes the predicted return. Through multiple iterations of the gradient descent method and the back-propagation mechanism, the $Q(s_t, a_t|\theta^Q)$ value can be obtained.

**Algorithm 1** A-DDPG Training Procedure

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor network $\mu(s|\theta^\mu)$ with weights $\theta^Q$ and $\theta^\mu$
Initialize target network $Q^{'}$ and $\mu^{'}$ with $\theta^{Q^{'}}$, $\theta^{\mu^{'}}$
Initialize replay buffer $\mathbb{B}$
**for** $episode = 0, 1, ..., M$ **do**
  Initialize a random process $\mathbb{N}$ for action exploration
  Receive initial observation state $s_1$
  **for** $t = 0, 1, ..., T$ **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathbb{N}_t$ according to the current policy $\theta^\mu$ and exploration noise $\mathbb{N}_t$
    Execute action $a_t$ and observe reward $r_t$ and observe new state $s_{t+1}$
    Store a random minibatch of $N$ transitions $(s_t, a_t, r_t, s_{t+1})$ from $\mathbb{B}$
    Set $y_t = r_t + \gamma Q^{'}(s_{t+1}, \mu^{'}(s_{t+1}|\theta^{\mu^{'}})|\theta^{Q^{'}})$
    Update critic by minimizing the loss:
      $L = \frac{1}{B}\sum_t(y_t - Q(s_t, a_t|\theta^Q)^2)$
    Update the actor policy using the sampled gradient:
      $\bigtriangledown_{\theta^\mu} J \approx \frac{1}{B}\sum \bigtriangledown_a Q\left(s_t, a_t|\theta^Q\right) \bigtriangledown_{\theta^\mu} \mu\left(s|\theta^\mu\right)|_{s_t}$
    Update the target network:
      $\theta^{Q^{'}} \leftarrow \tau\theta^Q + (1-\tau)\,\theta^{Q^{'}}$,
      $\theta^{\mu^{'}} \leftarrow \tau\theta^\mu + (1-\tau)\,\theta^{\mu^{'}}$
  **end for**
**end for**

### 4.4 Algorithm Design

We further describe the entire process from observation to running execution to better understand A-DDPG framework in Fig. 3.

**Store the training data.** The DRL agent interacts with the environment in Step 1, which is the first stage of observation. This step is mostly for obtaining the initial state and storing historical samples. On the one hand, the agent receives the environment's original state, including the server's placement status, and gathers environmental history samples that must be trained. The initial state, action, reward, and the next state are all included in these samples. On the other hand, the samples are then stored in the replay memory. Following that, the action is achieved using the greedy technique, as the neural network parameters are also randomly initialized, the parameters will not be updated at this step, and they are collectively called random actions. Then, according to the number of iterations, $epsilon$ is lowered. The simulator then performs the selected action, returning a new state and reward.

**Input the sampling data.** The replay buffer is introduced into the network in step 2 of the observation process, and the network is trained using replay buffer data with the goal of minimizing correlation between samples. More specifically, the previous state $s_t$, action $a_t$, new state $s_{t+1}$, and reward $r_t$ are assembled into $(s_t, a_t, r_t, s_{t+1})$ to enter the replay memory for parameter updating. Finally, using the $epsilon$ greedy strategy, the next action is chosen, and the cycle is repeated until the number of iterations approaches the limit, which is determined by the size of the replay buffer.

**Traning the A-DDPG network model.** Following the observation process, sufficient samples required for A-DDPG

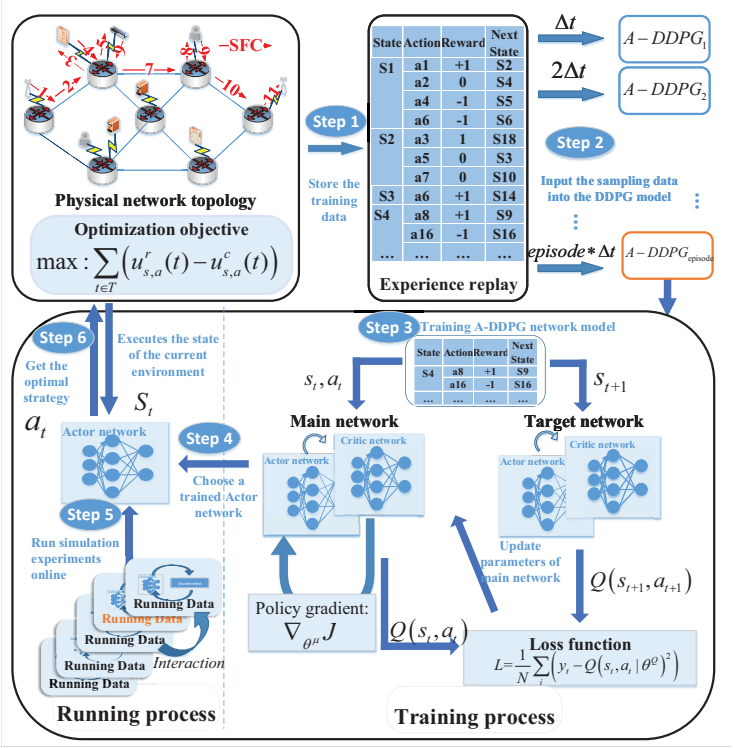



Fig. 3. The A-DDPG Framework.

training are obtained. Algorithm 1 describes the training process of Step 3 in Fig. 3. The A-DDPG agent, in further detail, initially sets the weights of the critic $Q(s,a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ networks as $\theta^Q$ and $\theta^\mu$, respectively (Line 1). After that, The A-DDPG agent additionally initializes a target network and replay buffer (Line 2-3). For each training episode (Line 4), the agent receives the current environment's state after adding exploration noise $N$ (Line 5-6). The DRL agent selects the action of VNF placement depending on the current policy (Line 7-8). The agent then takes action $a_t$, receive the rewards $r$ and the next state $s_{t+1}$ (Line 9). The state experience tuple $(s_t, a_t, r_t, s_{t+1})$ is then saved into an experience replay buffer (Line 10). Next, the actor network and the critic network will be updated (11-14). First we need to prepare significant training data: (1) calculate the predicted baseline $y_t$ (Line 11) and (2) calculate the policy $\mu(s_t|\theta^\mu) + \mathbb{N}_t$ and Q baseline $Q(s_t, a_t|\theta^Q)$. The mean squared error (MSE) of the predicted baseline and actual baselines is the value loss function (Line 12). After then, the neural network is trained using gradient descent (17) (Line 13), and the network's weight parameters are updated regularly (Line 14). The processes above iterate until convergence or the predefined episode is complete. Notably, $M$ is the total number of episodes, while $T$ denotes the number of training rounds in each episode.

**Choose a trained Actor network.** After obtaining its well-trained A-DDPG model training, the long-term cumulative reward of the action is preliminarily evaluated by inputting the current state in step 4.

**Run simulation experiments online.** The goal is to decrease the selection probability of the current actions with poor performance for optimizing the solution space. Step 5 is to run simulation experiments online by interacting with the environment. In each interaction, the reward of each placement strategy in the solution space is evaluated against the predicted values.

**Get the optimal strategy.** In Step 6, the agent in the A-DDPG model learns to perform the correct action (i.e. the placement strategy) in each state so that the agent can receive a good reward.

## 5 Performance Evaluation

### 5.1 Simulation Settings

The simulations are all implemented on an Intel (R) Core (TM) i7 Windows 10 64-bit system. Moreover, the network parameters, computing capabilities, and traffic requests are randomly generated as follows, which is similar to existing works [37], [38]. The basic environment of the simulation is set up to construct a network computing platform composed of 50 servers, which has $[1, 100]$ units of capacity. For each link in the network, its capacity is randomly assigned from the range $[2, 4]$ $Gb/s$ and its delay takes value in $[10, 50]$ $ms$ in the simulation. We simulate $[10, 100]$ requests and each request requires an SFC consisting of 3 to 6 different VNFs (i.e., firewall, NAT, IDS, load balancer, WAN optimizer and flow monitor) according to [39]. For the unit cost of Eqs. (4), (5), (6), we set $\Phi^{op} = 0.2$, $\Phi^{de} = 0.4$, and $\Phi^{tr} = 0.1$.

An input layer, an output layer, and 3 hidden layers make up the structure of our attention-based deep neural network. The 3 hidden layers comprise an attention layer and 2 fully connected layers. The two fully connected layers each include 32 and 16 hidden nodes, respectively. The hyperparameters for DRL are shown in Table 2, and the target network parameters are updated once every 200 steps.

TABLE 2
Hyper parameters for DRL.

| | | | |
|---|---|---|---|
| **Buffer size** | 10000 | **Learning rate** | $0.1, 0.01, 0.001$ |
| **Hidden nodes** | $32, 64$ | **Number of episodes** | 3000 |
| **Discounted factor** | 0.8 | **Hidden layer** | 3 |

The implementation of the A-DDPG algorithm is divided into three modules. The first is the construction of the underlying network environment, including the simulation of network topology nodes and link resources. Next, the request generation module. Each request contains an SFC and each SFC contains 3 to 6 VNFs. Finally, the DRL algorithm module runs the A-DDPG algorithm. Once the agent is well-trained after convergence, it can make the right decision for the VNF placement problem.

We compare our A-DDPG method with three counterpart algorithms: DDPG, NFVdeep [31], and Q-learning.

**DDPG**: DDPG is a model-free DRL algorithm to solve the VNF placement problem. The difference with A-DDPG is that it does not add an attention mechanism. For each request $r(\xi, \mathcal{F}, D) \in \mathcal{R}$, the DDPG algorithm tries to place $f \in \mathcal{F}_r$ on node $n \in \mathcal{N}$ only considering its current remaining resource capacity, regardless of the neighbors' states. Moreover, it considers actor and critic networks with two fully connected layers, in which the number of nodes is 32 and 16, respectively. Meanwhile, we set $\alpha = 0.01$, the batch size is 64, and $\gamma = 0.8$, which is consistent with the parameter settings of A-DDPG.

**NFVdeep**: NFVdeep is a state-of-the-art method for VNF placement problems. NFVdeep methods are a type of RL technique that relies upon optimizing parameterized policies concerning the expected return (long-term cumulative reward) by gradient descent.

$$\bigtriangledown_\theta J(\theta) = \left( \frac{\partial \mathcal{J}(\theta)}{\partial \theta_1}, ..., \frac{\partial \mathcal{J}(\theta)}{\partial \theta_{\mathfrak{n}}} \right) \tag{18}$$

where the parameter $\theta$ is updated as:

$$\theta_{i+1} = \theta_i + \alpha \bigtriangledown_{\theta_i} \mathcal{J}(\theta_i) \tag{19}$$

where $\alpha$ is the learning rate and $\mathfrak{n}$ is the number of neurons. During the training process, the agent processes one VNF of SFC in each MDP state transition. Then the reward for each state $s$ is calculated, and the physical network gives the reward to the NFVdeep agent. Subsequently, the NFVdeep agent is trained for updating the policy circularly until the reward converges.

**Q-learning**: This is an model-free reinforcement learning technique that seeks to find an optimal action to take at a given state. Q-learning utilizes a table Q to represent the expected total reward of taking that action from a given state. The Q-value is updated as follows:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \tag{20}$$

where $s$ represents the state at a certain moment, $a_t$ indicates the action taken at that moment, $Q(s_t, a_t)$ denotes the Q-value corresponding to the (state, action) pair, $r$ reflects the reward function, $Q(s_{t+1}, a_{t+1})$ represents the state transition function, and $a_{t+1}$ denotes the action corresponding to the next state.

For the above three methods, we compare their network utility, delay, and running time. Among them, the network utility reflects the resource occupancy of the nodes and links of the network according to the Eq. (1), and the total delay is calculated by the sum of each path and node processing delay to reflect the network utility of VNF placement.

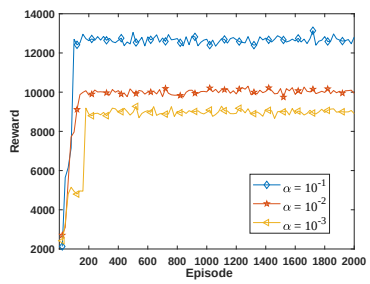
### 5.2 Simulation Results

Fig. 4(a) shows reward returned by A-DDPG under different learning rates ($0.1$, $0.01$ and $0.001$). The placement of all VNFs is completed in each training episode. It can be seen from Fig. 4(a) that the learning rate affects the value of reward in the algorithm's training progress. The reason is that the learning rate represents the amount that the weights are updated (a.k.a. the step size) during training. Smaller learning rates may lead to a slower weight update, so more training episodes are needed to achieve convergence of reward, whereas larger learning rates cause rapid changes and require fewer training episodes. According to our simulation results, when the learning rate is 0.01, A-DDPG achieves the best performance in terms of reward. Therefore, we will take the best learning rate for comparison with other algorithms. Its learning speed is acceptable, and it leads to faster convergence of reward function.

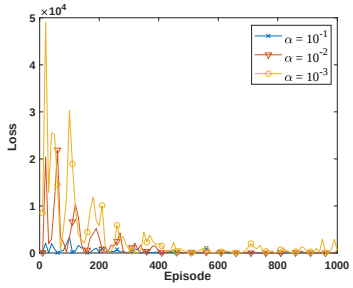
Fig. 4(b) shows the loss value of the A-DDPG method under different learning rates ($0.1$, $0.01$ and $0.001$). It can be seen from Fig. 4(b) that the learning rate affects the loss value in the algorithm's training step. The reason is that if the learning rate is too large, the loss function may directly exceed the global optimization of the learning process, whereas a small learning rate can cause the process to get stuck. When the learning rate is small, the changing speed of the loss function is slow. In this context, it will greatly increase the convergence complexity of the network, and the loss function is easy to be trapped in the local minima. Our simulation shows that the learning rate of $0.01$ provides the best performance for A-DDPG.

Figure 4(c) shows the loss of A-DDPG with different batch sizes, where the batch sizes are 32, 64, and 128, respectively. As can be seen from Fig. 4(c), as the episodes increase, the batch size will affect the value of the loss. We use batch gradient descent in the simulation to complete the iteration, which processes a portion of the samples at a time. A small sample will bring a large variance, which will cause the loss function to oscillate and slow down the convergence speed of the algorithm, especially when the network is complex. If the sample size is too large, the gradient estimation will be more accurate and stable, which may cause the neural network to converge to a poor local optimal solution point. Therefore, the batch size cannot be set too small or too large. According to our simulation results, the batch size can be set to $64$. In addition, we find that after a series of shocks, the loss value in Fig. 4(c) can always be stable near $0$ within a certain range. This indicates that our proposed algorithm can achieve convergence in VNF placement.

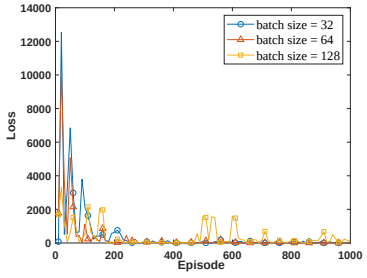
Fig. 4(d) shows the reward returned by all the algorithms. As the episodes increase, the value of the reward



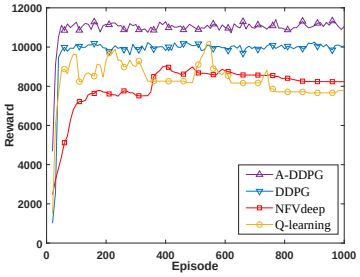

(a) The reward returned by A-DDPG under different learning rates.



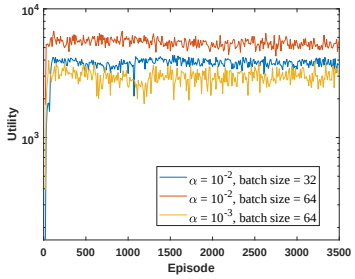

(b) The loss value of the A-DDPG under different learning rates.



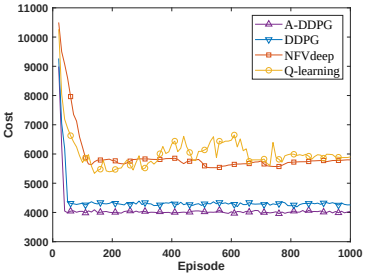

(c) The loss value of A-DDPG with different batch sizes.



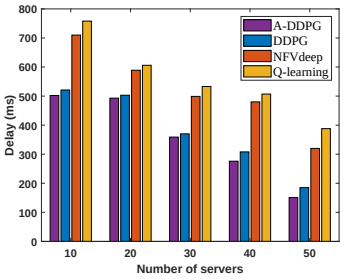

(d) The reward returned by all the algorithms.



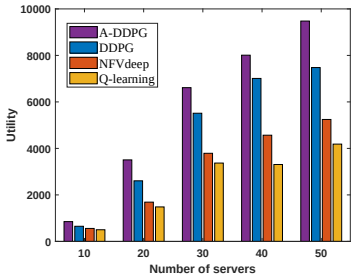

(e) The influence of learning rate and batch size on the utility for A-DDPG.



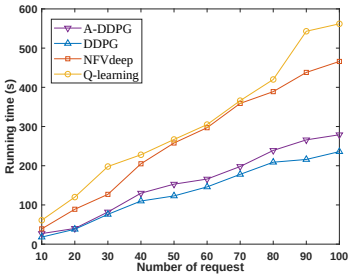

(f) The cost returned by all the algorithms.




(g) The influence of the number of servers on the delay for all the algorithms.




(h) The influence of the number of servers on the utility for all the algorithms.




(i) The influence of the number of servers on the running time for all the algorithms.

Fig. 4. Performance comparison of A-DDPG, DDPG, NFVdeep, and Q-learning.

gradually converges. In particular, we find that the A-DDPG algorithm is stable after being trained for 200 episodes. Subsequently, the reward value of A-DDPG somewhat fluctuates. On the one hand, this is due to the random generation of network requests, and the reward value is related to the completion of the network request. On the other hand, when poor samples are selected, one may end up in a local optimum, which results in a low reward value. It can be observed from Fig. 4(d) that Q-learning and NFVdeep always return the lowest reward because they do not directly use the deep network to select actions. DDPG performs better due to its capability to select actions directly. However, it is not as good as A-DDPG, since it fails to capture the neighbors' states. The A-DDPG algorithm can always achieve the highest reward after 200 episodes of training among the simulated algorithms. The reason is that the A-DDPG agent takes action by additionally paying attention to the states of neighbors, and the correct behavior enables the agent to obtain positive rewards faster during training, which accelerates the learning process. The results imply that the A-DDPG agent is more intelligent than other agents.

Fig. 4(e) depicts the influence of learning rate and batch size on the utility for A-DDPG. As the A-DDPG model iterates, the utility gradually converges towards a maximum where the model optimizes the weights. The utility value of $\alpha = 0.01$ is higher than that of $\alpha = 0.001$. We analyze that this is because the higher learning rate may miss the global optimization of the learning process, so it will cause the network to converge to a local optimum and obtain a low utility. Moreover, the speed of convergence when $batch\ size = 64$ is higher than the value when $batch\ size = 32$. This is because, as the batch size increases, the data processing speed becomes faster, which can reduce training time and enhance system stability.

Fig. 4(f) shows the cost returned by all the algorithms. As can be seen from Fig. 4(f), the training efficiency of Q-learning is lower than that of A-DDPG. More specifically, the cost of NFVdeep fluctuates at 5800 after 200 episodes. The cost of Q-learning fluctuates at 6000 after 1800 episodes. The cost of A-DDPG stabilizes after 100 episodes with slight fluctuations at 4000. Given these points, A-DDPG converges faster than NFVdeep and Q-learning. This is because, during initial training, VNFs are randomly placed

on the different servers, which incurs large operation and transmission costs. Through the training of Q-table and neural networks, Q-learning and A-DDPG agents can reduce unnecessary costs through training results. However, due to the use of neural networks, an A-DDPG agent is conducive to expressing complex network states. Compared with the discrete strategy in NFVdeep, A-DDPG can directly optimize the strategy (e.g., request rate) to meet the time-varying network states. Under the same conditions, it can process more network requests and reduce cost, thereby improving the processing capacity of the network. The result shows that an A-DDPG agent is more intelligent since it can achieve lower costs.

Fig. 4(g) shows the influence of the number of servers on the delay for all the algorithms. As shown in Fig. 4(g), the delay shows a downward trend as the number of servers increases. With the increase in the number of servers, it guarantees enough resources and applicable paths to accommodate requests. Our A-DDPG algorithm achieves a lower delay compared with the other three approaches. This is because, as the number of servers increases, the topology becomes more complicated. The placement of VNFs using NFVdeep and Q-learning can easily cause the server to fall into a local bottleneck due to performance degradation, whereas A-DDPG adds incentives for delay optimization in the reward function. The value of the reward increases more and more as delay decreases. Therefore, the node with the smaller delay will be selected to deploy the VNF in the strategic choice.

Fig. 4(h) shows the influence of the number of servers on the utility for all the algorithms. As shown in Fig. 4(h), A-DDPG achieves higher utility than the others. It reflects the superiority of A-DDPG in expressing decision-making in complex network environments. A-DDPG can obtain better utility under time-varying network states compared with the DDPG algorithm. NFVdeep performs well in an environment with $30$ servers. Q-learning is not sensitive to the number of servers. In short, the A-DDPG method can obtain higher utility by adopting an adaptive selection policy in a complex network environment.

Fig. 4(i) shows the running time performance of all the algorithms. As seen in Fig. 4(i), the running time of Q-learning is significantly higher than those of A-DDPG, DDPG, and NFVdeep. In addition, when there are more than $80$ requests, the running time of Q-learning increases rapidly. This is because when there are fewer requests, the servers provide a large aggregate capacity available on-demand, and there exist more feasible solutions for the VNF placement problem to achieve the best performance. However, with an increase in resources, the state space and action space in the Q-table greatly increase. In that case, finding the optimal strategy by looking up the table becomes difficult, and considering the complex calculations of nodes and links in large NFV networks, it takes more time to find the optimal strategy in large networks. Although A-DDPG and DDPG spend some more time to train the neural networks, after the training is completed and deployed, it only needs to use the well-trained neural networks for reasoning. A-DDPG has more running time than DDPG. This is because A-DDPG's neural network architecture has one more attention layer than DDPG, which causes a slight increase in running time. Therefore, A-DDPG consumes reasonable running times due to the powerful representation capabilities of the neural networks.

## 6 DRL-BASED SFC ROUTING

In the VNF placement problem, a complete graph is assumed, where the link delay is given in advance and the link bandwidth is not considered. However, the "link" between each node pair actually indicates a "path" in reality, which may traverse multiple nodes. In this section, we focus on solving the SFC routing problem to find a path from a source to a destination to maximize the number of accepted requests. The returned SFC routing solutions serve as the input for the VNF placement problem. Consequently, by leveraging DRL we first propose a Markov decision process including state, action, and reward. After that, we propose our Pointer network-based Actor-Critic (P-AC) algorithm to solve the SFC routing problem. Then, we discuss the training process for parameter updating according to the optimization strategy. Finally, we conduct simulations to compare P-AC with two existing DRL algorithms in terms of performance.

### 6.1 Problem Definition and Formulation

We formally define the SFC routing problem as follows:

***Definition* 2.** Given are a network $\mathcal{G} = \{\mathcal{N}, \mathcal{E}\}$ and a set of requests $R$. For each request $r_l(s_l, d_l, \xi, \mathcal{F}, D) \in \mathcal{R}$, the SFC routing problem is to find a path from a source node $s_l$ to a destination node $d_l$ at a flow rate $\xi$ in a specific order in $G$, such that the number of accepted request $\mathcal{O}$ is maximized.

The SFC routing problem is known to be NP-hard [40], which has been proved in [40] by reducing it to an NP-hard weight constrained shortest path problem (WCSPP). Due to the space limitation, we omit the proof of the NP-hardness of the SFC routing problem.

### 6.2 MDP model

In this subsection, we present the proposed DRL-based control model to solve the SFC routing problem mentioned above. The model is described as an MDP. We first design state space, action space, and reward in the MDP as follows:

**State**: The state space can be described as $S = \{S_{\mathbb{N}}, S_{\mathcal{E}}\}$, which includes node and edge states to accommodate requests. The node states consist of the source node, the destination node, the relay nodes, and the neighbors shortest path distances to different destinations. The edges states include end-to-end delay and links maximum available bandwidth to different destinations.

**Action**: An action is defined as an ordered pair $(u, v)$, where $u, v \in \mathcal{N}$. The action output for a coming flow request is also a vector which is the probability distribution of selecting neighbor nodes as the next stop.

**Reward**: Since we want to maximize the number of accepted requests, we define the reward function as to whether the request is accepted like the following:

$$R_r = \mathcal{O} - \beta_1 \max\left\{b_{i,k} - \overline{B_k}, 0\right\} - \beta_2 \max\left\{t_{q,k} - \overline{T_k}, 0\right\} \tag{21}$$

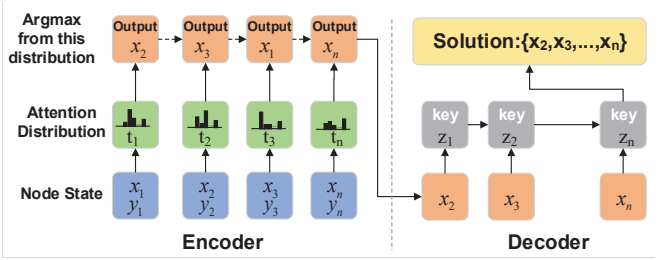



Fig. 5. Pointer Network Design of P-AC Framework.

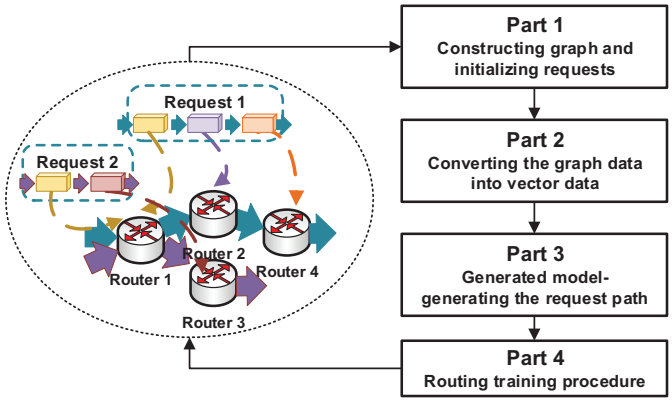



Fig. 6. The P-AC framework for the SFC routing problem.

where $b_{i,k}$ is the required bandwidth of $k$ at the node $i$, $t_{P_q,k}$ is the delay of $k$ at the link $q \in Q_k$, $\overline{B_k}$ is the bandwidth of current path of request $k$, and $\overline{T_k}$ is the delay of current path of request $k$. In the objective reward function in Eq. (21), successfully delivering a request can increase the reward value by one. Then $\beta_1 \max\{b_{i,k} - \overline{B_k}, 0\}$ increases the penalty value if requests are delivered later than respective deadlines. $\beta_2 \max\{t_{q,k} - \overline{T_k}, 0\}$ penalizes the total reward if the on-board logistic bandwidth exceeds the limit.

### 6.3 Routing Agent Design

Incorporating the above definitions, we start to design our P-AC framework.

#### 6.3.1 *Pointer Network Model*

To map state input to action output, we design a neural network as the policy function for all requests. The novel network structure uses attention as a pointer to select the node with the highest probability as the next-hop node for path selection at each time of selection. Since the length of the selected output path is uncertain, the number of static outputs of the existing neural network does not meet our needs. Inspired by [41], [42], we employ a pointer network [43] with structural graph embedding to generate the full path of the request.

Fig. 5 presents the architecture of the proposed P-AC neural network. We use the Struct2vec structure to generate the feature embedding of the network graph. The structure is mainly divided into two parts, namely the encoder and decoder. The input of the encoder network includes the state information of the nodes and the selected SFC routing path. The encoder of the pointer network model deploys LSTM cells with 128 hidden units. The decoder also uses LSTM cells with the same structure and then uses the *softmax* function on the output of the decoder to obtain the probability distribution of the next-hop node. Given a graph $\mathcal{G}$, Struct2vec first initializes a p-dimensional feature embedding $\mu_n = 0$ for each node $n \in \mathbb{N}$. The feature embedding $\mu_n$ is updated synchronously in each iteration as:

$$\mu_u^{r+1} \leftarrow f\left(x_u, \{\mu_v\}_{v\in\mathbb{N}}, \{l^o_{u,v}\}_{v\in\mathbb{N}_u}, \{b^o_{u,v}\}_{v\in\mathbb{N}_u}, \{t^o_{u,v}\}_{v\in\mathbb{N}_u}; \Upsilon\right) \tag{22}$$

where $\mathbb{N}$ is the set of neighbors of node in graph $G$, $x_u$ is a h-dimensional node feature of $u$. For example, if the node is a source node, its representation should contain two main types of information: (1) node information, including the label and processing delay, and (2) node's neighbor information, including the distance, bandwidth and traversing delay to all its neighbors. If the node type is a destination node, its representation should contain the node label and the deadline. The characteristics of the relay nodes include the requested information, the distance, traversing dela, and bandwidth from the source node to the destination node. We denote $l^o_{u,v}, b^o_{u,v}, t^o_{u,v}$ as the distance, bandwidth, and traversing delay for $o$ to route from $u$ to $v$, respectively. $\Upsilon$ is the nonlinear parameters of the pointer model. $F$ is a generic nonlinear mapping such as a neural network or kernel function by $\Upsilon$. Then, the nonlinear propagation function is defined as:

$$\begin{aligned}\mu_u^{r+1} \leftarrow Relu(\theta_1 \sum_{v\in N_u} \mu_v^r + \theta_2 Relu(\theta_3 \sum_{v\in N_u} l^o_{u,v}) + \theta_4 \\ Relu(\theta_5 \sum_{v\in N_u} b^o_{u,v}) + \theta_6 Relu(\theta_7 \sum_{v\in N_u} t^o_{u,v}) + \theta_8 x_u)\end{aligned} \tag{23}$$

where $\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6, \theta_7, \theta_8 \in \Upsilon$ are the model parameters. Then the embedded representation in Eq. (23) is used as the input of the pointer network.

To cope with the uncertain length of the output of the SFC routing problem, we adopt a novel pointer network as the actor network. The input of the pointer network is a vector of nodes and link states, and the output is also a set of vectors of uncertain length indicating the optimal path selection strategy. The pointer network uses attention as a pointer to select the node with the highest probability as the next-hop node for path selection at each selection. The parameters of the model are learned by maximizing the conditional probabilities for the training set, which can be defined as:

$$\theta^* = \arg\max \sum_{\bar{N}} \log p(S^N|\bar{N}) \tag{24}$$

where the $\bar{N} = \{\mathbb{N}_1, ..., \mathbb{N}_n\}$ is a sequence of $|n|$ vectors and $S^N$ is the output sequence of pointer network. We use LSTM to model the probability $p(S^N|\bar{N})$ of the pointer network. We use two independent LSTMs: a compiler for encoding the $\bar{N}$ sequence and a decoder for generating or decoding the output $S^N$. In this network, a sequence $\bar{N}$ is

used as the input to each iteration, and then the sequence with the highest probability $S^N$ is selected as the output of the decoder using the learned parameter $\theta^*$. Consequently, a sequence of paths with optimal probability is obtained.

#### 6.3.2 Algorithm Design

We now describe the P-AC framework for the SFC routing problem. The whole P-AC framework mainly consists of four parts: Part 1, Part 2, Part 3, and Part 4, as shown in Fig. 6.

Part 1 is to construct the non-fully connected graph with $N$ nodes and $E$ links, and then we initialize the source node and destination node of each request. All other nodes can transmit request $\mathcal{R}$ as connections. Therefore, we can enumerate all possible connected links in the graph. Then, using the Dijkstra algorithm, we find the shortest path of each edge $e \in \mathcal{E}$ in the network $\mathcal{G}$. Finally, the distance, bandwidth, and traversing delay of the requested path are recorded as the three weight attributes of the edge, denoted as $\{l^o_{u,v}\}_{v\in\mathbb{N}_u}, \{b^o_{u,v}\}_{v\in\mathbb{N}_u}, \{t^o_{u,v}\}_{v\in\mathbb{N}_u}$.

Part 2 converts the graph data into vector data. Step 2 can be divided into two steps, namely (1) generating a node topology map according to the parameters of step 1, and (2) converting the node topology map into a vector structure. According to the topology generated in the part 1, we know that some nodes are not directly connected. We find the relay nodes for the edges that are not directly connected and find the shortest distance, replacing the infinity value in the updated node topology matrix. Then we construct random paths for all requests by randomly generating a random order of $|train\_size|$ nodes, and then put the node with the property start to the first, so that we generate $|train\_size|$ requested paths with different lengths, and then the graph data and the requested paths are transformed into vectors. This is because the training input of the model is generally a vector structure. The purpose of step 2 is to facilitate the subsequent input of data into the model for training.

Part 3 is a generated model. The main goal of the model is to generate the requested path with the highest number of total request completions. The input of part 3 is the network state and the request path graph of part 2.Then, we design an Actor-Critic network structure based on a pointer network with structural graph embedding to generate the path of each request step by step. When generating a path, each step looks for the element with the largest weight in the current input sequence, and the weight judgment is obtained by the attention distribution as shown in Fig. 5.

Part 4 is to put vector data into model training. More specifically, the pointer network is performed in the routing training procedure. In this section, we train the model parameters by gradient function, which can be calculated by:

$$\bigtriangledown J\left(\Upsilon|S\right) \approx \frac{1}{B}\sum_{B}^{i=1}\Big(R - b\left(S_i\right) \times \bigtriangledown_{\Upsilon} log_{p_{\Upsilon}}\left(\pi_i|S_i\right)\Big) \quad (25)$$

where $B$ is the number of samples $\{S_1, S_2, ..., S_B\}$. It can be observed that given a well-formed $b(S_i)$ function [41], the gradient can be easily computed. In particular, Algorithm 2 in Part 4 is trained to find a path for each request by reinforcement learning of the SFC routing agent, and at each time slot, we can process multiple requests one by one. In the routing training procedure, the model parameters are updated iteratively. In each iteration, new graphs (constructed by network states as described in Part 1) are firstly sampled from $B$ (line 4), whose paths are then developed using pointer networks (line 5). The estimated reward value of these states is also generated using a critic network at the same time (line 6). Subsequently, the agent stores the quadruple $(s_t, a_t, r_t, s_{t+1})$ into the experience replay $B$ (Line 7). The next step is to update the pointer and critic network. Then the gradient of pointer network is calculated using Eq. (25), and the mean squared error objective of critic network is computed by $L = \frac{1}{B}\sum_t(O - Q(s_t, a_t|\theta^Q)^2)$ (line 8). Lastly, the model parameters are updated using the Adam optimizer [44] (line 9) with a mini-batch size. This finishes one iteration of the training algorithm. The algorithm terminates when the reward converges towards the optimal policy, or a pre-defined maximum number of iterations is reached. We set the maximum number of rounds to be 5000 based on simulation experience.

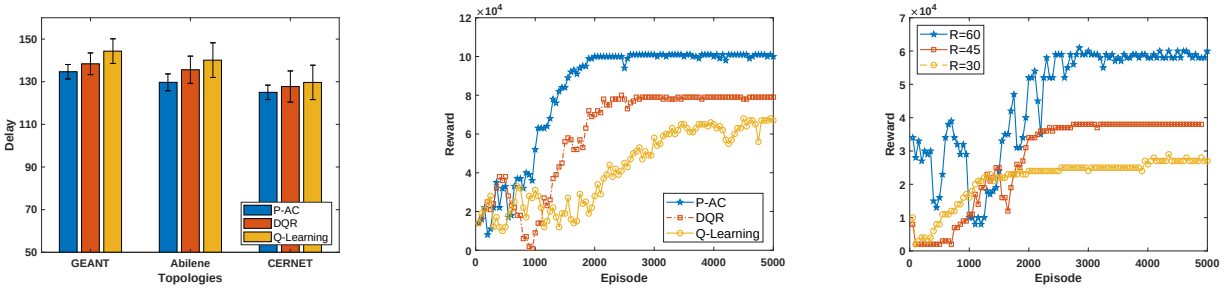
**Algorithm 2** Routing Training Procedure

1: Randomly initialize pointer and critic network with weights $\Upsilon^P$ and $\Upsilon^C$
2: Initialize sample_batch, training episode $M$, replay buffer $B$
3: **for** $episode = 0, 1, ..., M$ **do**
4: Take a sample of $S_i$ from batch size
5: Select action $a_t = \mu(s_t|\theta^\mu) + \mathbb{N}_t$ according to the current policy
6: Execute action $a_t$ and observe reward $r_t$ and observe new state $s_{t+1}$
7: Store a random minibatch of $N$ transitions $(s_t, a_t, r_t, s_{t+1})$ from $\mathbb{B}$
8: Update critic by minimizing the loss: $L = \frac{1}{B}\sum_t(O - Q(s_t, a_t|\theta^Q)^2)$
9: Update the actor policy using the sampled gradient: $\bigtriangledown J\left(\Upsilon|S\right) \approx \frac{1}{B}\sum_{B}^{i=1}\Big(R - b\left(S_i\right) \times \bigtriangledown_{\Upsilon} log_{p_{\Upsilon}}\left(\pi_i|S_i\right)\Big)$
10: **end for**

### 6.4 Performance evaluation

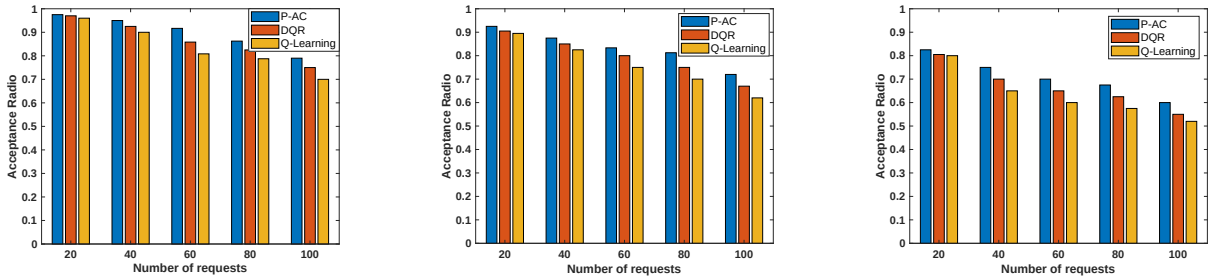
#### 6.4.1 Simulations Setting

We conduct simulations on three network topologies, namely, CERNET [45], GÉANT [46] and Abilene [47]. The topology parameters are shown in Table 3. CERNET is the first educational network funded by the Chinese government and managed directly by the Chinese Ministry of Education. GÉANT is the pan-European data network for the research and education community. Abilene is the leading academic network created by the Internet2 community, connecting more than 200 universities in the United States. For each link in the network, its capacity is randomly assigned from the range $[2, 4]$ $Gb/s$ and its delay takes value in $[10, 50]$ $ms$, which is in accordance with the real GÉANT topology information [48]. We keep the parameters of both Abilene and CERNET network topologies the same as GÉANT to facilitate the comparison of the algorithms' performance. Additionally, all requests in the network are




(a) Path delay in three topologies for 100 requests. (b) Reward by all the algorithms for 100 requests. (c) Reward of the P-AC algorithm under different number of requests.

Fig. 7. Performance comparison of P-AC, DQR, and Q-learning.




(a) CERNET. (b) Abilene. (c) GÉANT.

Fig. 8. Acceptance Ratio (AR) of three algorithms in three topologies: (a) CERNET (b) Abilene (c) GÉANT.

randomly generated. Each request has a random flow rate between 1 Mbps and 5 Mbps, which is in accordance with [49].

TABLE 3
The Parameters for CERNET, GÉANT, and Abilene Topologies.

| Network Topologies | Nodes | Links |
|---|---|---|
| CERNET | 39 | 106 |
| GÉANT | 32 | 82 |
| Abilene | 39 | 89 |

We compare our P-AC algorithm with two counterpart algorithms: DQR [50] and Q-learning.

**DQR**: DQR is a DQN-based greedy online SFC routing method. The difference with P-AC is that it does not add a Critic network. For each request $r(\xi, \mathcal{F}, D) \in \mathcal{R}$, the DQR algorithm tries to find the path on $\mathcal{G}$ only considering its current remaining link bandwidth, regardless of the neighbors' states. Moreover, it considers current and target Q networks with LSTMs. Meanwhile, we set $\alpha = 10^{-2}$, the batch size to be $64$ and $\gamma = 0.8$ in DQR, which are consistent with the parameter settings of P-AC.

**Q-learning**: Q-Learning is a model-free reinforcement learning technique that makes decisions. For each request $r(\xi, \mathcal{F}, D) \in \mathcal{R}$, the Q-learning algorithm tries to find the path from source to destination in $\mathcal{G}$, such that the number accepted requests is maximized.

For the above two methods, we compare their returned delay value and reward value, which is reflected by the number of requests in Eq. (21).

### 6.4.2 *Simulation Results*

Fig. 7(a) shows the path delay performance returned by all the algorithms in three topologies for 100 requests. As shown in Fig. 7(a), the achieved delay performance of all the algorithms behaves in a slightly downward trend as the topologies' size increases. For a fixed number of 100 requests, P-AC achieves a lower delay performance than the other two algorithms. However, the total delay result returned by all these three algorithms does not show a big difference with increasing link connectivity. This is because the paths learned by these three algorithms are roughly the same distance, so the length of the paths do not differ significantly. As a result, there is no big difference in delay among these three algorithms. However, there is a difference in the AR performance for these three algorithms in the network as shown in Fig. 8(a), 8(b) and 8(c).

Fig. 7(b) depicts the reward values returned by different algorithms for 100 requests. It can be seen from the figure that with the episodes increasing, the reward values of the P-AC algorithm converge to a large value. In addition, we can see that P-AC algorithm converges much earlier when it is trained for 3800 episodes, while DQR and Q-Learning algorithms converge at 5000 and 9000 episodes, respectively. This is because P-AC algorithm employs experience replay compared to the other two methods. In summary, the results verify the advantages of the P-AC algorithm in terms of reward and convergence speed compared with other baselines.

Fig. 7(c) shows the reward of the P-AC algorithm for different number of requests (30, 45 and 60). It can be seen

from Fig. 7(c) that the number of requests affects the reward in the algorithm's training process. On the one hand, this is because more requests can be accommodated as the number of requests increases, and hence the value of the reward increases accordingly. On the other hand, when the number of requests increases, the SFC routing agent needs more episodes to learn the appropriate routing policy to route the requests.

Figs. 8(a), 8(b) and 8(c) depict the Acceptance Ratio (AR) of the three algorithms for different number of requests. AR is defined by the number of accepted requests divided by the total number of requests. When the number of requests increases, the achieved AR values of these three algorithms decrease. This is because as the number of accepted requests increases, the whole network becomes more congested. Therefore the remaining available link bandwidth becomes smaller, and then the request acceptance rate decreases subsequently. Nevertheless, we see that the P-AC algorithm can always achieve the highest AR value since P-AC introduces an attention-based pointer network, which verifies its superiority.

## 7 Conclusion

In this paper, we have first studied the VNF placement problem, which is to place the requested VNFs for each request such that the network utility is maximized. To solve it, we present a DRL framework with an attention mechanism, called A-DDPG, which consists of three processes: observation process, training process, and online running process. By introducing an attention mechanism, we can focus on more critical information, such as the state of neighbors, to reduce the attention to other unnecessary nodes and improve the training efficiency of the model. Subsequently, we propose an Actor-Critic-based learning algorithm, called P-AC, for the SFC routing problem. P-AC adopts a novel design of a pointer network that enables decisions on SFC routing. The pointer network of P-AC uses attention as a "pointer" to select the node with the highest probability as the next-hop node for path selection at each time. Via simulations, we find that our proposed algorithms can outperform the state-of-the-art in terms of network utility, delay, cost, and AR. In our future work, we plan to implement the proposed algorithms to verify their efficiency and scalability in real-world experimental network scenarios.

## Acknowledgements

The authors would like to acknowledge and thank Prof. Fernando A. Kuipers for his effort and contribution in the preliminary version of this work. The work of Song Yang is partially supported by the National Natural Science Foundation of China (NSFC, No. 62172038). The work of Fan Li is partially supported by the NSFC under Grant No. 62072040. The work of Liehuang Zhu is partially supported by NSFC under Grant No. 62232002. The work of Xiaoming Fu is partially supported by EU H2020 COSAFE project (Contract No. 824019) and EU Horizon CODECO project (Contract No. 101092696). Song Yang is the corresponding author.

## References

[1] N. He, S. Yang, F. Li, S. Trajanovski, F. A. Kuipers, and X. Fu, "A-DDPG: Attention mechanism-based deep reinforcement learning for NFV," in *IEEE IWQoS*, 2021, pp. 1–10.

[2] H. Ren, Z. Xu, W. Liang, Q. Xia, P. Zhou, O. F. Rana, A. Galis, and G. Wu, "Efficient algorithms for delay-aware NFV-enabled multicasting in mobile edge clouds with resource sharing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 9, pp. 2050–2066, 2020.

[3] M. Scazzariello, L. Ariemma, G. Di Battista, and M. Patrignani, "Megalos: A scalable architecture for the virtualization of network scenarios," in *IEEE NOMS*, 2020, pp. 1–7.

[4] S. Yang, F. Li, S. Trajanovski, R. Yahyapour, and X. Fu, "Recent advances of resource allocation in network function virtualization," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 2, pp. 295–314, 2021.

[5] X. Fu and E. Modiano, "Learning-NUM: Network utility maximization with unknown utility functions and queueing delay," in *Proceedings of the Twenty-second International Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing*, 2021, pp. 21–30.

[6] D. T. Nguyen, C. Pham, K. K. Nguyen, and M. Cheriet, "Placement and chaining for run-time IoT service deployment in edge-cloud," *IEEE Transactions on Network and Service Management*, vol. 17, no. 1, pp. 459–472, 2019.

[7] J. Pei, P. Hong, K. Xue, and D. Li, "Efficiently embedding service function chains with dynamic virtual network function placement in geo-distributed cloud system," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 10, pp. 2179–2192, 2018.

[8] Q. Zhang, Y. Xiao, F. Liu, J. C. Lui, J. Guo, and T. Wang, "Joint optimization of chain placement and request scheduling for network function virtualization," in *IEEE ICDCS*, 2017, pp. 731–741.

[9] M. Golkarifard, C. F. Chiasserini, F. Malandrino, and A. Movaghar, "Dynamic VNF placement, resource allocation and traffic routing in 5G," *Elsevier Computer Networks*, vol. 188, p. 107830, 2021.

[10] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," in *AAAI Conference on Artificial Intelligence*, 2018.

[11] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[12] R. Z. Courville, Ruslan Salakhudinov and Y. Bengio, "Show, attend and tell: Neural image caption generation with visual attention," *International Conference on Machine Learning*, 2015.

[13] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[14] T. Gao, X. Li, Y. Wu, W. Zou, S. Huang, M. Tornatore, and B. Mukherjee, "Cost-efficient VNF placement and scheduling in public cloud networks," *IEEE Transactions on Communications*, 2020.

[15] 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-SAC*, vol. 38, no. 2, pp. 263–278, 2019.

[16] L. Gu, D. Zeng, W. Li, S. Guo, A. Y. Zomaya, and H. Jin, "Intelligent VNF orchestration and flow scheduling via model-assisted deep reinforcement learning," *IEEE J-SAC*, vol. 38, no. 2, pp. 279–291, 2019.

[17] D. Li, P. Hong, K. Xue *et al.*, "Virtual network function placement considering resource optimization and SFC requests in cloud datacenter," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 7, pp. 1664–1677, 2018.

[18] D. Bhamare, R. Jain, M. Samaka, G. Vaszkun, and A. Erbad, "Multi-cloud distribution of virtual functions and dynamic service deployment: Open ADN perspective," in *2015 IEEE International Conference on Cloud Engineering*, 2015, pp. 299–304.

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

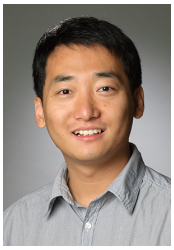
[20] H. Feng, J. Llorca, A. M. Tulino, D. Raz, and A. F. Molisch, "Approximation algorithms for the NFV service distribution problem," in *IEEE INFOCOM*, 2017, pp. 1–9.

[21] P. Hong, K. Xue, D. Li *et al.*, "Resource aware routing for service function chains in SDN and NFV-enabled network," *IEEE Transactions on Services Computing*, 2018.

[22] W. Wang, H. Xu, G. Zhao, and L. Huang, "Real-time and consistent route update based on segment routing for NFV-enabled networks," in *International Conference on Wireless Algorithms, Systems, and Applications*. Springer, 2021, pp. 335–344.
[23] W. Ma, O. Sandoval, J. Beltran, D. Pan, and N. Pissinou, "Traffic aware placement of interdependent NFV middleboxes," in *IEEE INFOCOM*, 2017, pp. 1–9.
[24] H. Tang, D. Zhou, and D. Chen, "Dynamic network function instance scaling based on traffic forecasting and VNF placement in operator data centers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 3, pp. 530–543, 2018.
[25] S. Yang, F. Li, S. Trajanovski, X. Chen, Y. Wang, and X. Fu, "Delay-aware virtual network function placement and routing in edge clouds," *IEEE Transactions on Mobile Computing*, 2019.
[26] M. Nakanoya, Y. Sato, and H. Shimonishi, "Environment-adaptive sizing and placement of nfv service chains with accelerated reinforcement learning," in *IEEE IM*, 2019, pp. 36–44.
[27] P. Sun, J. Lan, J. Li, Z. Guo, and Y. Hu, "Combining deep reinforcement learning with graph neural networks for optimal vnf placement," *IEEE Communications Letters*, 2020.
[28] 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 Journal on Selected Areas in Communications*, 2021.
[29] J. Pei, P. Hong, K. Xue, D. Li, D. S. Wei, and F. Wu, "Two-phase virtual network function selection and chaining algorithm based on deep learning in SDN/NFV-enabled networks," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 6, pp. 1102–1117, 2020.
[30] 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 *IEEE ICDCS*, 2019, pp. 934–943.
[31] Y. Xiao, Q. Zhang, F. Liu, J. Wang, M. Zhao, Z. Zhang, and J. Zhang, "NFVdeep: Adaptive online service function chain deployment with deep reinforcement learning," in *Proceedings of the International Symposium on Quality of Service*, 2019, pp. 1–10.
[32] P. T. A. Quang, A. Bradai, K. D. Singh, and Y. Hadjadj-Aoul, "Multi-domain non-cooperative VNF-FG embedding: A deep reinforcement learning approach," in *IEEE INFOCOM WKSHPS*, 2019, pp. 886–891.
[33] R. Tong, S. Xu, B. Hu, J. Zhao, L. Jin, S. Guo, and W. Li, "VNF dynamic scaling and deployment algorithm based on traffic prediction," in *IEEE IWCMC*, 2020, pp. 789–794.
[34] C. E. Shannon, "A mathematical theory of communication," *ACM SIGMOBILE*, vol. 5, no. 1, pp. 3–55, 2001.
[35] J. S. Chase, R. P. Doyle, and S. D. Ims, "Multi-tier service level agreement method and system," 2006.
[36] S. Verbrugge, D. Colle, M. Pickavet, P. Demeester, S. Pasqualini, A. Iselt, A. Kirstadter, R. Hu lsermann, F.-J. Westphal, and M. Jager, "Methodology and input availability parameters for calculating opex and capex costs for realistic network scenarios," *Journal of Optical Networking*, vol. 5, no. 6, pp. 509–520, 2006.
[37] T. Li, C. S. Magurawalage, K. Wang, K. Xu, K. Yang, and H. Wang, "On efficient offloading control in cloud radio access network with mobile edge computing," in *IEEE ICDCS*, 2017, pp. 2258–2263.
[38] K. Gardner, S. Borst, and M. Harchol-Balter, "Optimal scheduling for jobs with progressive deadlines," in *IEEE INFOCOM*, 2015, pp. 1113–1121.
[39] M. Savi, M. Tornatore, and G. Verticale, "Impact of processing costs on service chain placement in network functions virtualization," in *IEEE NFV-SDN*, 2015, pp. 191–197.
[40] I. Dumitrescu and N. Boland, "Algorithms for the weight constrained shortest path problem," *International Transactions in Operational Research*, vol. 8, no. 1, pp. 15–29, 2001.
[41] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, "Neural combinatorial optimization with reinforcement learning," *arXiv preprint arXiv:1611.09940*, 2016.
[42] H. Dai, E. B. Khalil, Y. Zhang, B. Dilkina, and L. Song, "Learning combinatorial optimization algorithms over graphs," *arXiv preprint arXiv:1704.01665*, 2017.
[43] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," *arXiv preprint arXiv:1506.03134*, 2015.
[44] Z. Zhang, "Improved adam optimizer for deep neural networks," in *2018 IWQoS*, 2018, pp. 1–2.
[45] China Educ. Res. Netw. (GENET), "CERNET topology," https://www.edu.cn/cernet_fu_wu/internet_n2/, Accessed November, 2021.
[46] S. Uhlig, B. Quoitin, J. Lepropre, and S. Balon, "Providing public intradomain traffic matrices to the research community," *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 1, pp. 83–86, 2006.
[47] Abilene, "Abilene topology," http://totem.run.montefiore.ulg.ac.be/datatools.html, Accessed November, 2021.
[48] GÉNET, "GÉNET topology," https://geant3plus.archive.geant.net/Pages/Resources/Maps.html.
[49] Z. Meng, H. Xu, M. Chen, Y. Xu, Y. Zhao, and C. Qiao, "Learning-driven decentralized machine learning in resource-constrained wireless edge computing," in *IEEE INFOCOM*, 2021, pp. 1–10.
[50] S. Q. Jalil, M. H. Rehmani, and S. Chalup, "DQR: Deep Q-Routing in software defined networks," in *IEEE International Joint Conference on Neural Networks (IJCNN)*, 2020, pp. 1–8.

## 8 Biography

**Nan He** received her B.E. degree in Computer and Information from Hefei University of Technology, China in 2016 and received her M.S. degree from Jilin University, China in 2019. She is currently a Ph.D. student in the School of Computer Science, Beijing Institute of Technology. Her research interests include network function virtualization and network resource optimization management based on reinforcement learning.



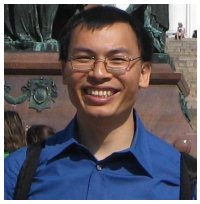
**Song Yang** received the B.S. degree in software engineering and the M.S. degree in computer science from Dalian University of Technology, Dalian, Liaoning, China, in 2008 and 2010, respectively, and the Ph.D. degree from Delft University of Technology, The Netherlands, in 2015. From August 2015 to August 2017, he worked as postdoc researcher for the EU FP7 Marie Curie Actions CleanSky Project in Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen (GWDG), Göttingen, Germany. He is currently an associate professor at School of Computer Science in Beijing Institute of Technology, China. His research interests focus data communication networks, cloud/edge computing and network function virtualization.

**Fan Li** received the BEng and MEng degrees in communications and information system from the Huazhong University of Science and Technology, Wuhan, China, in 1998 and 2001, respectively, the MEng degree in electrical engineering from the University of Delaware, Newark, Delaware, in 2004, and the PhD degree in computer science from the University of North Carolina at Charlotte, Charlotte, North Carolina, in 2008. She is currently a professor with the School of Computer Science, Beijing Institute of Technology, China. Her current research focuses on wireless networks, ad hoc and sensor networks, and mobile computing. Her papers won best paper awards from IEEE MASS (2013), IEEE IPCCC (2013), ACM MobiHoc (2014), and Tsinghua Science and Technology (2015). She is a member of IEEE and ACM.

**Stojan Trajanovski** received the master's (with distinction) degree in advanced computer science from the University of Cambridge, Cambridge, United Kingdom, in 2011, and the PhD (cum laude) degree from the Delft University of Technology, Delft, The Netherlands, in 2014. He is currently an applied scientist with Microsoft, working in London, United Kingdom and Bellevue, Washington. He was in a similar role with Philips Research in Eindhoven, The Netherlands from 2016 to 2019. Before that, he spent some time as a postdoctoral researcher with the University of Amsterdam and with the Delft University of Technology. He successfully participated at international science olympiads, winning a bronze medal at the International Mathematical Olympiad (IMO), in 2003. His main research interests include network science and complex networks, machine learning, game theory, and optimization algorithms.

**Liehuang Zhu** received the BE and ME degrees from Wuhan University, Wuhan, China, in 2001 and 1998, respectively, and the PhD degree in computer science from the Beijing Institute of Technology, Beijing, China, in 2004. He is currently a professor with the School of Cyberspace Security, Beijing Institute of Technology, Beijing, China. He has published more than 150 peer-reviewed journal or conference papers. He has been granted a number of IEEE best paper awards, including IWQoS 17', TrustCom 18', and ICA3PP 20. His research interests include security protocol analysis and design, blockchain, wireless sensor networks, and cloud computing.

**Yu Wang** is currently a Professor in the Department of Computer and Information Sciences at Temple University. He holds a Ph.D. from Illinois Institute of Technology, an MEng and a BEng from Tsinghua University, all in Computer Science. His research interest includes wireless networks, smart sensing, and mobile computing. He has published over 200 papers in peer reviewed journals and conferences. He has served as general chair, program chair, program committee member, etc. for many international conferences (such as IEEE IPCCC, ACM MobiHoc, IEEE INFOCOM, IEEE GLOBECOM, IEEE ICC), and has served as Editorial Board Member for several international journals (including IEEE Transactions on Parallel and Distributed Systems and IEEE Transactions on Cloud Computing). He is a recipient of Ralph E. Powe Junior Faculty Enhancement Awards from Oak Ridge Associated Universities (2006), Outstanding Faculty Research Award from College of Computing and Informatics at the University of North Carolina at Charlotte (2008), Fellow of IEEE (2018), and ACM Distinguished Member (2020).



**Xiaoming Fu** received his Ph.D. in computer science from Tsinghua University, Beijing, China in 2000. He was then a research staff at the Technical University Berlin until joining the University of Göttingen, Germany in 2002, where he has been a professor in computer science and heading the Computer Networks Group since 2007. He has spent research visits at universities of Cambridge, Uppsala, UPMC, Columbia, UCLA, Tsinghua, Nanjing, Fudan, and PolyU of Hong Kong. Prof. Fu's research interests include network architectures, protocols, and applications. He is currently an editorial board member of IEEE Network, IEEE Transactions on Network and Service Management, IEEE Transactions on Network Science and Engineering, IEEE Networking Letters, and Elsevier Computer Communications, and has served on the organization or program committees of leading conferences such as INFOCOM, ICNP, ICDCS, MOBICOM, MOBIHOC, CoNEXT, ICN and Networking. He is an IEEE Fellow, an IEEE Communications Society Distinguished Lecturer, an ACM Distinguished Member, a fellow of IET and member of the Academia Europaea.