EdgeGym: A Reinforcement Learning Environment for Constraint-Aware NFV Resource Allocation

Jing Su, Suku Nair
AT&T Center for Virtualization
Southern Methodist University
Dallas, USA
{suj, nair}@smu.edu

Leo Popokh
CMS
Hewlett Packard Enterprise
Dallas, USA
leonid.i.popokh@hpe.com

***Abstract*—Optimizing resource allocation in Network Functions Virtualization (NFV) deployment remains a challenging problem due to the complex interactions between network functions and the limited resources available at the network edge. Deep reinforcement learning (DRL) has achieved impressive results in a variety of domains. This paper presents EdgeGym, a reinforcement learning environment to simulate the edge network contexts and constraints for NFV resource allocation. EdgeGym allows researchers and practitioners to evaluate and compare different reinforcement learning algorithms for optimizing the allocation of resources in NFV environments, taking into account various constraints such as affinity policies and maximum latency. We demonstrate the effectiveness of EdgeGym through extensive experiments on training and action masking efficiency. EdgeGym provides a reliable framework for advancing the DRL agent performance in NFV resource allocation and paves the way for further research in this area.**

***Index Terms*—Resource Allocation, NFV, Deep Learning, Reinforcement Learning Environments, Gym**

I. Introduction

Network Function Virtualization (NFV) enables network operators to accelerate the deployment of new virtual devices in a more dynamic approach that leads to building a software-based network to reduce operating expenses and capital expenses. With the growth of edge computing and cloud computing, software-based virtual network functions (VNFs) can be deployed on edge servers close to end users to support a comprehensive range of new services with high bandwidth and low latency. However, one of the main challenges in deploying NFV is provisioning VNFs with sufficient resources on edge servers without compromising network quality of service (QoS) and Service-Level Agreement (SLA). Resource allocation optimization is widely recognized as an $\mathcal{NP}$-hard problem [1]–[3] in the telecommunication operation and maintenance area, which is nontrivial to address in a complex network as it involves a vast number of optimization variables resulting from the multidimensional space of network component parameters and states. Fig. 1 illustrates a typical NFV Infrastructure (NFVI) architecture and demonstrates that the NFVI contains a wide variety of resource types and stakeholders. Accordingly, determining the optimal resource allocation is an important and challenging problem to examine in edge computing and cloud computing.

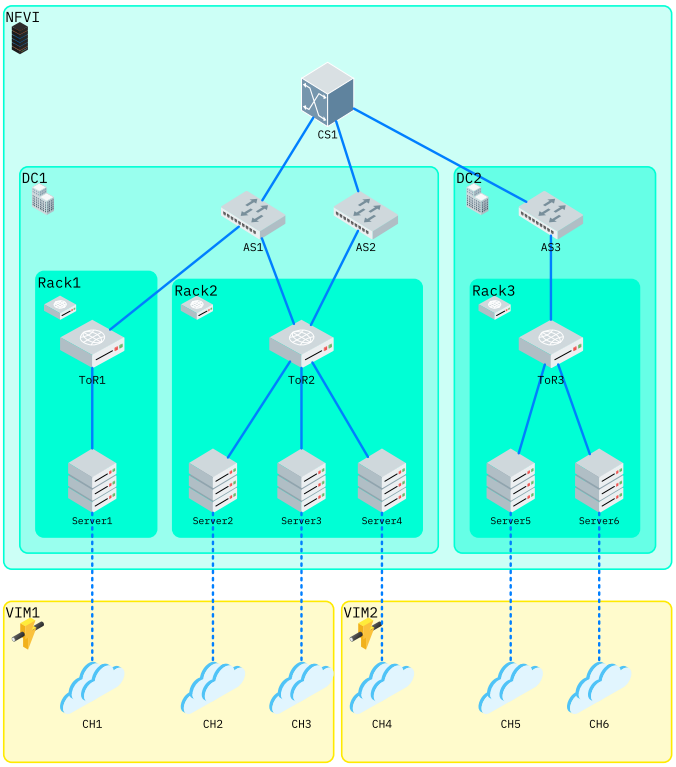


Fig. 1: A typical NFVI architecture comprises a wide variety of resource types and stakeholders.

Reinforcement Learning (RL) is a machine learning technique for the process of making informed decisions by learning from experience. The introduction of deep neural networks made deep reinforcement learning (DRL) possible to process high-dimensional inputs without the need for hand-crafted feature representations. Deep reinforcement learning has apparent benefits in resource allocation, which can exhibit a fast execution time than search-based approaches and comprehensively learn the problem structure while expanding the learning capacity. In addition, the trained models are effortless to use and deploy in a distributed manner, and the inference process can be accelerated by existing hardware (e.g., graphics card).

OpenAI Gym [4] provides a standardized toolkit for devel-

oping and comparing reinforcement learning algorithms with a common interface. It allows for direct comparison between RL algorithms and assessment of generalization performance. It also provides a unified interface to convert any arbitrary task into a custom Gym environment.

In this context, this paper presents EdgeGym, a novel environment focused on constraint-aware NFV resource allocation for the training and evaluating reinforcement learning agents and algorithms while compliant with the European Telecommunication Standards Institute (ETSI) NFV specification [5].

The contribution in this paper can be summarized as follows:

1) We formulate a Markov decision process (MDP) to describe the NFV resource allocation problem in a reinforcement learning environment.
2) We present an ETSI-compliant framework to build, train and evaluate RL agents and algorithms for NFV resource allocation research.
3) We propose a set of methods to implement various constraints in the production environment.

The remainder of this article is organized as follows. Section II addresses the related work. Section III briefly describes RL and resource allocation and presents the theoretical basis. Section IV details the architecture design, internal components, and agent development. Finally, Section V concludes this article and discusses future research.

II. Related Work

In recent years studies and research on the resource allocation and VNF placement problem have become a hot issue in academia and industry. Typically this problem is categorized as a resource management problem in NFV systems [6]. In order to achieve some specific optimization objectives such as minimizing the number of occupied servers and end-to-end latency, some mathematical programming methods such as Binary Integer Programming (BIP) [3], [7], [8], Integer Linear Programming (ILP) [9]–[12], and mixed ILP (MILP) [13]–[16] are extensively used. However, since the resource allocation problem is $\mathcal{NP}$-hard [17], it is challenging to efficiently search for optimal solutions, particularly in large-scale networks with a large action space. For this reason, heuristic solutions are typically proposed with near-optimal results but a short running time [18], while constraint programming methods are proposed with an optimal guarantee but a prohibitive execution time [19].

Nevertheless, we identified some works which focused on utilizing machine learning to solve the NFV resource allocation and VNF placement problem. To the best of our knowledge, there are no frameworks that help researchers and engineers efficiently design, train, and evaluate RL agents to optimize the NFV resource allocation with consideration of constraints.

Wang *et al.* [20] introduced a k-nearest neighbor (k-NN) supervised learning classifier to solve the resource allocation problem. Xiao *et al.* [21] presented a policy gradient based adaptive DRL approach to acquire dynamic network state transitions and optimize the deployment of service function chains (SFCs). Pei *et al.* [22] presented a Double Deep Q Network based off-policy DRL algorithm to determine the optimal VNF placement solution. Sun *et al.* [23] proposed a data-driven method to combine DRL and Graph Neural Networks (GNN) to solve the VNF placement problem. Su *et al.* [24] proposed a policy gradient based DRL approach to make optimal allocation decisions.

Our work can set up an RL gym for any definition data compatible with the ETSI NFV standard and delivers a fully customizable environment. Experimental RL agent policies and algorithms can be implemented and examined using EdgeGym. A stochastic baseline agent is also included for preliminary comparison.

III. Preliminaries

This section introduces the notation and formalizes the concept of reinforcement learning for NFV resource allocation, the action selection and evaluation for agent training, and constraints of latency and affinity.

A. Reinforcement Learning and Resource Allocation

The regular formalization of RL contains six key elements: *agent*, *observation*, *action*, *state*, *environment*, and *reward*. The agent performs an action upon observation, and the environment feedbacks a reward and new state resulting from that action. We defined them in the NFV resource allocation context as follows:

Agent: The agent makes resource allocation decisions as actions. It works like the OpenStack scheduler [9] in some ways. Fig. 2 demonstrates the agent acquiring information from VM and NFVI constraints and then making resource allocation decisions to NFV Management and Orchestration (MANO) for actual scheduling.

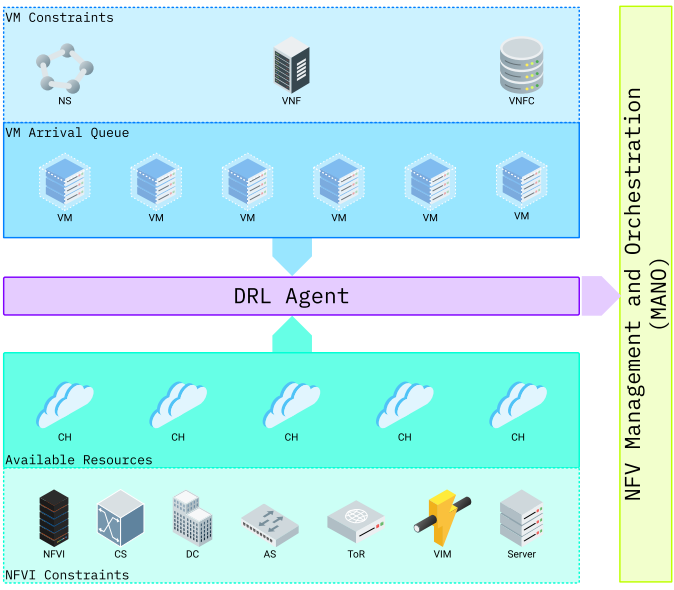


Fig. 2: Involved entities and constraints for DRL agent.

Observation: The observation reflects what the agent observes from the current state of the environment. It will include current virtual machine (VM) demands and NFVI topology as supporting information for the agent.

Action: The actions are a set of all possible moves the agent can make at any given time point. Assigning a VM to a compute host (CH) is an action.

State: The state denotes the solid and present status of the agent in the environment, like the currently placed VMs, and the available computing resources. It is highly related to observation, but a state can be either partially or fully observable.

Environment: The environment is where the agent executes actions and where states and rewards are computed. In EdgeGym, NFVI topology and VM arrival queue form the environment.

Reward: The reward is the feedback that measures how the action succeeds or fails. In EdgeGym, the reward of action is mainly impacted by latency impact, soft constraint satisfaction, and hard constraint violation.

In summary, taking action causes a state transition, after which the agent receives observations and a reward. The overall objective for such a system is to learn a policy π, which is a means to choose an action at any given state in order to maximize the expected reward over a certain period of time. The policy samples an action $a \in \mathcal{A}$ at state s according to $\pi(a \mid s)$ as a resource allocation decision to NFV MANO.

B. Latencies and Affinities

Latency in EdgeGym is measured as hops. A hop is defined as a data packet transmitted from one entity to the next. The VM latency is the maximum communication hops to all other VMs under the same VNF defined in Equation (1). A lower communication overhead means lower latency and leads to compact placement in terms of physical distance, thus achieving maximum system capacity and reducing energy costs. An efficient resource allocation decision seeks to minimize the overall VM communication overhead within each VNF. For this reason, as shown in Equation (2), the VNF latency is determined by the highest VM latency for all VMs under this VNF.

$$latency_{vm_i} = \max(\left[latency_{vm_i vm_j} \text{ for } vm_j \in VNF_{vm_i}\right]) \tag{1}$$

$$latency_{vnf_i} = \max([latency_{vm_i} \text{ for } vm_i \in vnf_i]) \tag{2}$$

Affinity indicates that all the instances in this scope must be assigned to the same containers, and anti-affinity indicates that no instances in the scope can be placed on the same container. In this context, the instances refer to VMs, the scope refers to network service (NS), and the container refers to a compute host or a data center. As a result, each VM will have corresponding attributes ch_policy and dc_policy for fine-grained control. The possible values of a policy attribute is defined in Equation (3). If a policy attribute is unset, it indicates there is no affinity enforcement for the related container in the scope.

$$policy = \begin{cases} 1, & \text{affinity} \\ 2, & \text{anti-affinity} \\ 0, & \text{unset.} \end{cases} \tag{3}$$

C. Action Selection and Evaluation

We categorized resource occupancy, maximum latency tolerance, and affinity policies as hard constraints, which are the hard limitations that cannot be broken. In contrast, soft constraints like network services distance minimization are a less forcing nature that may be violated but as many as possible should be satisfied. In EdgeGym, the hard constraints are enforced by masking out [25] the ineligible action choices. After applying the invalid action masking, the logit of the impossible action are replaced with $-\infty$ in which logit l is the vector of unnormalized predictions.

To stimulate the agent to reduce the latency of all VMs within the NS and avoid dispersed VM placements, we propose an approach that adds multi-stage weight penalties to the rewards. The multi-stage VM-to-VM placement cases and corresponding penalties are defined in Table I. A standalone VNF is a VNF that does not belong to any NS. The VLINK stands for a virtual link in this table which implies a physical connection path.

TABLE I: Multi-Stage VM-to-VM Placement Penalty Definition

Case	Penalty
No VLINK to same NS	0
Any VM inside standalone VNF	0
Have VLINK to same NS and inside same server	0
Have VLINK to same NS and inside same rack	1
Have VLINK to same NS and share aggregation switch	2
Have VLINK to same NS and inside same data center	4
Have VLINK to same NS but none of above satisfied	6

Consequently, the reward R_a for action a will be the negative value of the sum of latencies and penalties for current VM vm_a in Equation (4). Hence, lower penalties and lower latency to all other VMs on the same VNF will result in a higher reward value. For this reason, the agent will learn to minimize the latency and place the VM near the sibling VMs as close as possible.

$$R_a = -(\sum_{j=0}^{j<N_{vnf_{vm_a}}} latency_{vm_a vm_j} + \sum penalty_{vm_a}) \tag{4}$$

IV. Methodology

The following section will discuss the methodology behind our work, including MDP simulation and system design, as well as special workarounds for handling constraints.

A. MDP Simulation

The Markov decision process (MDP) [26] formally describes an environment for reinforcement learning. We model the NFV resource allocation problem in EdgeGym as an MDP tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$.

In this context, $\mathcal{S}$ is a finite set of NFVI states while the states include a representation of network topology and available resources. $\mathcal{A}$ is a finite set of actions in which an action is a resource allocation decision act by the agent. In an MDP, an agent is the entity training to make correct decisions. At a time step t, it observes from the environment state $s_t \in \mathcal{S}$ and chooses from a finite set of actions $a_t \in \mathcal{A}$.

Besides, $\mathcal{P}$ is a state transition probability matrix while the current state depends only on its immediate previous state as $\mathcal{P}^a_{ss'} = \mathbb{P}[S_{t+1} = s' \mid S_t = s, A_t = a]$. The environment transforms following a dynamics $p : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to [0, \infty)$ defining the probability density of the next state. Moreover, $\mathcal{R}$ is a reward function that $\mathcal{R}^a_s = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a]$. Eventually, the environment sends a feedback reward as $r : \mathcal{S} \times \mathcal{A} \to [0, 1]$.

The ultimate goal of the agent is to maximize the cumulative reward, i.e., $\max \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t r\left(s_t, a_t\right)\right]$, where γ is a discount factor to account for delayed rewards and $\gamma \in [0, 1]$. In these circumstances, this formulation is simplified by focusing on the problem with immediate rewards. Hence the delay will not be considered.

B. System Design

The recent breakthroughs in integrating deep neural networks with RL have been made possible by the availability of massive quantities of simulated data for learning by various algorithms. Simulated environments in which RL algorithms can learn from a virtually unlimited data source are becoming essential for creating cutting-edge algorithms. However, little work has been done to date on designing an RL environment specifically for training and evaluating resource allocation in NFV-enabled networks. For this reason, our work will focus on designing and implementing such an environment. We propose a gym-based environment specifically for the task of resource allocation and VNF placement. In summary, the EdgeGym consists of the following components:

- *Preprocessing*: parse the input NFVI topology data in JSON [27] or TOML [28] format and preprocesses the data in the proper format for numericalization.
- *Numericalization*: receives the preprocessed data and generates the action and state space. Preload the data and render caches like server hops table for training acceleration.
- *Action Space*: analyze the compute hosts and generate the discrete action space.
- *Observation Space*: sets the available states of the resource occupancy with related contextual knowledge.
- *Reward Function*: a predefined and configurable function for the rewards and penalties that the environment will feedback the agent for its actions.
- *Edge Core*: the principal component that plays that interacts with the agent as the environment. It obtains the current state and observations, receives the actions from the agent, and sends the feedback in accordance with the reward function. The core also records statistics and historical information such as step rewards and latencies.

Furthermore, the EdgeGym includes these main procedures:

- **Initialization**: initializes the empty occupancy state or restores from a previously saved allocation state. Initializes all entities with their initial state and builds the action and observation space.
- **Step**: perform the resource allocation decision, calculate and feedback rewards and the next state to the agent.
- **Reset**: invert all the performed resource allocation actions and revert the occupancy state to the initial status.

Besides, EdgeGym has a built-in stochastic baseline agent for comparison when developing a new agent policy or algorithm. This agent randomly selects the assigned compute host for the incoming VM without learning from the interactions.

C. VM Arrival Queue

In a modern telecommunication cloud, VNFs are formed by groups of single or interconnected virtual machines (VMs). Depending on demands, one or several VMs will arrive at the time step t. EdgeGym maintains a VM arrival queue to manage the sequential arrival while implementing the QoS and SLA provision. Each VM will be assigned a *priority* attribute to determine its queue position. The priority value can be calculated on the basis of the QoS and SLA requirements or the arrival sequence. Fig. 3 depicts the EdgeGym environment and agent observation.

At each timestep, the agent observes the current VM and NFVI state and then makes a resource allocation action decision. The EdgeGym receives this action and calculates a reward value using the reward function. The agent learns from the reward and updates the policy to maximize future expected rewards.

D. Server Hops Table

An objective of EdgeGym is to obtain an efficient placement of network services and routing of the VM communication flows without violating the constraints of the maximum resource capacity of the compute hosts and tolerable latency of VMs and VNFs. Since we define VM latency as the maximum communication latency to all other VMs in the same VNF. The VM-to-VM communication latency can be in four cases depending on their position relationship: *Inside Server*, *Inside Rack*, *Share AS*, and *Cross DC*. Fig. 4 demonstrates the corresponding flow path and latency in terms of hops.

Each VM has a *vm_max_ltcy* attribute that specifies the maximum latency that can be tolerated. Any VM latency that exceeds its *vm_max_ltcy* value in a placement result means that the placement violates the max latency constraint and is thus infeasible. The latency of a VNF is obtained by calculating the maximum latency for all of its VMs. Similarly, VNFs have a

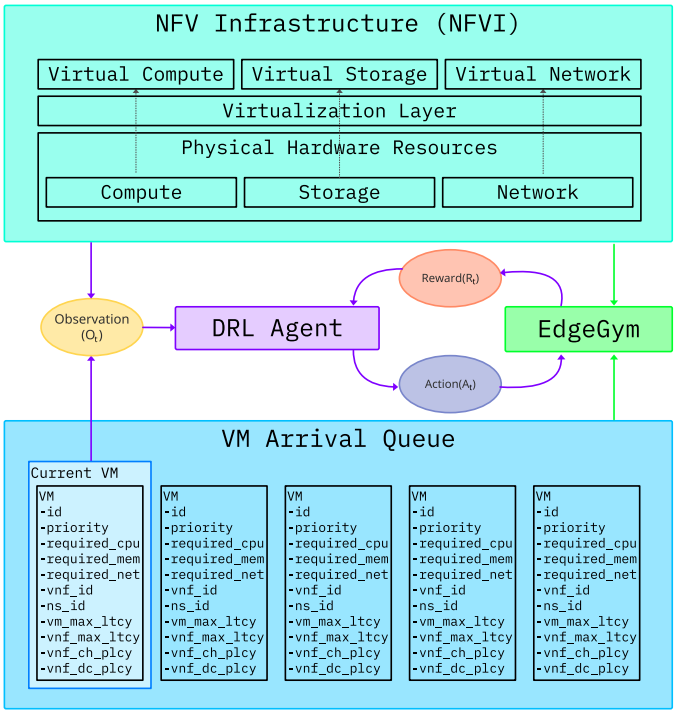


Fig. 3: The EdgeGym integrates both NFVI structure and VM arrival queue to generate agent observation.

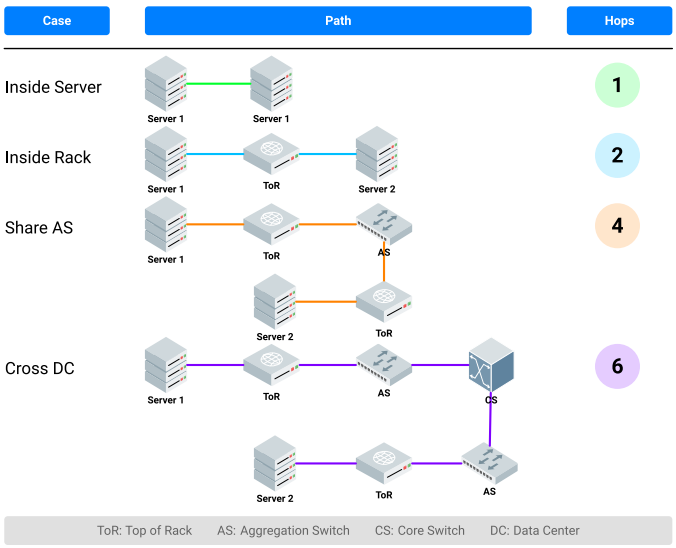


Fig. 4: Flow path and latency in terms of hops in different scenarios.

vnf_max_ltcy property in which all feasible VM placement results need to satisfy this inherited constraint.

One challenge in EdgeGym is efficiently determining the VM latency to calculate the rewards. Given the fact that the NFVI environment can be considered to be not constantly changing since it is a virtual mapping of physical hardware, we developed a mechanism to generate a server hops table for fast lookup at the environment initialization phase, as shown in Algorithm 1.

Algorithm 1: Server Hops Table Generation

Result: $server\_hops$, server hops table as 2-D tensor
Data: $servers$
$server\_hops \leftarrow []$;
forall $s1 \in servers$ **do**
  $hops \leftarrow []$;
  forall $s2 \in servers$ **do**
    if $s1.id = s2.id$ **then**
      $hops$.append(1)
    else if $s1.tor\_id = s2.tor\_id$ **then**
      $hops$.append(2)
    else if $s1.as\_conn$ & $s2.as\_conn$ **then**
      $hops$.append(4)
    else
      $hops$.append(6)
    end
  end
  $server\_hops$.append($hops$)
end

Each entity will be assigned a universally unique identifier attribute *id* and an entity-wise unique sequence number *seq* in the data preprocessing stage. The algorithm checks the equality of server id and rack id to determine *Inside Server* and *Inside Rack* status. Each server will have a N_{as} length binary bits array as_conn representing aggregation switch connection in numerical shape while N_{as} is the number of total aggregation switches in the environment. Each bit corresponds to the link with the aggregation switch of the corresponding sequence, and value 1 implies there is a link, while value 0 indicates no connection. By performing binary AND (&) operations on the as_conn properties of the two servers, we can quickly establish whether they are in a *Share AS* state. The calculation of the reward function can be significantly accelerated by lookup this server hops table.

V. Evaluation and Analysis

The EdgeGym provides a stochastic baseline agent while this agent will sample the actions from the masked-out action space to enforce the constraints. In our evaluation settings, we build a typical NFVI environment with two data centers while each data center has two aggregation switches and three racks. There are 18 servers with randomly generated configurations and connections seated on the racks. Table II summarizes the baseline agent evaluation settings. In the training process, the EdgeGym will generate random VM demand within the given

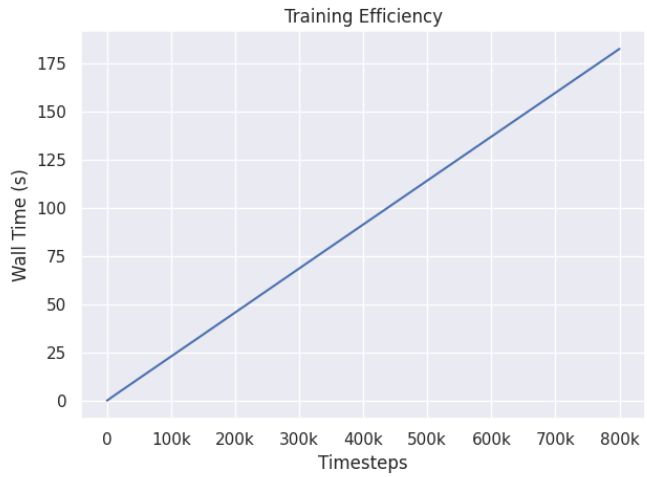
configuration range. The invalid action which violates the max latency and affinity constraints will be masked. Therefore all the resource allocation decisions will satisfy the implicit constraints.

TABLE II: Baseline Agent Evaluation Settings

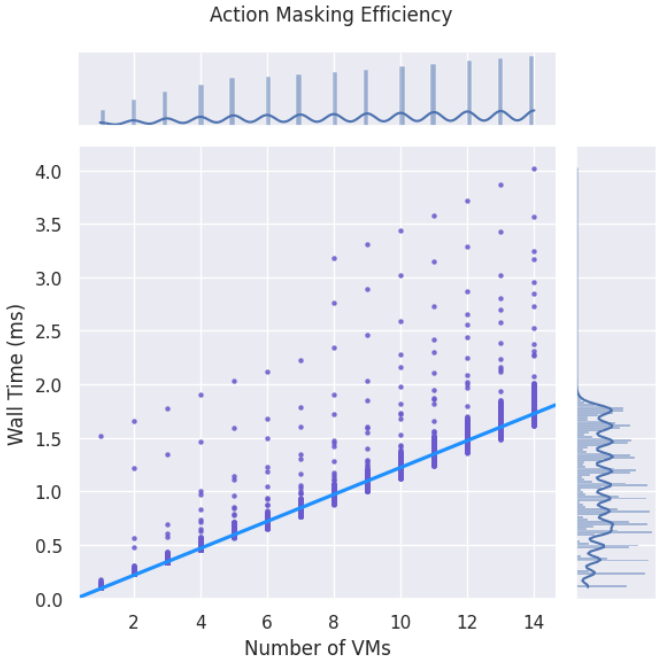
Parameter	Value
VIM Randomizer	
n_vim	3
n_ch_per_vim	6
ch_cpu_range_in_cores	$[32, 64]$
ch_mem_range_in_gb	$[128, 512]$
ch_net_range_in_gb	$[1, 10]$
VNF Randomizer	
n_min_vnf	2
n_max_vnf	10
n_standalone_vnf	-1
fixed_vnf_max_latency	6
n_min_vm_per_vnf	1
n_max_vm_per_vnf	10
vm_cpu_range_in_cores	$[1, 16]$
vm_mem_range_in_gb	$[4, 64]$
vm_net_range_in_gb	$[1, 2]$
fixed_vm_max_latency	6
fixed_ch_policy	0
fixed_dc_policy	0
Evaluation Settings	
total_timestep	$800,000$
log_timestep_interval	100

We evaluate the effectiveness in terms of training and action masking. After introducing the constraints, the agent training process should be in a reasonable time. We use timestep for the training efficiency evaluation. One timestep is the minimum unit of the agent making a resource allocation decision. The action masking complexity depends on the number of VMs since a more significant number of VMs also introduces massive constraints. Therefore we use the number of VMs for the action masking efficiency evaluation. Fig. 5 shows the evaluation result conducted on a single NVIDIA Tesla V100 GPU for $800,000$ timesteps.

From the results, the EdgeGym completed $800,000$ timesteps within 3 minutes, which shows it will not be a weakness for complicated agent training. The action masking is also instantaneous and can be completed in single-digit milliseconds.



(a) Training Efficiency: timesteps vs. wall time in second.



(b) Action Masking Efficiency: number of VMs vs. wall time in milliseconds.

Fig. 5: Evaluation results in terms of training and action masking efficiency.

VI. Conclusions and Future Work

This paper presents EdgeGym, a DRL framework to create reproducible NFV environments for optimizing resource allocation with reinforcement learning applications. It consists of an OpenAI Gym, a set of mechanisms to enforce constraints, and a stochastic agent for baseline purposes. The main aim of this work is to narrow the gap between reinforcement learning and NFV resource allocation research. We believe that an intelligent reinforcement learning agent can benefit the network operator in various aspects, such as energy-saving, performance boost, and latency reduction. We would also like to push the further research focus on RL agent optimization without ignorance of the constraints. We hope that EdgeGym can motivate researchers in these critical directions.

The ongoing and future works on EdgeGym will introduce Multi-Agent Reinforcement Learning (MARL) [29] integration. We expect MARL can open the possibilities for connecting different agent strategies. We are also interested in extending the baseline agents set to implement more popular algorithms like Proximal Policy Optimization (PPO) [30] and Advantage Actor Critic (A2C) [31] for metrics analysis. Lastly, we want to explore the integration with constraint programming for efficient and intelligent resource allocation decisions.

REFERENCES

[1] F. Bari, S. R. Chowdhury, R. Ahmed, R. Boutaba, and O. C. M. B. Duarte, "Orchestrating Virtualized Network Functions," *IEEE Transactions on Network and Service Management*, vol. 13, no. 4, pp. 725–739, Dec. 2016.

[2] T.-W. Kuo, B.-H. Liou, K. C.-J. Lin, and M.-J. Tsai, "Deploying chains of virtual network functions: On the relation between link and server usage," in *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, Apr. 2016, pp. 1–9.

[3] J. Liu, Y. Li, Y. Zhang, L. Su, and D. Jin, "Improve Service Chaining Performance with Optimized Middlebox Placement," *IEEE Transactions on Services Computing*, vol. 10, no. 4, pp. 560–573, Jul. 2017.

[4] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI Gym," *arXiv:1606.01540 [cs]*, Jun. 2016. [Online]. Available: http://arxiv.org/abs/1606.01540

[5] M. Ersue, "Etsi nfv management and orchestration-an overview," *Presentation at the IETF*, vol. 88, 2013.

[6] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De Turck, and R. Boutaba, "Network Function Virtualization: State-of-the-Art and Research Challenges," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 236–262, 2016. [Online]. Available: https://dx.doi.org/10.1109/comst.2015.2477041

[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, Oct. 2019.

[8] Y. Liu, J. Pei, P. Hong, and D. Li, "Cost-Efficient Virtual Network Function Placement and Traffic Steering," in *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, May 2019, pp. 1–6.

[9] O. Litvinski and A. Gherbi, "Openstack scheduler evaluation using design of experiment approach," in *16th IEEE International Symposium on Object/Component/Service-Oriented Real-time Distributed Computing (ISORC 2013)*, Jun. 2013, pp. 1–7.

[10] D. Li, P. Hong, K. Xue, and j. Pei, "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, Jul. 2018.

[11] D. Li, P. Hong, K. Xue, and J. Pei, "Virtual network function placement and resource optimization in NFV and edge computing enabled networks," *Computer Networks*, vol. 152, pp. 12–24, Apr. 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128618305000

[12] D. Qi, S. Shen, and G. Wang, "Towards an efficient VNF placement in network function virtualization," *Computer Communications*, vol. 138, pp. 81–89, Apr. 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0140366418308247

[13] T. Lin, Z. Zhou, M. Tornatore, and B. Mukherjee, "Demand-Aware Network Function Placement," *Journal of Lightwave Technology*, vol. 34, no. 11, pp. 2590–2600, Jun. 2016.

[14] M. Zeng, W. Fang, and Z. Zhu, "Orchestrating Tree-Type VNF Forwarding Graphs in Inter-DC Elastic Optical Networks," *Journal of Lightwave Technology*, vol. 34, no. 14, pp. 3330–3341, Jul. 2016.

[15] 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, Mar. 2019.

[16] H. Hawilo, M. Jammal, and A. Shami, "Network Function Virtualization-Aware Orchestrator for Service Function Chaining Placement in the Cloud," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 3, pp. 643–655, Mar. 2019.

[17] B. Addis, D. Belabed, M. Bouet, and S. Secci, "Virtual network functions placement and routing optimization," in *2015 IEEE 4th International Conference on Cloud Networking (CloudNet)*, Oct. 2015, pp. 171–177.

[18] S. Khebbache, M. Hadji, and D. Zeghlache, "Scalable and cost-efficient algorithms for VNF chaining and placement problem," in *2017 20th Conference on Innovations in Clouds, Internet and Networks (ICIN)*, Mar. 2017, pp. 92–99.

[19] L. Popokh, J. Su, S. Nair, and E. Olinick, "IllumiCore: Optimization Modeling and Implementation for Efficient VNF Placement," in *2021 International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, Sep. 2021, pp. 1–7.

[20] J.-B. Wang, J. Wang, Y. Wu, J.-Y. Wang, H. Zhu, M. Lin, M. Lin, and J. Wang, "A Machine Learning Framework for Resource Allocation Assisted by Cloud Computing," *IEEE Network*, vol. 32, no. 2, pp. 144–151, Apr. 2018.

[21] Y. Xiao, Q. Zhang, F. Liu, Jia Wang, J. Wang, Jia Wang, Miao Zhao, M. Zhao, Z. Zhang, and J. Zhang, "NFVdeep: Adaptive online service function chain deployment with deep reinforcement learning," *Proceedings of the International Symposium on Quality of Service*, pp. 1–10, Jun. 2019.

[22] J. Pei, P. Hong, M. Pan, J. Liu, and J. Zhou, "Optimal VNF Placement via Deep Reinforcement Learning in SDN/NFV-Enabled Networks," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 2, pp. 263–278, 2020. [Online]. Available: https://dx.doi.org/10.1109/jsac.2019.2959181

[23] 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*, vol. 25, no. 1, pp. 176–180, 2021. [Online]. Available: https://dx.doi.org/10.1109/lcomm.2020.3025298

[24] J. Su, S. Nair, and L. Popokh, "Optimal Resource Allocation in SDN/NFV-Enabled Networks via Deep Reinforcement Learning," in *2022 IEEE Ninth International Conference on Communications and Networking (ComNet)*, Nov. 2022, pp. 1–7.

[25] S. Huang and S. Ontañón, "A Closer Look at Invalid Action Masking in Policy Gradient Algorithms," *arXiv:2006.14171 [cs, stat]*, Jun. 2020. [Online]. Available: http://arxiv.org/abs/2006.14171

[26] R. BELLMAN, "A Markovian Decision Process," *Journal of Mathematics and Mechanics*, vol. 6, no. 5, pp. 679–684, 1957. [Online]. Available: https://www.jstor.org/stable/24900506

[27] F. Pezoa, J. L. Reutter, F. Suarez, M. Ugarte, and D. Vrgoč, "Foundations of JSON schema," in *Proceedings of the 25th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 2016, pp. 263–273.

[28] T. Preston-Werner and P. Gedam, "Toml v1.0.0," Jan 2021. [Online]. Available: https://toml.io/en/v1.0.0

[29] K. Zhang, Z. Yang, and T. Başar, "Multi-Agent Reinforcement Learning: A Selective Overview of Theories and Algorithms," in *Handbook of Reinforcement Learning and Control*, ser. Studies in Systems, Decision and Control, K. G. Vamvoudakis, Y. Wan, F. L. Lewis, and D. Cansever, Eds. Cham: Springer International Publishing, 2021, pp. 321–384. [Online]. Available: https://doi.org/10.1007/978-3-030-60990-0_12

[30] J. Schulman, Filip Wolski, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," *arXiv: Learning*, Jul. 2017.

[31] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Harley, T. P. Lillicrap, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ser. ICML'16. New York, NY, USA: JMLR.org, Jun. 2016, pp. 1928–1937.