A Hybrid Learning Framework for Service Function Chaining Across Geo-Distributed Data Centers

TAO TANG1, BINWEI WU2, AND GUANGMIN HU3

1School of Information and Communication Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China
2National Key Laboratory of Science and Technology on Communications, University of Electronic Science and Technology of China, Chengdu 611731, China
3School of Resources and Environment, University of Electronic Science and Technology of China, Chengdu 611731, China

Corresponding author: Guangmin Hu (hgm@uestc.edu.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant 0561701074.

ABSTRACT Service function chaining (SFC) focuses mainly on deploying various network functions in geographically distributed data centers and providing interconnect routing among them. Traditional (convex optimization-based) SFC algorithms exhibit some disadvantages on the scalability and accuracy. Recently, researches have shown the effectiveness of deep reinforcement learning (DRL) in the field of SFC. However, current DRL-based algorithms possess an extremely large action space, which leads to slow convergence and poor scalability. Some researchers relieve this issue by reformulating the SFC problem, which usually results in low utilization and high cost. To address this issue, we develop a hybrid DRL-based framework which decouples the VNF deployment and flow routing into different modules. In the proposed framework, a DRL agent is only responsible for learning the policy of VNF deployment. We customize the structure of the agent base on deep deterministic policy gradient (DDPG) and adopt several techniques to improve the learning efficiency, such as adaptive parameter noise, wolpertinger policy, and prioritized experience replay. The flow routing is conducted in a game-based module (GBM). We design a decentralized routing algorithm for the GBM to address the scalability. The end-to-end latency of flows is minimized while the resource capacity and location constraints are satisfied. During the learning process of the proposed framework, the DRL agent improves its deployment policy with the reward from the GBM (the value of reward depends on flow routing). Thus, the VNF deployment and flow routing are still jointly optimized. Compared to existing DRL-learning algorithms, the proposed hybrid DRL framework can achieve a lower cost since 1) the action space is significantly reduced due to flow routing decoupling; 2) the flow routing procedure is more efficient (the GBM adopts model-based information, e.g., the gradient). Through trace-driven simulations, we show the efficiency of our algorithm compared to existing DRL-based algorithms.

INDEX TERMS Service function chaining, deep reinforcement learning, game theory, cloud computing.

I. INTRODUCTION Traffic flows in packet networks may need to pass through a series of middleboxes (network functions) to acquire various network services. To accommodate the network changes (e.g., flow rate variations), network resources should be adaptively scheduled so that ubiquitous services can be provided. From the perspective of the network operators, it is truly painful to schedule resources on a hardware level. Thus, the service function chaining (SFC) technique is proposed, which utilizes network function virtualization (NFV) and software-defined networking (SDN) paradigms to improve network flexibility [1]. Hardware-based middleboxes are 'softwared' into virtualized network functions (VNFs) which can run on common servers in edges clouds or data centers. Meanwhile, the network operators can schedule the network traffic on flow-level granularity accordingly, which decreases the network cost and ensures a quality user experience.

The goal of SFC is to achieve significant cost reduction compared to traditional service provisioning with dedicated hardware [2]. On-demand deployment and scaling of VNF instances play key roles in cost conservation. As flow demands vary, network operators adaptively scale the number and the placement of VNF instances to save the cost.
(e.g., rental cost) according to the distribution of flows. Meanwhile, flow routing has significant impacts on quality-of-service (QoS), e.g., end-to-end delay, as declared in the service level agreement (SLA). Therefore, a SFC algorithm needs to jointly coordinate the placement of VNF instances and the routing of user flows. Meanwhile, the resource capacity and location constraints need to be satisfied during the scheduling. It has been proved that SFC is a mixed integer programming (MIP) problem with high-complexity, which is hard to solve using traditional optimization algorithms [3].

Recently, deep reinforcement learning (DRL) shows its great potential in related areas, such as navigation, traffic engineering, and flow control. Compared to traditional optimization algorithms, DRL has advantages in time-varying systems with complicated environment states. The agent of DRL can learn an approximately optimal policy in a short time through trial-and-error interactions. The policy determines the action to be taken when the perceived states of the environment are given. The authors of [4]–[6] have proven the efficiency of DRL in the field of SFC.

However, existing DRL-based SFC algorithms face several challenges. First, some algorithms do not take into consideration the variations of user demands. The DRL model needs to be re-trained once the flow patterns have changed. Second, the existing DRL-based algorithms have a large action space since the optimization of SFC involves both VNF deployment and flow routing [7]. A large action space would lead to a slow learning rate and inaccurate results. Third, some ideal assumptions are proposed to shrink the action space, meanwhile, led to performance degradation and higher cost.

This paper addresses the above-mentioned issues by proposing an innovative hybrid DRL-based algorithm for SFC deployment. Unlike existing learning-based algorithms, we decouple the flow routing from the DRL agent into a game-based module (GBM). The DRL agent only focuses on the placement of VNF instances so that the action space is significantly reduced. During the learning procedure of the DRL agent, the GBM is responsible for (1) generating a (sub-)optimal routing, (2) feedback the reward (reflect the quality of the flow routing given the current deployment policy). According to the reward (which covers the assessment on both deployment and routing), the DRL agent updates the parameterized policy to improve the outcome (the placement of VNF instances). To sum up, a joint optimization on both VNF deployment and flow routing can be carried out with the coordination between the DRL agent and the GBM. Note that, the optimization of flow routing in the GBM is conducted with a model-based algorithm. Model-related information, such as the gradient, is used to improve efficiency. Therefore, the proposed algorithm achieves a lower cost compared to a pure DRL-based algorithm.

1) We formulate the SFC problem in a geo-distributed environment. The VNF deployment and flow routing are jointly optimized to minimize the overall cost. A hybrid DRL-based framework is proposed to conduct optimization.

2) We propose a DRL agent to learn the optimal policy for the VNF deployment. The NN (neural network) structure of the DRL agent is redesigned to produce feasible VNF placement. Adaptive parameter noise and prioritized experience replay are adopted, which improve the learning efficiency.

3) We develop a decentralized routing algorithm for the GBM. Since the routing algorithm is carried out in a decentralized manner, the availability of the GBM can be ensured in large-scale networks. Moreover, we prove the convergence and the feasibility of the proposed algorithm.

The remaining parts of this paper are organized as follows. Section 2 reviews the related works in the field of SFC. In Section 3, we describe the network model and the objective of optimization. In Section 4, we detail the structure and learning procedure of the proposed DRL agent. The decentralized routing algorithm for the GBM is illustrated in Section 5. In Section 6, the performance of the proposed algorithms are verified with simulations. Finally, we conclude this paper in Section 7.

II. RELATED WORKS
There have been plenty of works on the deployment of SFC. Initial studies are mostly conducted on conventional optimization algorithms. To reduce the operation cost and gain in service agility, the authors of [8] proposed an eigendecomposition-based algorithm which can dynamically place the VNF instances accordingly. The authors of [9] focused on the reliability issue and presented a suboptimal fast estimation algorithm to improve service quality. In [10], the authors proposed a heuristic deployment algorithm to reduce service delay and enhance the reliability of services. Works mentioned above mainly study the problem in a static scenario.

Some works also focused on the deployment of SFC in a dynamic environment. The authors of [11] considered the influence of the arrival of new users and the departure of old users. A column-generation-based algorithm is provided to reduce the resource consumption and signaling overhead during the deployment. The authors of [12] proposed a concept of ‘following chain’ to address the reliability issue, which re-arranges the placement of VNF instances based on the mobility pattern of users. The authors of [13] studied the scenarios with multiple service chains. A deployment algorithm based on binary integer programming is developed to improve network performance and reduce resource rent. An ILP (integer linear programming)-based flow scheduling algorithm is proposed in [14]. Apart from traditional decision variables (e.g., placement and routing), dynamic power control technology is considered, which can further reduce network energy consumption. Similar work is conducted in [15], in which the lightly-loaded servers are deactivated to reduce the energy consumption.

Recently, DRL has achieved great success in various fields such as network resource scheduling [16], big data
analyzing [17], Internet routing [18], multimedia stream rate regulating [19] and so on. Compared with traditional optimization algorithms, DRL performs better in time-varying systems. The goal of DRL is to learn a strategy through interaction with the environment. The policy is represented by a parameterized neural network (NN). The input of NN is the state (describing the environment) while the output is the corresponding action. DRL obtains knowledge from historical behaviors and ultimately gets a better strategy. Once the training is completed, the corresponding results can be obtained directly.

Due to the wild success of DRL, researchers began to apply DRL in the field of SFC. Initially, researchers considered the deployment of a single VNF, i.e., the auto-scaling of VNF. The authors of [20] proposed a learning algorithm based on parallel Q-learning. It makes a balance between the service delay and the network cost. The authors of [21] vertically scaled the VNF instance using the Actor-Critic algorithm. To improve resource utilization and service reliability, the algorithm adjusts the number of VNF instances by observing the overheads in the network.

Some authors began to use DRL to solve the VNF placement problem in SFC scenarios. The authors of [22] focused on the reliability and proposed a VNF instance placement algorithm based on Deep Q-Network (DQN). The network overhead is reduced while service reliability can be guaranteed. Similar work has been done in [23], which optimizes the placement of VNF instances using Q-learning. The authors of [24] considered the selection of VNF type and proposed a Q-learning-based algorithm to optimize the quality of service. However, these DRL-based algorithms usually have a low learning rate due to the large action space.

In our work, we redesign the NN structure of the DRL-agent based on DDPG and improve the learning efficiency effectively.

Along with the placement, flow scheduling also needs to be considered in the deployment of SFC. However, the dimension of flow scheduling is much higher than the VNF placement. Using the learning algorithm directly for flow scheduling would result in low learning efficiency. Thus, the authors of [7] proposed an improved algorithm based on DDPG, which relieves the issue by customized the action exploration and experience replay during the learning. The authors of [25] developed an Asynchronous Advantage Actor-Critic (A3C)-based algorithm for SFC deployment. They also assumed that flows with the same source node can only use VNF instances on the same clouds. The authors of [26] proposed a two-stage framework. Firstly, they used DQN to generate several candidate solutions. Then, traditional optimization algorithms are employed to refine the solutions. The authors of [27] decomposed the SFC deployment into a VNF placement procedure and a flow scheduling procedure. Deep learning (DL) is used to optimize the two procedures in an isolated manner. Similarly, the SFC deployment is decoupled to two procedures in our work. We apply a game-based method in the flow scheduling procedure to promote efficiency.

III. SYSTEM MODEL
A. GEO-DISTRIBUTED NFV SYSTEM
A typical geo-distributed NFV system consists of $N$ distinct data centers ($\mathcal{N} = \{1, 2, \ldots, N\}$). NFV providers run a DL-based scheduler for placement and scaling each VNF service chain to provide ubiquitous service to users [2]. There can be many network flows between various source regions and destination regions that traversing the same service chain. For ease of presentation, we group the individual network flows which have the same source region and destination region, and refer to the flow group as one flow hereinafter. Then each flow corresponds to a source-destination region pair. Our goal is to place sufficient instances of each VNF to geo-distributed data centers, satisfying the need of all flows. We consider that instances of the same VNF can be deployed onto different data centers, and different flows can share the same VNF instance.

Different service chains are scheduled by different schedulers, and we focus on one scheduler handling one service chain [6]. We suppose there are $M$ distinct types of VNF ($\mathcal{M} = \{1, 2, \ldots, M\}$) in the service chain and $I$ flows ($\mathcal{I} = \{1, 2, \ldots, I\}$) traversing the service chain. Let $n_i^k$ and $n_i^j$ be flow $i$’s source region and destination region, respectively.

The system works over a potentially large time span consisting of $T$ time slots ($\mathcal{T} = \{1, 2, \ldots, T\}$). Flow rate of each $i \in \mathcal{I}, f_i(t)$ ($t \in \mathcal{T}$), varies over time. The state-of-the-art methods [6], [28] with some advanced schemes like RNN (recurrent neural network) can comparatively predict the flow rate precisely. Since that is not the emphasis of this paper, we assume that $f_i(t)$ can be accurately predicted. The total flow demand in time slot $t$ can be represented as $F(t) = \sum_{i} f_i(t)$.

We use $x(t) = \{x_{m,n}(t)\}_{m,n}$ to describe the overall deployment in time slot $t$, where $x_{m,n}(t)$ denotes the number of instances of VNF $m$ to be deployed in data center $n$ in time slot $t$. For simplicity, we use $v_{m,n}$ to abstract the instances of VNF $m$ in data center $n$ owing to their homogeneity. Similarly, $e_{n_1,n_2}$ describes the transmission links between data center $n_1$ and $n_2$.

Note that routing strategies have significant influences on system performance. The feasible paths of flow $i$ (originating from $n_i^k$ and ending at $n_i^j$) is denoted by $\mathcal{P}_i$ and $\mathcal{P} = \cup_i \mathcal{P}_i$. A path $p \in \mathcal{P}_i$ is a set containing the constituent vertices (e.g., $v_{m,n}$)/edges (e.g., $e_{n_1,n_2}$). Then, the routing strategies can be represented as $y(t) = \{y_i(t)\}_{i \in \mathcal{I}}$, where $y_i(t) = \{y_{p}(t)\}_{p \in \mathcal{P}_i}$ describes the strategy of flow $i$. We interpret $y_{p}(t) \geq 0$ as the amount of flow $i$ choosing path $p \in \mathcal{P}$ in time slot $t$ and $\sum_p y_{p}(t) = f_i(t)$. Given $x(t)$ and $y(t)$, the aggregated workload on the instances in $v_{m,n}$ can be obtained, denoted as $w_{m,n}(t)$:

$$w_{m,n}(t) = \sum_{i \in \mathcal{I}} \sum_{p \in \mathcal{P}} y_{p}(t) \Lambda_{m,n,p}.$$  

(1)

Here $\Lambda_{m,n,p}$ is the path-vertex indicator. $\Lambda_{m,n,p} = 1$ implies a path $p$ traverses $v_{m,n}$, while $\Lambda_{m,n,p} = 0$ means $v_{m,n}$ is not
on path $p$. Note that $w_{m,n}(t)$ is equally distributed inside $v_{m,n}$ to achieve a better performance (due to their homogeneity). Let $P_m$ denote the processing capability of one instance of VNF $m$, which depends on resource capabilities of the VM (Virtual Machine), including CPU, memory and network I/O [29]. Thus, to make $y(t)$ feasible, $w_{m,n}(t) \leq \frac{1}{2}P_m x_{m,n}(t)$. Here $\xi \in (0, 1]$ is a coefficient for traffic burst, and the degree of traffic burst is proportional to $\frac{1}{\xi}$. We list important notations in Table 1.

**B. COST STRUCTURE**

The goal of the NFV provider is to minimize the overall cost when serving the flows. We consider the following costs.

1. **Operational cost.** The cost can be interpreted as the cost of renting VMs or containers for running the instances. Let $O_{m,n}$ be the operational cost for running each instance of VNF $m$ in data center $n$. The overall operational cost at $t$ can be represented as:

$$C_{\text{oper}}(t) = \sum_{m \in M} \sum_{n \in \mathcal{N}} O_{m,n} \cdot x_{m,n}(t) \quad (2)$$

2. **Deployment cost.** When launching a new VNF instance, a deployment cost is incurred for copying a VM image containing the network function to the data center. Let $D_{m,n}$ denote the cost for deploying VNF $m$ in data center $n$, while there is no instance of VNF $m$ deployed in $n$ in the previous time slot. We use binary variable $k_{m,n}(t)$ to indicate if any instance of VNF $m$ is deployed in data center $n$ in $t$ ($k_{m,n}(t) = 1$) or not ($k_{m,n}(t) = 0$). The total deployment cost in $t$ can be expressed as:

$$C_{\text{dep}}(t) = \sum_{m \in M} \sum_{n \in \mathcal{N}} D_{m,n} [k_{m,n}(t) - k_{m,n}(t - 1)]^+ \quad (3)$$

where $[\cdot]^+ = \max(\cdot, 0)$. For simplicity, we assume $k_{m,n}(0) = 0$.

3. **Delay cost.** The end-to-end delay consists of the transmission latency and the processing latency. Let $l_{n_1,n_2}$ denote the transmission delay from data center $n_1$ to $n_2$ ($n_1, n_2 \in \mathcal{N}$). Then, the overall transmission delay in time slot $t$ can be computed as:

$$d_{\text{trans}}(t) = \sum_{i \in \mathcal{I}} \sum_{p \in \mathcal{P}} [y^i_p(t) \cdot \sum_{e_{n_1,n_2} \in \mathcal{E}_p} l_{n_1,n_2}] \quad (4)$$

We assume a constant processing delay of VNF instances, which is denoted as $d_{m,n}^r$. Given a routing strategy $y(t)$, the overall processing delay is represented as:

$$d_{\text{proc}}(t) = \sum_{i \in \mathcal{I}} \sum_{p \in \mathcal{P}} \sum_{v_{m,n} \in \mathcal{V}} y^i_p(t) \cdot d_{m,n}^r \quad (5)$$

We multiply the delay by a cost per unit of time, $L$, to convert the overall delay to a cost, which is formulated as:

$$C_{\text{delay}}(t) = L \times [d_{\text{trans}}(t) + d_{\text{proc}}(t)] \quad (6)$$

Now, the overall cost of geo-distributed NFV system over the entire time span $T$ can be expressed as:

$$C_{\text{all}} = \sum_{t \in T} [C_{\text{oper}}(t) + C_{\text{dep}}(t) + C_{\text{delay}}(t)] \quad (7)$$

**C. A HYBRID DEEP LEARNING-BASED SCHEDULER**

At the beginning of each time slot $t$, the NFV provider adjusts service chain provision and routing strategy to accommodate upcoming flow variations. Both deployment $x(t)$ and routing strategy $y(t)$ are dynamically optimized in order to minimize the overall cost.

To achieve this goal, a hybrid deep learning-based scheduler is proposed in Figure 1. Different from existing DL-based schedulers, our proposed scheduler decouples the deployment procedure and the routing optimization into two separated modules, i.e., a DRL module (built for VNF instance placement, $x(t)$) and a GBM (integrated for routing strategies, $y(t)$). The former one is built on a model-free method, i.e., DDPG, which is a deep reinforcement learning framework for continuous action space. The DRL module...
improves the deployment policy in a trial-and-error manner. Here, the policy is the rule of determining $x(t)$ under different circumstances (e.g., different $f(t)$). During the learning procedure, the DRL module observes the system state and produces a $x(t)$ using its current policy. A reward signal will be then feedback associated with the current $x(t)$. Based on the reward, the policy of DRL would gradually be improved to obtain a smaller $C_{\text{delay}}$. The GBM is designed for routing based on the game theory. Given a VNF placement solution (i.e., $x(t)$) from the DRL module, the GBM is in charge of finding an optimal routing strategy (i.e., $y(t)$) with minimum $C_{\text{delay}}(t)$. It is noticed that the GBM is essentially a model-based algorithm that can use the model information (i.e., the gradient) to accelerate the learning speed and improve the accuracy. By the interactions among the environment, the proposed framework will finally obtain a good policy for both $x(t)$ and $y(t)$ under dynamic network scenarios. The details of the DRL module and the GBM are described in Section 4 and 5, respectively.

Compared to existing DL-based algorithms (which are mostly built on the model-free framework, e.g., DQN, AC, DDPG), the proposed hybrid deep learning-based scheduler has advantages on both accuracy and convergence speed. First, in our algorithm, the dimension of the learning in the DRL module is significantly reduced since $y(t)$ is decoupled into the GBM. The reduced dimension means that the learning procedure in our algorithm is more efficient and improves convergence and accuracy. Second, the optimization in our GBM is carried with a model-based optimization (based on game theory). The model-based algorithm can leverage the model information to accelerate the convergence speed and improve accuracy. Therefore, our proposed algorithm can achieve better performance in both convergence time and accuracy.

IV. A HYBRID LEARNING FRAMEWORK FOR SERVICE FUNCTION CHAINING

A. ONLINE DEEP REINFORCEMENT LEARNING

There are many available DRL algorithms with different features, such as DQN, DDPG and A3C. We design the DRL module based on the DDPG algorithm, which is a well-investigated deep reinforcement learning for continuous action space. Compared to other DRL algorithms, DDPG can achieve a better performance on large-scale problems, which is an admirable feature in our scenarios.

The basic DDPG training procedure can be viewed as a combination of policy-based and value-based methods. It learns the optimum policy and its value function through interactions with the environment. The agent is composed of two types of NN: the actor network (i.e., policy) and the critic network (i.e., estimated value function). The role of the actor is to define parameterized policy and generate actions according to the observed network state (e.g., predicted flow rate, $x(t-1)$), while the critic is in charge of evaluating current actions considering the reward (motivated by the action and $C_{\text{delay}}(t)$ from the GBM). In detail, the critic produces a temporal difference error (TD-error) which indicates whether current actions are getting better or worse than expected. Then, we adjust both the actor and the critic accordingly to reduce the TD-error mostly (with the sample gradient).

In our algorithm, the DRL module decides the deployment $x(t)$ in each time slot $t$. A straightforward design is to let the DRL agent decide the deployment of all VNFs in one action. However, this design leads to an exponential action space, which incurs significant training costs and slow convergence. Therefore, in each time slot $t$, we let the agent use $M$ (number of VNF types) steps to complete the deployment $x(t)$. In each step $t$, the agent only focuses on the deployment of one VNF $m \in M$.

B. DEFINITION OF STATE, ACTION, AND REWARD

Nevertheless, to utilize the DDPG algorithm, we shall first accurately define state, action, and reward. Upon a state $S$, an agent of DDPG takes an action $A$ and observes a new state $S'$. As a result, a reward can be calculated to judge the effectiveness of the action. We describe the state, action, and reward associated with our problem as following.

1) STATE

The state $S$ in time slot $t$ should include the following:

A) VNF deployment in the previous time slot $t-1$, i.e.,

$$\bar{x} = [\tilde{x}_{m,n}]_{m \in M, n \in N},$$

where $\tilde{x}_{m,n} = x_{m,n}(t-1)$.

B) Flow profile (including their sources, destinations, and demands), i.e., $r = [r_{m,n}]_{m \in M, n \in N}$. We set $r_{n_1,n_2} = f_i(t)/F_i(t)$ when there is an incoming flow $i$ with $n_1 = n_i^1$ and $n_2 = n_i^2$. Otherwise, $r_{n_1,n_2} = 0$.

C) Profile of VNF $m$ which is to be decided, i.e., $h = \{h_{m,\text{int}} \in M \}. Suppose in this step, we are about to deploy VNF $m$, then $h_m = F_i/\sum_{\xi}$. Otherwise, $h_m = 0$.

D) VNF deployment in previous steps in the current time slot $t$, i.e.,

$$\bar{x} = [\tilde{x}_{m,n}]_{m \in M, n \in N}.$$ We initialize $\tilde{x}_{m,n}$ to all zero at the beginning of time slot $t$. In each step, $\tilde{x}_{m,n}$ is continuously updated as $\tilde{x}_{m,n} \leftarrow \tilde{x}_{m,n} + A_n$, where $A$ is the action decided by NN in the current step and $A_n$ is the $n_{th}$ element of $A$.

To sum up, the state $S$ can be represented as

$$S = \{\bar{x}, \tilde{x}, r, h\}$$

(8)

2) ACTION

In each step, the DRL agent produces an action $A$ which is the deployment of current VNF $m$ over $N$ distinct data centers, i.e., $\sum_{n \in N} A_n = F_i/F_m/\xi$ and $A_n \geq 0, \forall n \in N$.

3) REWARD

The agents would get a non-zero reward at the end of each time slot. We set the reward as the inverse of the overall cost, in which $C_{\text{oper}}(t)$ and $C_{\text{dep}}(t)$ can be directly obtained inside the DRL module while $C_{\text{delay}}(t)$ is feedback from the GBM.

C. CUSTOMIZED DDPG ALGORITHM DESIGN

Incorporating the above definitions, we start to design our algorithm based on DDPG. However, there are some
limitations of DDPG which prevent our implementation. We summarize them as follows.

1) In traditional DDPG, the feasible space of $\mathcal{A}$ is a $N$-dimensional box. However, in our scenarios, the action $\mathcal{A}$ should be located on a simplex, i.e., $\sum n \in N A_n = F(t)/P_m/\xi$ and $A_n \geq 0, \forall n \in N$.

2) DDPG is originally designed for continuous action space while VNF deployment is a discrete decision-making process. Both NN architecture and the exploration mechanism of DDPG should be reinvestigated.

3) Conventional DDPG randomly draws the samples from the replay buffer for training. Sometimes, it may be trapped into bad samples with low efficiency, which leads to slow convergence and bad performance.

To tackle the above-mentioned issues, we customize our algorithm based on DDPG with the following techniques.

1) SOFTMAX LAYER AND WOLPERTINGER POLICY
In order to make the output of actor network (i.e., $\mathcal{A}$) feasible, we add an embedded layer (i.e., the Softmax layer) in Figure 2. The Softmax function (also known as the normalized exponential function) applied by the Softmax layer is a function that takes a vector of $N$ real numbers as input and normalizes it into a probability distribution consisting of $N$ probabilities proportional to the exponentials of the input numbers. By embedding a Softmax layer, we acquire a deployment distribution of VNF instances on $N$ data centers. Multiplying with a constant (i.e., $F(t)/P_m$), a relaxed deployment (continuous variable) of current VNF $m$ can be obtained. Finally, a series of candidate actions can be get with Wolpertinger policy which selects $k$ most similar actions (compared to relaxed deployment) using the $k$-nearest neighbour algorithm.

2) ADAPTIVE PARAMETER NOISE
Traditional RL (reinforcement learning) uses action space noise to change the likelihoods associated with each action the agent might take from one moment to the next. This method is not suitable for our scenarios since it might violate the feasibility of $\mathcal{A}$. Therefore, we apply the adaptive parameter noise for the action exploration, which adds adaptive noise to the parameters of the actor network. Parameter space noise injects randomness directly into the parameters of the agent, altering the types of decisions it makes such that they always fully depend on what the agent currently senses. Meanwhile, since the structure of the actor network is customized as Figure 2, the output of the action always falls in the feasible space of $\mathcal{A}$. Besides, similar to the cases in [7], during the learning procedure, we increase the possibilities of the solution on the boundary of the feasible space.

3) PRIORITIZED EXPERIENCE REPLAY
Traditional RL samples the experiences from replay buffer with equal importance, which ignores the difference in the value of each experience. Therefore, to improve the learning efficiency and avoid local optima, we adopt the technique of prioritized experience replay in [30]. In prioritized experience replay, the DRL agent draws experiences from replay buffer with weights proportional to their TD-error.

Incorporating the above design, we summarize the proposed algorithm in Algorithm 1.

**Algorithm 1 Training Procedure for the DRL Module**

1: Randomly initialize critic network $Q(S, \mathcal{A}|\theta^Q)$ and actor $\pi(S|\theta^\pi)$ with weights $\theta^Q$ and $\theta^\pi$
2: Initialize target network $Q'$ and $\pi'$ with weights $\theta^Q' \leftarrow \theta^Q$ and $\theta^\pi' \leftarrow \theta^\pi$
3: Initialize the replay buffer $B$
4: for $\tau = 1$ to $\tau_{\text{max}}$ do
5: Initialize the state $S$
6: while True do
7: Apply the adaptive parameter noise to obtain $a$
8: Adopt the Wolpertinger policy to obtain the placement (i.e., $x_{m,n}(t)$) for current VNF $m$
9: Execute the placement $x_{m,n}(t)$, observe the reward $R$ (including $C_{\text{delay}}(t)$ from the GBM), and new state $S'$
10: Store the transition $(S, \mathcal{A}, S', R)$ into replay buffer
11: if Replay buffer is full then
12: Apply the prioritized experience replay to sample a mini-batch of transition from replay buffer $B$
13: Update the actor policy with the sampled gradient
14: Update target networks for actor and critic
15: end if
16: if This episode is finished then
17: break
18: else
19: $S = S'$
20: end if
21: end while
22: end for
V. PROJECTION-BASED DECENTRALIZED ALGORITHM FOR ROUTING OPTIMIZATION

The GBM is responsible for seeking an optimal routing strategy $y(t)$ when given $x(t)$. Note that the optimization is repeated in every time slot $t$. Therefore, we omit the time stamp ‘$t$’ for simplicity and formulate the problem as follows:

\[
\min \quad C_{\text{delay}} \\
\text{s.t.} \quad y_p^i \geq 0, \quad \forall i \in \mathcal{I}, \quad \forall p \in \mathcal{P} \\
\sum_p y_p^i = f_i^i, \quad \forall i \in \mathcal{I} \\
w_{m,n} \leq \xi_{P_m x_{m,n}}, \quad \forall m \in \mathcal{M}, \forall n \in \mathcal{N}
\]

(9) and (10) give the mathematical integrity of flow routing while (11) prohibits the overload of VNF instances. Note that the feasible strategies of flow $i$ depend on other flows' behavior owing to (11). Thus, (11) is the coupled constraint while (9) and (10) are the local constraints.

(P1) is a typical linear programming problem with multiple constraints. There are several LP (linear programming) solvers exist, such as the interior point method. However, note that the dimension of $\mathcal{X}$ (the feasible region of $y_p^i$) is $N^M$. Consider the scenarios of SFC, the solution space of (P1) is extremely large. Meanwhile, the speed of solving (P1) matters since it is a part of the learning procedure of the DRL module (responsible for reward calculation). However, it is also noticed that a precise solution of (P1) is unnecessary since an approximated answer could also work for the learning. Therefore, to address the above-mentioned issue, we design a decentralized algorithm for the GBM in this section.

A. GAME FORMULATION

The routing problem in (P1) can be illustrated in Figure 3. Flows from $I$ distinct sources traverse through VNF instances sequentially. Each edge $(e_{n_1,n_2})$ and vertex $(v_{m,n})$ in Figure 3 incurs a cost which represents the latency in (6).

We solve the routing problem in Figure 3 with the framework of the selfish non-atomic routing game, $\mathcal{G} = (\mathcal{I}, \mathcal{Y}, U)$. The players are flows (i.e., $\mathcal{I}$) which distribute their traffic among feasible paths (i.e., $\mathcal{Y}$). For player $i$ (i.e., flow $i$), the routing cost of choosing $i$ is

\[
U_i^j \left( y^j, y^{-i} \right) = \sum_{e_{n_1,n_2} \in \mathcal{P}} I_{e_{n_1,n_2}} + \sum_{v_{m,n} \in \mathcal{P}} d_{m,n}^i
\]

where $y^{-i}$ is the routing strategies of all flows other than $i$. The cost of player $i$ can be represented as

\[
U_i^j \left( y^j, y^{-i} \right) = \sum_p y_p^i U_p^i \left( y_p^j, y_p^{-i} \right).
\]

Each player in $\mathcal{G}$ attempts to minimize its cost $U_i^j(y)$ by changing $y^j$. Then, we arrive at the following definition.

Definition (Nash equilibrium of $\mathcal{G}$) Let $y^* = \{ (y^i)^*, (y^{-i})^* \}$ be a strategy profile for the game $\mathcal{G}$. $y$ is a Nash equilibrium if, for every player $i \in \mathcal{I}$, the following relationships holds.

\[
y^i = \arg\min_{y^i \in \mathcal{Y}} U_i^j \left( y^j, (y^{-i})^* \right), \quad \forall i \in \mathcal{I}
\]

\[
y^j = \begin{cases} y^j \sum_p y_p^j = f_i^j, \quad y_p^j \geq 0, \forall p \in \mathcal{P} \\ \end{cases}
\]

Note that $\mathcal{G}$ does not consider the coupled constraint (11).

To tackle this issue, we propose an extended game $\mathcal{G}_{\text{ext}}$ with $I+1$ players. Besides the $I$ players in $\mathcal{G}$, an additional player ($i = I+1$) is introduced to cope with (11). The cost function of player $i = I+1$ is represented as

\[
J_{I+1}^i (y, \lambda) = \sum_m \sum_n \lambda_{m,n} \left[ w_{m,n} (y) - \xi_{P_m} \right]
\]

where $\lambda = \left[ \lambda_{m,n} \right]_{m \in \mathcal{M}, n \in \mathcal{N}}$ is the decision variables. Particularly, $\lambda_{m,n} \geq 0, \forall m \in \mathcal{M}, n \in \mathcal{N}$. Meanwhile, an auxiliary term is added to the cost function of player $i \in \mathcal{I}$ as

\[
J_i^i \left( y, y^{-i}, \lambda \right) = U_i^j(y) + \sum_m \sum_n \lambda_{m,n} w_{m,n} (y)
\]

We use $z = (y, \lambda)$ to denote the strategy profile of the game $\mathcal{G}_{\text{ext}}$. In $\mathcal{G}_{\text{ext}}$, each player in $\mathcal{G}_{\text{ext}}$ carry out following optimization to minimize its individual cost.

\[
\mathcal{G}_{\text{ext}} = \min_{\sum \lambda \in \mathcal{R}^{I+1 \times \mathcal{N}}} J_i^i \left( y, y^{-i}, \lambda \right) \quad \forall i \in \mathcal{I} \\
\min_{\sum \lambda \in \mathcal{R}^{I+1 \times \mathcal{N}}} J_{I+1} (\tilde{y}, \tilde{\lambda}), \quad i = I+1
\]

Note that in (17), there are no coupled constraints during the optimization.

Definition (Nash equilibrium of $\mathcal{G}_{\text{ext}}$) Let $\tilde{z}$ be a strategy profile of $\mathcal{G}_{\text{ext}}$. Then, $\tilde{z}$ is a NE (Nash equilibrium) of the game $\mathcal{G}_{\text{ext}}$ if

\[
\tilde{y}^i = \arg\min_{y^i \in \mathcal{Y}} J_i^i \left( y^j, \tilde{y}^{-i}, \tilde{\lambda} \right), \quad \forall i \in \mathcal{I} \\
\tilde{\lambda} = \arg\min_{\lambda \in \mathcal{R}^{I+1 \times \mathcal{N}}} J_{I+1} (\tilde{y}, \lambda), \quad i = I+1
\]

Now, we focus on the relationships between $\mathcal{G}_{\text{ext}}$ and the solution of (P1).

Theorem 1: Suppose $z^* = [y^*; \lambda^*]$ is a NE of the game $\mathcal{G}_{\text{ext}}$. Then, $y^*$ is the optimal solution of (P1).
Meanwhile, the KKT system of (P1) at the optimal point can be expressed as
\[
U_p^i(y) + \sum_{m,n} \lambda_{m,n} \Lambda_{p,m,n} - \mu_i^j + v^j = 0
\]
for all \(i \in I\), \(v^j, \mu_i^j \geq 0\), and
\[
\sum_{p \in P} y_p^j = f^j, \quad v^j \geq 0 \quad \forall \ i \in I
\]
(19)
From the perspective of player \(i = I + 1\), the follow relationships hold since \(\lambda^j\) is an optimal solution of (18).
\[
w_{m,n}(y) - \xi P_m v_{m,n} = -\eta_{m,n} \quad \forall m \in M, \forall n \in N
\]
\[
\lambda_{m,n} = 0 \quad \forall m \in M, \forall n \in N
\]
(20)
Meanwhile, the KKT system of (P1) at the optimal point can be expressed as
\[
\frac{\partial C_{delay}(y)}{\partial y_p^j} - \mu_i^j + v^j + \sum_{m,n} \lambda_{m,n} \Lambda_{m,n,p} = 0
\]
\[
y_p^j \geq 0, \mu_i^j \geq 0, \mu_i^j v_p^j = 0
\]
\[
\sum_i \sum_p y_p^j \Lambda_{m,n,p} - \delta P_m, \quad \lambda_{m,n} \geq 0
\]
\[
\lambda_{m,n} \left( \sum_p y_p^j \Lambda_{m,n,p} - P_m \right) = 0
\]
(21)
It is clear by inspection that \(y\) solves (V-A) with multiplier \(\lambda\), \(\mu\), and \(v\) if and only if \((y, \lambda, v)\) solves (V-A) and (20) with multipliers \(\mu, v, \eta = P_m - w_{m,n}(y)\), which completes the proof.

The existence of NE of \(G_{ext}\) can also be guaranteed since (P1) is essentially a convex optimization problem. The object function \(C_{delay}\) is bounded and the feasible solution space is closed and compact.

**B. PROJECTION-BASED DECENTRALIZED ALGORITHM FOR TASK ROUTING**

In this section, we propose a decentralized algorithm to find the optimal task routing strategy. Theorem 1 has shown that the optimal \(y\) of (P1) can be obtained via the NE of \(G_{ext}\). Note that (P1) arises in a high-dimensional space. Conventional game theory-based schemes (e.g., best response-based or replicator dynamics-based iterations) would result in slow convergence and high computational complexity, which is not preferable for our scenarios.

To tackle this issue, we propose a projected-based algorithm to find an optimal \(y\) of (P1). The proposed algorithm alternately updates \(y\) and \(\lambda\). We assume the presence of central control which broadcasts a tentative price \(\lambda_{t}\) at every step \(t\) of the algorithm. Based on this price and the aggregate of the strategies at the previous step, each agent locally computes \(y_{t+1}\). The central control updates the price to \(\lambda_{t+1}\) depending on the congestion of \(v_{m,n}\). The detailed of the proposed algorithm is described in Algorithm 2.

**Algorithm 2 Decentralized Algorithm for Flow Routing**

1: Set initiate strategies, i.e., \(y_{(0)}, \lambda_{(0)}\).
2: Local: Each player \(i \in I\) updates their strategies according to the following rule:
\[
y^j_{t+1} = P_y \left[ y^j_t - \alpha_t D^j_{y_t} \right]
\]
where the gradient \(D^j_{y_t}\) is
\[
D^j_{y_t} = \left[ U_p^j(y) + \sum_{m,n} \lambda_{m,n,p} \right]_{p \in P}
\]
(23)
3: Central: Player \(i = I + 1\) updates \(\lambda\) according to the following rule:
\[
\lambda_{t+1} = P_{\lambda} \left[ \lambda_{t} - \alpha_t D^\lambda_{\lambda_t} \right]_{n \in N}
\]
where the gradient \(D^\lambda_{\lambda_t}\) is
\[
D^\lambda_{\lambda_t} = \left[ w_{m,n}(y_{(t+1)}) \right]_{m \in M, n \in N} - P_m.
\]
(24)
4: Repeat Line 2 and 3 for \(\tau = t + 1\)

In the proposed algorithm, each player \(i \in I\) only knows its private information (including its local objective function \(J(y^i, y^{-i}, \lambda)\) and local feasible set \(\lambda^i\)) and a broadcast dual variable \(\lambda\). This is a preferable feature when the players interact over large-scale networks. Moreover, in each iteration, only projection and gradient operators are involved.

The gradient of players in Algorithm 2 is easy to obtain. Note that \(U_p^j(y)\) is a constant. Player \(i \in I\) only needs \(\lambda\) to compute its gradient. Meanwhile, the central control only need to collect the information about the congestion on \(v_{m,n}\). Moreover, the strategy update of player \(i \in I\) can be conducted in a parallel manner. In each iteration, only \(M \times N\) values need to be exchanged.

**Theorem 2:** Algorithm 2 converges to the NE of game \(G_{ext}\).

Proof: The compact form of the algorithm can be written as
\[
z_{t+1} = z_t + \alpha_t v^J(z_t).
\]
(25)
First, we prove (26) through the contradiction.
\[
\lim_{t \to \infty} \inf \left( v^J(z_t), z_t - \xi \right) \leq 0, \quad \forall z \in Z.
\]
(26)
If (26) is not hold (the limitation value of (26) is a positive value \(\xi\)), we can deduce the following relationship by
summing $\alpha_\tau \langle \nabla J (z_\tau), z_\tau - z \rangle$ from $0$ to $\infty$.

$$\sum_{\tau=0}^{\infty} \alpha_\tau \langle \nabla J (z_\tau), z_\tau - z \rangle$$

\[
\geq \sum_{\tau=0}^{k_0} \alpha_\tau \langle \nabla J (z_\tau), z_\tau - z \rangle + \sum_{k=k_0+1}^{\infty} \alpha_\tau \langle \nabla J (z_\tau), z_\tau - z \rangle
\]

\[
\geq \sum_{\tau=0}^{k_0} \alpha_\tau \langle \nabla J (z_\tau), z_\tau - z \rangle + \epsilon \sum_{k=k_0+1}^{\infty} \alpha_\tau \geq \infty \quad (27)
\]

However, for any $z \in Z$, we have

$$\|z_{\tau+1} - z\|^2 \leq \|z_\tau - \alpha_\tau \nabla J (z_\tau) - z\|^2 - \|\nabla J (z_\tau)\|^2$$

$$\leq \|z_\tau - z\|^2 - 2\alpha_\tau \langle \nabla J (z_\tau), z_\tau - z \rangle$$

$$+ \alpha_\tau^2 \|\nabla J (z_\tau)\|^2 \quad (28)$$

The first inequality holds due to the property of projection (Lemma 3.1 in [31]). By reorganizing (28), we can deduce

$$2\alpha_\tau \langle \nabla J (z_\tau), z_\tau - z \rangle \leq \|z_\tau - z\|^2 - \|z_{\tau+1} - z\|^2$$

$$+ \alpha_\tau^2 \|\nabla J (z_\tau)\|^2 \quad (29)$$

Note that $\alpha_\tau^2$ is summable and $\nabla J (z_\tau)$ can be bounded due to the boundness of the primary variable $\gamma$ and the multiplier $\lambda$ in [31]. Thus, the value of $2\alpha_\tau \langle \nabla J (z_\tau), z_\tau - z \rangle$ is a finite value which is in contradiction with (27). Thus, (26) holds.

(26) implies there is a limitation point $\bar{z} \in Z$ such that $\langle \nabla J (\bar{z}), z_\tau - \bar{z} \rangle \geq 0, \forall z_\tau \in Z$ [31]. Since $\nabla J (\cdot)$ is a pseudo-monotone function, we can directly deduce (30) (the property of pseudo-monotone functions).

$$\langle \nabla J (z_\tau), z_\tau - \bar{z} \rangle \geq 0, \quad \forall z_\tau \in Z. \quad (30)$$

Replace $z$ with $\bar{z}$ in (29) and apply (30), we have

$$\|z_{\tau+1} - \bar{z}\|^2 - \|z_\tau - \bar{z}\|^2 \leq \alpha_\tau^2 \|\nabla J (z_\tau)\|^2 \quad (31)$$

Summing (31) over $\tau \in [s, k]$, and we have

$$\|z_k - \bar{z}\|^2 \leq \|z_s - \bar{z}\|^2 + \sum_{\tau=s}^{k-1} \alpha_\tau^2 \|\nabla J (z_\tau)\|^2 \quad (32)$$

Take limitation on $\tau = k$ and then $\tau = s$, we can deduce

$$\lim_{k \to \infty} \sup_{z_k} \|z_k - \bar{z}\|^2 \leq \lim_{s \to \infty} \inf_{z_s} \|z_s - \bar{z}\|^2 \quad (33)$$

Then, we can conclude that $\{\|z_{\tau} - \bar{z}\|^2\}$ converges, which means Algorithm 2 would finally converge to $\bar{z}$. Furthermore, note that $\langle \nabla J (\bar{z}), z_\tau - \bar{z} \rangle \geq 0, \forall z_\tau \in Z$, which is essentially the first order condition of NE in (18). Thus, we can conclude that the limitation point produced by Algorithm 2 is a NE of game $\mathcal{G}_{\text{ext}}$. \hfill \square

VI. PERFORMANCE EVALUATION

A. SIMULATION SETUP

We evaluate our proposed algorithm through numerical results. The whole time span is one day and each time slot lasts half an hour. The flow pattern is set according to the real-world traffic from Huawei Technologies Co Ltd. The peak hour happens during 9 ~ 11 PM when the flow rate is about 720 Mbps. The bottom flow rate is about 160 Mbps and appears between 4 and 6 AM. For simplicity, all the flow rates are normalized within (0, 1]. We use eight Google data center locations to create our data center network (expect when testing algorithm performance with different number of data centers). The transmission latency between two data centers is proportional to their distance. There are six flows and four types of VNFs in the system on default. The data processing capability of each VNF instance is configured between 600 Mbps and 900 Mbps according to the VNF type.

The operational cost falls in (0, 1] and the deployment cost is between (0, 0.1] according to the Amazon EC2 pricing scheme. We set the weight $L$ for the latency cost to be 1.

In the proposed hybrid DRL-based framework, we add three fully-connected hidden layers in the actor network. Each layer contains 128 neurons whose activation function is ReLU (rectified linear unit). The output layer uses Softmax function and has $N$ neurons as outputs, where $N$ is the number of data centers. A similar structure (other than the output layer) is adopted for the critic network whose output layer is a linear neuron. During DRL training, the discount factor $\gamma$ is set to be 0.99. Learning rates for actor network and critic network are all 0.0001. We use Adam optimization algorithm for parameter update. Entropy weight $\beta$ is set to be 5. We build a memory buffer with a size equaling $1 \times 10^6$. During the training procedure, we use a batch with size 512.

We compare the proposed hybrid learning framework with the following algorithms:

1) Pure DDPG algorithm, P-DDPG: A DDPG-based algorithm similar to the proposed algorithm. P-DDPG does not use the advanced strategy exploration techniques and prioritized experience replay during the learning.

2) DRL-based algorithm, DRL: A state-of-art DRL algorithm proposed in [6], in which the authors assumed that each flow can only use the VNF instances in the same data center.

3) Customized DDPG algorithm, C-DDPG: A state-of-art DRL algorithm proposed in [7]. In C-DDPG, the actor network produces the VNF placement and flow routing simultaneously.

B. PERFORMANCE

Figure 4 plots the network cost with different number of VNF types (i.e., $M$). It can be seen that the network cost continues to rise as $M$ increases. Among all the algorithms, the proposed hybrid learning algorithm can achieve the minimum network cost with different $M$ values. In contrast, the network cost of the hybrid learning algorithm is about $9.2\%$, $15.5\%$ and...
30.9% lower than P-DDPG, DRL and C-DDPG on average, respectively. Additionally, we can see that, compared with P-DDPG and DRL, the advantages on network cost of the hybrid learning algorithm increase with $M$. The advantages against P-DDPG owing to the structure of learning agents and training strategies in the proposed hybrid learning, which enhance the efficiency of learning (i.e., the optimality on VNF instance deployment). The advantages increase as $M$ grows, which makes the gap in network costs become larger. Compared with DRL, the proposed hybrid learning algorithm permits a higher resource utilization, which reduces the network cost. Meanwhile, a simpler training process in our algorithm makes the advantages become more prominent.

As for C-DDPG, its action space (including feasible routes of all business flows) grows exponentially with $I$. Thus, when $M$ is large, the learning efficiency of C-DDPG is low and the result becomes unstable.

Figure 5 shows the algorithm performance with different number of user flows (i.e., $I$). As the number of flows increases, the deployment cost and the service latency continuously increase. It can be seen that the hybrid learning algorithm can always obtain the lowest network cost under all cases. By comparison, the network cost of the hybrid learning algorithm is about 11.9%, 16.7% and 28.9% lower than P-DDPG, DRL and C-DDPG on average, respectively. Moreover, the performance gaps enlarge with $I$. The reason is that, with a bigger $I$, the scale of the problem increases. The hybrid learning algorithm and P-DDPG adopt a game theory-based optimization method for the optimization of flow scheduling. Some model information (such as the gradient) can be used to improve the efficiency of the algorithms. The increase in $I$ has a relatively small effect on the learning efficiency of the algorithms (the hybrid learning and P-DDPG). On the other hand, DRL decides the flow path one by one. The increase in $I$ results in a more complicated training process (more steps involved in an episode). The dimension of the state space increases rapidly with $I$, which reduces the learning efficiency and results in larger performance gaps. C-DDPG optimizes the deployment of the service chain from the perspective of flow routing. The action space exponentially increases with $I$. Thus, the algorithm is not appropriate when $I$ is large.

Figure 6 plots the algorithm performance with different network sizes (i.e., the number of data centers $N$). Theoretically, the algorithms can get a better solution for the VNF deployment when given a bigger $N$. Thus, the curves in Figure 6 (i.e., network cost) gradually decrease with $N$. It can be seen that among all the algorithms, the proposed hybrid learning algorithm can achieve the minimum network cost under all scenarios. Compared with P-DDPG and DRL, the network cost is 11.3% and 20.2% lower on average separately. Meanwhile, as $N$ increases, the performance gaps (in percentages) increases rapidly. This shows that the proposed hybrid learning algorithm can work efficiently in large-scale networks and has better scalability. The action space in C-DDPG increases exponentially with $N$. Thus, its result is unreliable with a bigger $N$. When $N$ increases from 8 to 10, the network cost under C-DDPG even increases.
Figure 7 shows how the network cost varies with time. It can be seen that the changes in the network cost are basically consistent with the fluctuations of the total demands. At around 5:00 AM, the network cost is the lowest. As the total demands increase, it admits its peaks at around 10:00 PM (i.e., 22:00). With more service flows in the network, more VNF instances need to be deployed. Thus, higher operational cost and deployment cost will be incurred. Therefore, the variations on the network cost and total demands share a similar tendency. Compared with other algorithms (P-DDPG, DRL and C-DDPG), the hybrid learning algorithm can achieve the minimum network cost. Its cost is 8.9%, 18.6% and 29.8% lower than P-DDPG, DRL and C-DDPG on average. The advantages of the proposed algorithm mainly come from the simple learning structure and the efficient flow scheduling algorithm.

Figure 8 plots the variation of network cost under different delay coefficients (i.e., $L$). Service latency is one of the key indicators that affect user service experience. By modifying the delay coefficient, the performance characteristics in different scenarios can be acquired. It can be seen that in both delay-sensitive and delay-insensitive scenarios, the hybrid learning deployment algorithm can effectively deploy the service chain. This shows that the proposed algorithm has strong adaptability in different scenarios. Operators can employ the proposed algorithm to serve various latency requirements in different scenarios. In contrast, the network cost of the proposed algorithm is 11.8%, 19.4% and 29.1% lower than P-DDPG, DRL and C-DDPG on average.

In Figure 9, we compare the learning speed (or convergence speed) between the proposed algorithm and DDPG. DQN and A3C are not taken into comparison due to their different learning structures. We can observe that the proposed algorithm converges nearly three times faster than DDPG. Meanwhile, a lower network cost can be achieved.

VII. CONCLUSION

In this paper, we propose a hybrid DRL learning framework for the SFC problem. The VNF placement and flow routing are jointly optimized to minimize the overall cost, including the operational cost, the deployment cost, and the delay cost. Unlike existing DRL-based algorithms, we decouple flow routing from the learning agent into a game-based module (GBM). The DRL agent, which is designed based on the DDPG, only focuses on the optimization of the VNF deployment policy. To further improve the learning efficiency, we customize the NN structure of the DRL agent and adopt several techniques such as adaptive parameter noise, Wolpertinger policy, and prioritized experience replay. In the GBM, we design a decentralized algorithm for flow routing based on game theory to address the scalability. During the learning procedure, the DRL agent adjusts the deployment policy based on the reward generated by the GBM. The learning efficiency is better than conventional DRL-based algorithms owing to two reasons. First, in the proposed algorithm, the action space of the DRL agent is significantly reduced. Second, conventional DRL-based algorithms are essentially model-free methods. However, in our algorithm, the GBM conducts the optimization of flow routing with model-based algorithms. The algorithm efficiency is improved since module-based information is adopted, such as the gradient. Finally, through numerical results, we validate
that the proposed algorithm can achieve a lower system cost through limited training episodes compared to existing DRL-based algorithms.

REFERENCES

[1] P. Quinn and J. Guichard, “Service function chaining: Creating a service plane via network service headers,” Computer, vol. 47, no. 11, pp. 38–44, Nov. 2014.

[2] M. Ghaznavi, N. Shabirai, S. Kamali, R. Ahmed, and R. Boutaba, “Distributed service function chaining,” IEEE J. Sel. Areas Commun., vol. 35, no. 11, pp. 2479–2489, Nov. 2017.

[3] G. Mezreij, T. Botero, “Resource allocation in NFV: A comprehensive survey,” IEEE Trans. Netw. Service Manage., vol. 13, no. 3, pp. 518–532, Sep. 2016.

[4] S. Troia, R. Alvizu, and G. Maier, “Reinforcement learning for service function chain reconfiguration in NFV-SDN metro-core optical networks,” IEEE Access, vol. 7, pp. 167944–167957, 2019.

[5] M. Nakamura, Y. Sato, and H. Shimoniishi, “Environment-adaptive sizing and placement of nfv service chains with accelerated reinforcement learning,” in Proc. IFIP/IEEE Symp. Integr. Netw. Service Manage. (IM), Apr. 2019, pp. 36–44.

[6] Z. Luo, C. Wu, Z. Li, and W. Zhou, “Scaling geo-distributed network function chains: A prediction and learning framework,” IEEE J. Sel. Areas Commun., vol. 37, no. 8, pp. 1838–1850, Aug. 2019.

[7] L. Gu, D. Zeng, W. Li, S. Guo, A. Zomaya, and H. Jin, “Deep reinforcement learning based VNF management in geo-distributed edge computing,” in Proc. IEEE 39th Int. Conf. Distrib. Comput. Syst. (ICDCS), Jul. 2019, pp. 934–943.

[8] M. Mechi, C. Ghribi, and D. Zeghlache, “A scalable algorithm for the placement of service function chains,” IEEE Trans. Netw. Service Manage., vol. 13, no. 3, pp. 533–546, Sep. 2016.

[9] R. Yu, G. Xue, and Z. Zhang, “QoS-aware and reliable traffic steering for service function chaining in mobile networks,” IEEE J. Sel. Areas Commun., vol. 35, no. 11, pp. 2522–2531, Nov. 2017.

[10] H. Hwilo, M. Jammal, and A. Shami, “Network function virtualization-aware orchestrator for service chaining function in the cloud,” IEEE J. Sel. Areas Commun., vol. 37, no. 3, pp. 643–655, Mar. 2019.

[11] J. Liu, W. Lu, F. Zhou, P. Lu, and Z. Zhu, “On dynamic service function chain deployment and adjustment,” IEEE Trans. Netw. Service Manage., vol. 14, no. 3, pp. 543–553, Sep. 2017.

[12] Y. T. Chen and W. Liao, “Mobility-aware service function chaining in 5G wireless networks with mobile edge computing,” in Proc. IEEE Int. Conf. Commun. (ICC), May 2019, pp. 1–6.

[13] P. Mei, H. Xue, and D. Li, “Efficiently embedding service function chains with dynamic virtual network function placement in geo-distributed cloud system,” IEEE Trans. Parallel Distrib. Syst., vol. 30, no. 10, pp. 2179–2192, Oct. 2019.

[14] N. Hui, A. Tomassilli, F. Giroire, and B. Jaumard, “Energy-efficient service function chain provisioning,” IEEE/OSA J. Opt. Commun. Netw., vol. 10, no. 3, pp. 114–124, Mar. 2018.

[15] M. M. Tajiki, S. Salsano, L. Chiarafiglio, M. Shojafar, and B. Akbari, “Joint energy efficient and QoS-aware path allocation and VNF placement for service function chaining,” IEEE Trans. Netw. Service Manage., vol. 16, no. 1, pp. 374–388, Mar. 2019.

[16] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, “Resource management with deep reinforcement learning,” in Proc. 15th ACM Workshop Hot Topics Netw. (HotNets), 2016, pp. 50–56.

[17] Y. He, F. R. Yu, N. Zhao, V. C. M. Leung, and H. Yin, “Software-defined networks with mobile edge computing and caching for smart cities: A big data deep reinforcement learning approach,” IEEE Commun. Mag., vol. 55, no. 12, pp. 31–37, Dec. 2017.

[18] Z. Xu, J. Tang, J. Meng, W. Zhang, Y. Wang, C. H. Liu, and D. Yang, “Experience-driven networking: A deep reinforcement learning based approach,” in Proc. IEEE Conf. Comput. Commun. (INFOCOM), Apr. 2018, pp. 1871–1879.

[19] H. Mao, R. Ne'travali, and M. Alizadeh, “Neural adaptive video streaming with pensieve,” in Proc. Conf. ACM Special Interest Group Data Commun., Aug. 2017, pp. 197–210.

[20] E. Barrett, E. Howley, and J. Duggan, “Applying reinforcement learning towards automating resource allocation and application scalability in the cloud,” Concurrency Comput., Pract. Exper., vol. 25, no. 12, pp. 1656–1674, Aug. 2013.

[21] H. T. Nguyen, T. V. Do, A. Hegyi, and C. Rotter, “An approach to apply reinforcement learning for a VNF scaling problem,” in Proc. 22nd Conf. Innov. Clouds, Internet Netw. Workshops (ICIN), Feb. 2019, pp. 94–99.

[22] H. R. Khezri, P. A. Moghaddam, M. K. Farshbafian, V. Shah-Mansouri, H. Kebriaei, and D. Niyato, “Deep reinforcement learning for dynamic reliability aware NFV-based service provisioning,” in Proc. IEEE Global Commun. Conf. (GLOBECOM), Dec. 2019, pp. 1–6.

[23] S. I. Kim and H. S. Kim, “A research on dynamic service function chaining based on reinforcement learning using resource usage,” in Proc. 9th Int. Conf. Ubiquitous Future Netw. (ICUFN), Jul. 2017, pp. 582–586.

[24] G. Li, H. Zhou, B. Feng, G. Li, and S. Yu, “Automatic selection of security service function chaining using reinforcement learning,” in Proc. IEEE Globecom Workshops (GC Wkshps), Dec. 2018, pp. 1–6.

[25] S. Guo, Y. Dai, S. Xu, X. Qiu, and F. Qi, “Trusted cloud-edge network resource management: DRL-driven service function chain orchestration for IoT,” IEEE Internet Things J., vol. 7, no. 7, pp. 6010–6022, Jul. 2020.

[26] J. Sun, G. Huang, G. Sun, H. Yu, A. K. Sangaiah, and V. Chang, “A Q-learning-based approach for deploying dynamic service function chains,” Symmetry, vol. 10, no. 11, p. 646, Nov. 2018.

[27] B. Li, W. Lu, S. Liu, and Z. Zhu, “Deep-learning-assisted network orchestration for on-demand and cost-effective vNF service chaining in inter-DC elastic optical networks,” J. Opt. Commun. Netw., vol. 10, no. 10, pp. D29–D41, 2018.

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

[29] Y. Jia, C. Wu, Z. Li, F. Li, and A. Liu, “Online scheduling of NFV service chains across geo-distributed datacenters,” IEEE/ACM Trans. Netw., vol. 26, no. 2, pp. 609–710, Apr. 2018.

[30] Y. Hou, L. Liu, Q. Wei, X. Xu, and C. Chen, “A novel DDPG method with prioritized experience replay,” in Proc. IEEE Int. Conf. Symp. Man Cybern. (SMC), Oct. 2017, pp. 316–321.

[31] M. Zhu and E. Frazzoli, “Distributed robust adaptive equilibrium computation for generalized convex games,” Automatica, vol. 63, pp. 82–91, Jan. 2016.

TAO TANG received the B.E. degree in communication engineering from the University of Electronic Science and Technology of China, Chengdu, China, in 2013, where he is currently pursuing the Ph.D. degree in information and communication engineering with the School of Information and Communication Engineering. His research interests include social networks, cloud computing, and complex networks.

BINWEI WU received the Ph.D. degree in information and communication engineering from the University of Electronic Science and Technology of China, Chengdu, China, in 2020. His research interests include novel network architecture, MEC networks, and vehicular networks.

GUANGMIN HU received the Ph.D. degree from the Chengdu University of Technology, in 2000. He is currently a Professor with the University of Electronic Science and Technology of China. His research interests include network signals and information processing, information visualization, and network behavior and security. **