DRL-FORCH: A Scalable Deep Reinforcement Learning-based Fog Computing Orchestrator

Nicola Di Cicco*, Gaetano Francesco Pittalà†, Gianluca Davoli†, Davide Borsatti†,
Walter Cerroni†, Carla Raffaelli†, Massimo Tornatore*
*Department of Electronics, Information, and Bioengineering (DEIB), Politecnico di Milano, Italy
†Department of Electrical, Electronic, and Information Engineering, University of Bologna, Italy

***Abstract*—We consider the problem of designing and training a neural network-based orchestrator for fog computing service deployment. Our goal is to train an orchestrator able to optimize diversified and competing QoS requirements, such as blocking probability and service delay, while potentially supporting thousands of fog nodes. To cope with said challenges, we implement our neural orchestrator as a Deep Set (DS) network operating on sets of fog nodes, and we leverage Deep Reinforcement Learning (DRL) with invalid action masking to find an optimal trade-off between competing objectives. Illustrative numerical results show that our Deep Set-based policy generalizes well to problem sizes (i.e., in terms of numbers of fog nodes) up to two orders of magnitude larger than the ones seen during the training phase, outperforming both greedy heuristics and traditional Multi-Layer Perceptron (MLP)-based DRL. In addition, inference times of our DS-based policy are up to an order of magnitude faster than an MLP, allowing for excellent scalability and near real-time online decision-making.**

***Index Terms*—Fog Computing, Reinforcement Learning, Orchestration, Optimization, Deep Learning**

I. Introduction

Fog Computing (FC) is a distributed computing paradigm that extends Cloud Computing (CC) by offloading tasks to devices situated in close proximity to the user, making use of their computing, storage, and communication resources [1]. FC can provide, among other benefits, lower latency for service access and fruition compared to CC. As FC is meant to handle heterogeneous devices with context-specific constraints (e.g., limited power, unstable connectivity, etc.), an orchestration layer, represented by an entity referred to as the fog orchestrator, is required to oversee the management of resources and the deployment of services. As such, a proper node selection policy is crucial in the effectiveness of FC systems, as multiple factors must be considered, including but not limited to the available computing power and the proximity of the service to the consumer.

Deep Reinforcement Learning (DRL) is a rapidly growing field that has the potential to revolutionize the way FC networks are optimized and controlled. Thanks to its capability of learning without explicit human supervision, DRL has found broad applications in networking, such as in traffic control [2], routing optimization [3], [4], and resource allocation [5], [6].

However, one severe limitation of traditional DRL-based approaches, often based on Multi-Layer Perceptron (MLP) neural networks, is their inability to deal with variable-sized input and output spaces. For instance, in the context of service orchestration in FC, the same DRL-based orchestrator must be applicable to clusters of fog nodes of arbitrary size, as the number of fog nodes is bound to change over time due to random connections/disconnections. Unfortunately, since the input and output dimensionalities of MLPs are defined on fixed-dimensional vector spaces, it is often impossible to generalize DRL-based approaches without retraining from scratch. These limitations may prevent the deployment of DRL-approaches in real application scenarios.

In this work, to cope with this fundamental challenge, we formulate the problem of online service orchestration in FC networks as a Machine Learning problem defined on sets. Indeed, we argue that a cluster of fog nodes is better represented as a set, i.e., as an arbitrary-size collection of distinct elements, rather than as a fixed-dimensional vector representation (as in traditional MLP-based approaches). Following this intuition, we leverage Deep Sets (DS) neural networks [7] to implement our DRL-based policy. DS are a family of neural networks tailored for processing data structured as sets. Once trained, DS networks can perform inference to sets of arbitrary size without the need for retraining, with a computational complexity linear in the number of elements in the input set. Therefore, integrating DS into FC networks allows for learning generalizable policies independently of the number of nodes available at a particular time in the network.

As a reference application scenario, we build upon the state-of-the-art FORCH orchestrator [8], a service orchestration system specifically designed for flexibility, which extends the Everything-as-a-Service (XaaS) model to fog computing environments. At each service request arrival, telemetry data together with service specifications are fed to a DRL-based policy, which selects the best node for serving the request. The choice of the nodes is driven by a properly-shaped reward signal, which in this work aims to find an optimal trade-off between blocking probability and requested service latency. Our illustrative numerical results on a simulation environment show that our DS-based policy is able to generalize its knowledge for numbers of fog nodes up to two orders of magnitude larger than training, outperforming both standard greedy heuristics and MLP-based policies. We, therefore, lay the foundations for implementing DRL-based intelligence in real orchestration software.

N. Di Cicco and G. F. Pittalà are co-first authors. Our source code is publicly available at https://github.com/bonsai-lab-polimi/netsoft2023-drl-forch.

II. Related Work

DRL, thanks to its capability of optimizing long-term objectives in dynamical systems, has found broad application in the context of resource allocation in FC networks. In this Section, we briefly survey recent literature on the topic and highlight our main contributions compared to the state-of-the-art.

Literature sharing the most similarity with our work considers the problem of resource allocation in FC networks. In [9], authors investigate the task of placing service containers in fog nodes given a list of service demands to minimize the number of deployed containers and optimize QoS metrics. An MLP-based Deep-Q Learning (DQN) agent is proposed, outperforming traditional optimization methods. In [10], authors consider the task of allocating service requests in a Fog Radio Access Network (F-RAN). An MLP-based DQN agent is proposed to optimize the utilization of edge resources, the utility of the served requests, and the acceptance probability. In [11], authors propose an intelligent Reinforcement Learning (RL)-based resources management at the network controller side, for sustainable Fog Radio Access Networks. Authors devise an MLP-based DQN to minimize the energy consumption in the network. In [12], authors propose three MLP-based DRL scheduling algorithms for the cloud-fog continuum, minimizing both the makespan and processing cost of workflows.

A second relevant body of literature considers the task offloading/migration/caching problem in FC networks. In [13], authors deal with the issues of content caching strategy, computation offloading policy, and radio resource allocation in Fog Computing networks, with the objective of minimizing the end-to-end delay. Authors develop an MLP-based DRL algorithm for solving the proposed optimization problem. In [14], authors focus their attention on offloading device-to-device issues in FC industrial applications. In particular, they make use of two different scheduling algorithms based on RL called Dynamic Reinforcement Learning Scheduling (DRLS) and Deep Dynamic Scheduling (DDS), showing how these two algorithms can drastically reduce energy costs compared to other offloading/non-offloading schemes. In [15], authors consider the problem of many-to-many task offloading in a dynamic vehicular environment. In particular, they adopt a Multi-Agent Gated actor Attention Critic (MA-GAC) approach, leading to an efficient offloading optimization process in a distributed manner. In [16], authors propose a decentralized task offloading method based on Transformer and Policy Decoupling-based Multi-Agent Actor-Critic (TPDMAAC), highlighting the flexibility of this algorithm, as it can be adapted to other scenarios by the fine-tuning of its parameters even with an uncertain load in the edge server. In [17], authors present a new component migration strategy in an NFV-based hybrid cloud/fog system considering the mobility of both end users and fog nodes. In particular, they propose a DRL approach, based on a Double Deep-Q Network (DDQN), to decide where and when to migrate application components, achieving better results in both delay and power consumption compared to other state-of-the-art application migration strategies. In [18], authors deal with the cooperative edge caching problem in F-RANs to minimize the content transmission delay. Authors propose a Multi Agent Reinforcement Learning (MARL)-based cooperative caching scheme, that applies a DDQN on each Fog Access Point (F-AP).

While all of the aforementioned works comprise significant advances in the application of DRL in FC environments, they do not address the problem of zero-shot generalization on unseen (and possibly more complex) deployment environments. Most state-of-the-art DRL approaches for FC make use of MLP deep neural networks for implementing their DRL policy. As MLPs operate on fixed-dimensional vector spaces, it is impossible to generalize their learned knowledge to different deployment scenarios without retraining. Multi-agent approaches such as in [15], [18] partially address the scalability issue of state/action spaces by distributing control over multiple learning agents. However, a distributed MARL-based approach is fundamentally incompatible with centralized service orchestration, and limitations in zero-shot generalization due to fixed input/output dimensionalities of MLPs still stand. In contrast, [16] explicitly considers the problem of fixed input/output dimensionality of MLPs, but their countermeasure still requires fine-tuning when applied to testing scenarios different than training. Overall, the intrinsic difficulty of zero-shot generalization makes deployment of DRL-based orchestrators in real scenarios problematic.

In this work, to cope with this scalability challenge, we consider online service orchestration in FC networks as a Machine Learning problem defined on sets of fog nodes. We make use of deep neural network architectures tailored for processing data structured as sets, enabling generalization to numbers of fog nodes of arbitrary size, possibly significantly larger than training. To illustrate our findings, we build on the FORCH [8] orchestrator, abstract its core functionalities into online decision-making, and train via DRL a DS neural network to optimize the process of service allocation.

III. Background

In this Section, we provide some background on the basics of DRL and on permutation-equivariant/invariant Deep Sets [7] neural networks, and we discuss how DS can be leveraged for learning DRL-based policies that can deal with variable state/action spaces without the need for retraining.

A. Deep Reinforcement Learning

Deep Reinforcement Learning (DRL) is a Machine Learning paradigm that targets sequential decision-making problems [19]. A DRL agent learns by trial-and-error via repeated interactions with a dynamic environment, with the objective of maximizing the accumulation of rewards. Specifically, a DRL environment is often modeled as a Markov Decision Process (MDP). An MDP consists in a tuple $\langle \mathcal{S}, \mathcal{A}, P, R, \gamma, \mu \rangle$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{P}$ is the transition probability matrix, R is the reward function, γ is the discount factor, and μ is the initial state distribution. The goal of DRL is to learn a neural network-based policy $\pi_\theta(a|s)$, $a \in \mathcal{A}$, $s \in \mathcal{S}$,

parameterized by θ, such that the discounted accumulation of rewards over a time-horizon T is maximized:

$$\theta = \arg\max_{\theta} J(\pi_\theta) = \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t R_t \,\middle|\, \pi_\theta\right] \tag{1}$$

Additionally, we define the discounted return at time step t as $G_t = \sum_{k=t+1}^{\infty} \gamma^{k-t-1} R_k$. Moreover, given a policy π, we define its value function $V_\pi(s) = \mathbb{E}_\pi[G_t | s_t = s]$.

While small-size MDPs can be solved to optimality with tabular RL methods, the state/action spaces of real-world MDPs may grow prohibitively large. In the context of FC, the resource occupation of fog nodes, which in its simplest form can be represented by a real number in $[0, 1]$, may take an infinite amount of possible values. As such, one needs to leverage function approximation via deep neural networks for solving large-size MDPs.

In the following, we will refer to Actor-Critic DRL algorithms, specifically to Proximal Policy Optimization (PPO) [20]. PPO is among the most widely used DRL algorithms thanks to its training efficiency and robustness to the choice of hyperparameters. Briefly, training Actor-Critic algorithms requires defining both a policy $\pi_\theta(a|s)$ and a value function approximator $V_\xi(s)$, parameterized by θ and ξ, respectively. Once the policy is trained, the value function approximator is discarded, and only the policy is kept for inference.

While traditional DRL-based approaches in FC employ MLP neural networks for implementing $\pi_\theta(a|s)$ and $V_\xi(s)$, the input/output spaces of said functions are limited by the dimensionality of state/action spaces in the training environment. For instance, if the dimensionality of the state/action space depends on the number of fog nodes, MLP networks cannot be applied without retraining on a different number of fog nodes, as in MLPs the dimensionality of the input space is hard-coded in the neural network architecture. In the following, we will tackle this problem by formulating the problem of resource allocation in FC as a problem defined on sets, and we will employ neural network architectures tailored for processing data structured as sets.

B. Deep Set Neural Networks

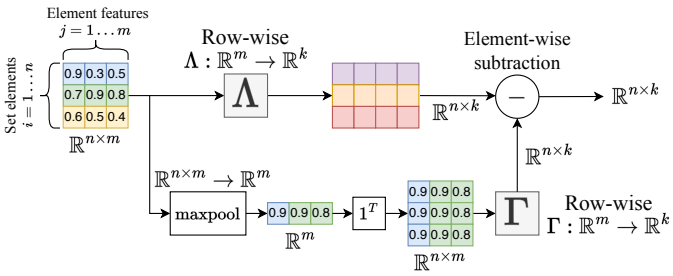
We now consider the case in which the input data to our neural network can be represented as a set. Formally, we consider our input as a set of n elements $X = \{x_1, \ldots, x_n\}$. Our goal is to design neural networks that are insensitive to the ordering of elements in the set. Formally, we would like neural networks that are either permutation-equivariant or permutation-invariant with respect to the ordering of the elements in the input sets. We define a function f to be permutation-invariant if $f(X) = f(p(X))$ for any permutation p, i.e., if the output of the function does not depend on the ordering of the input set elements. Similarly, we define a function f to be permutation-equivariant if $f(p(X)) = p(f(X))$ for any permutation p, i.e., if the permutation on the input set elements is reflected on the output.

In this work, we choose Deep Sets (DS) [7] as the blueprint for implementing permutation-equivariant and permutation-invariant deep neural networks. The properties of Deep Sets that make them an attractive choice for implementing scalable and intelligent fog orchestrators are: i) Deep Sets implement either permutation-equivariant or permutation-invariant neural networks, i.e., the learned functions are not bound to a specific indexing of the fog nodes; ii) the time complexity of Deep Sets scales linearly with the number of fog nodes, and computations can be trivially parallelized over different elements in the set; and iii) Deep Sets can infer on sets with an arbitrary number of elements, allowing generalization to different numbers of fog nodes. Formally, considering sets of n elements each one characterized by m features, Deep Sets realize a permutation-equivariant function $f(\mathbf{x})\colon \mathbb{R}^{n\times m} \to \mathbb{R}^{n\times k}$ as follows:

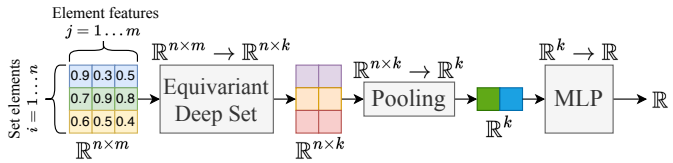
$$f(\mathbf{x}) = \sigma(\mathbf{x}\mathbf{\Lambda} - \mathbf{1}^{\mathsf{T}}\mathbf{\Gamma}\text{maxpool}(\mathbf{x})) \tag{2}$$

where $\mathbf{x} \in \mathbb{R}^{n\times m}$ are the input features, $\mathbf{\Lambda}, \mathbf{\Gamma} \in \mathbb{R}^{m\times k}$ are trainable parameters, and $\sigma(\cdot)$ is an element-wise nonlinearity (e.g., ReLU). The above equation realizes a permutation-equivariant function with respect to the rows of $\mathbf{x}$. The product $\mathbf{x}\mathbf{\Lambda}$ consists in applying the same linear transformation $\mathbf{\Lambda}$ to each row of $\mathbf{x}$. The function maxpool extracts the maximum of each column of $\mathbf{x}$, thus being permutation-invariant. One can therefore stack multiple layers in the form of Eq. (2) to build Deep Set permutation-equivariant neural networks. The update rule of Deep Sets is closely related to Message Passing Graph Neural Networks (MPNNs) [21]. Briefly, Deep Sets can be seen as MPNNs where at each message-passing iteration all nodes receive the same message. Since in this work we do not assume an adjacency model among fog nodes, we found Deep Sets more suitable to our needs than the more general MPNNs. The computational flow of permutation-equivariant Deep Sets is illustrated in Fig. 1a.

The computations in Eq. (2) do not assume a fixed value for n, i.e., the matrix multiplications appearing in Eq. (2) can be performed irrespectively from the size of the input set.



(a) Permutation-equivariant Deep Set.



(b) Permutation-invariant Deep Set.

Fig. 1. Computational flow of permutation-equivariant and permutation-invariant Deep Set neural networks.

As such, once a Deep Sets networks are trained, they can perform inference on input sets of arbitrary size. Moreover, their computational complexity is linear in the number of elements in the set, and computations can be parallelized over the elements in the set (i.e., the rows of $\mathbf{x}$) in a similar fashion to "batching" in deep neural networks.

In the context of Actor-Critic DRL algorithms, permutation-equivariant Deep Sets can be leveraged to implement a policy $\pi_\theta(a|s)\colon \mathbb{R}^{n\times m} \rightarrow \mathbb{R}^n$ for a discrete action space over the set elements. In particular, the state $s \in \mathbb{R}^{n\times m}$ represents a set of n elements each characterized by m features, and the action $a \in \mathbb{R}^n$ represents a categorical distribution over the n elements in the input set.

In a similar fashion, one can leverage permutation-invariant Deep Sets to implement the value function approximator $V_\xi(s)\colon \mathbb{R}^{n\times m} \rightarrow \mathbb{R}$. In this case, one requires permutation invariance as the output of $V_\xi(s)$ is a scalar. To realize permutation-invariant Deep Sets, one can apply a permutation-invariant pooling operator $pool\colon \mathbb{R}^{n\times m} \rightarrow \mathbb{R}^m$ to the final layer of an equivariant Deep Set network (e.g., max, sum or mean pooling). The pooling operator yields a fixed-size representation of an arbitrary-size set. Said representation can then be further transformed by a function $f\colon \mathbb{R}^m \rightarrow \mathbb{R}$ (e.g., a small MLP) to yield the desired output scalar. The computational flow of permutation-invariant Deep Sets is illustrated in Fig. 1b.

IV. DRL-FORCH: Deep Reinforcement Learning-based Fog Orchestrator

A. System Model

Our reference environment is the FORCH orchestration system [8], designed to provide services to users in a dynamic way, with a focus on resource utilization efficiency and service fruition latency. In principle, the orchestrator listens for service requests from the users and attempts to activate said services on the available fog nodes. In this context, one of the key issues is how to properly select the node on which to deploy the service. This problem is made even more delicate by the nature of the fog environment, where available nodes may change over time due to current nodes disconnecting and new ones joining. Therefore, in the context of DRL for service orchestration in FC scenarios, designing a generalized policy that is able to handle the variable number of fog nodes is a crucial research challenge.

The main entities in the DRL-FORCH system architecture are illustrated in Figure 2. Our proposed DRL-FORCH orchestrator leverages retrieved telemetry data from the fog network to perform online decision-making. In the current implementation, FORCH employs Prometheus [22], paired with a simple Python-based system monitoring routine, to collect and store information on the resource utilization of the fog nodes, including usage data on CPU, RAM, disk, and network interfaces.

Formally, we consider an orchestrator managing a set V of fog nodes. Service requests from users arrive randomly, upon which the orchestrator chooses either to serve them in one of

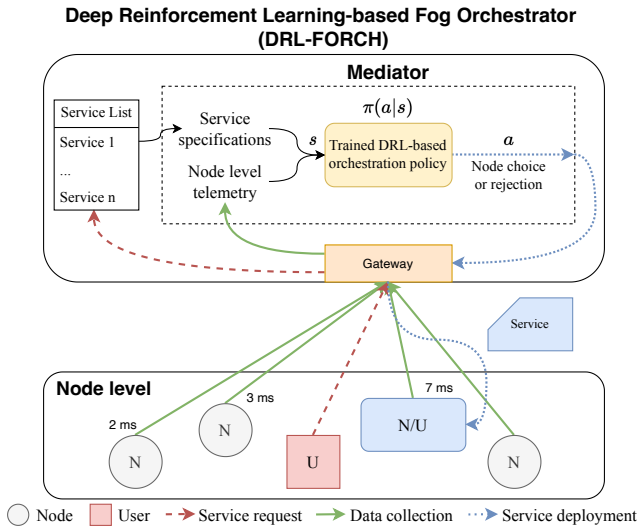


Fig. 2. Service deployment decision making in DRL-FORCH.

the available fog nodes or to reject them (e.g., by blocking the service request or by sending it to a cloud infrastructure). The FORCH system maintains an index of all the services it supports, including information needed to refer to each service in the orchestration context (name, identifier, service type), as well as to activate it (virtualization technology-specific details). Each request specifies the service the user needs, which is associated with a specific service type. In FORCH, services are classified into different types based on their deployment model, within the XaaS paradigm. Specifically, with references to the FORCH architecture introduced in [8] and extended in [23], we consider the following types of service:

- Fog Virtualization Engine (FVE), instantiated according to the Infrastructure-as-a-Service (IaaS) model. It enables the deployment of a variety of computing environments on top of a generic virtualization layer available on a fog node (e.g., Docker).
- Software Development Platform (SDP), instantiated according to the Platform-as-a-Service (PaaS) model. It allows the node to provide the user with a set of tools, including libraries, platforms, or interpreters to develop and run generic applications (e.g., Python SDK).
- Application (APP), instantiated according to the Software-as-a-Service (SaaS) model. It lets a fog node host a specific application that can be accessed by consumers through a specific interface (e.g., a Web-based application).
- Lightweight Atomic Functions (LAF), instantiated according to the Function-as-a-Service (FaaS) model. Based on the event-driven serverless computing execution model, this service is fully managed by the fog node and activated at need (e.g., real-time multimedia transcoding).

We presume that nodes that make their resources available to the fog orchestration system support at least one of the services that FORCH recognizes. Nodes offering SDP, APP,

and/or LAF services will only be able to be employed to serve requests for the specific type of service they offer. Conversely, we expect that nodes able to instantiate FVE services (i.e., nodes capable of deploying virtualized platforms) can instantiate any other type of services, provided a suitable source (e.g., a container image) exists for them.

Generally, we assume FaaS and SaaS elements to be the ones with tighter requirements on latency and softer needs in terms of computational power, while PaaS and IaaS elements to be less strict in terms of latency requirements but more demanding in terms of computational power. We argue that this is a sensible rationale, as the former two services types are generally associated with on-demand fast service provisioning, reflected in their looser requirements in terms of computational power, and their tighter requirements regarding latency times; on the opposite, the latter two service types usually correspond to a need for more computation power at the expense of service latency.

In our system model, we view fog nodes as heterogeneous devices with inherent limits on computational resources [24]. Specifically, we consider two classes of fog nodes: fixed nodes and mobile nodes. Fixed nodes are specialized nodes located at the edge of the local infrastructure, that assist the orchestrator whenever no suitable mobile node can be selected for the provisioning of services. As such, they possess vast computational resources, and provide services with lower latency times compared to the mobile ones. Mobile nodes, instead, represent devices that make their resources available to the orchestrator for a limited time, generally not known a priori, as they are reachable only when connected to the same local network of the orchestrator, before disconnecting, either logically or physically. As such, those nodes are likely to offer fewer computational resources than fixed nodes, and are generally expected to exhibit less predictable latency performance. However, mobile nodes are expected to provide valuable support in handling services that have softer constraints in terms of latency and required computational power, allowing the orchestrator to reserve fixed nodes for more demanding tasks, thereby improving resource usage efficiency.

In this work, we assume that resource occupation in terms of CPU, RAM, disk, and bandwidth of fog nodes is represented as normalized values in $[0, 1]$. We assume that fog nodes are available if their resource occupation does not exceed 0.95 and 0.5 for fixed and mobile nodes, respectively, i.e., we assume that mobile nodes have approximately half the overall capacity of fixed nodes in terms of CPU, RAM, disk, and bandwidth.

Furthermore, we make some assumptions regarding the service latency that each fog node can provide and those required by each service class. In this work, we assume that each service belongs to a "latency class" that goes from 1 to 10, where class 1 includes services with the tightest requirements on latency, and 10 denotes best-effort. Each fog node in the system is also associated with a latency class, representing the lowest latency value the node can support (i.e., the services it can deploy satisfying their requirements on latency). Specifically, we randomly assign a latency class in the range from 5 to 10 for mobile nodes and from 1 to 5 for fixed ones, respectively, consistently with the above rationale. This information is leveraged in the process of choosing on which node to activate a requested service.

Finally, we assume that, for each service FORCH supports, upper bounds on the requested computational resources are known a priori by the orchestrator.

B. Formulating Orchestration in Fog Computing as an MDP

To train a DRL-based orchestrator for FC networks, we must provide an MDP formulation for our FC environment. In particular, we need to provide suitable definitions for the state space $\mathcal{S}$, the action space $\mathcal{A}$, and the reward function R. Crucially, we need to engineer our state and action spaces to be compatible with our Deep Sets-based permutation-equivariant policy and permutation-invariant value function neural networks.

Action Space: we consider a discrete action space of dimension $|V| + 1$. Given a service request, the orchestrator either allocates it to one of the $|V|$ fog nodes or rejects it. Rejection can be due to a lack of computational resources in the fog nodes, or proactive, i.e., rejecting a service request with the long-term goal of serving more in the future. Rejection can be intended as offloading the service request to the cloud, in which the availability of computational resources is assumed to be unconstrained.

We note that the action space grows linearly with $|V|$, which may pose scalability issues for large numbers of fog nodes. Still, not all actions may be simultaneously permitted given a specific state. Indeed, a fog node may not have enough computational resources to satisfy a service request or may not host the appropriate containerized service. A simple solution found in previous literature [9], [10] is to issue negative rewards (i.e., penalties) whenever an invalid action is chosen. Even though penalties discourage the agent from choosing invalid actions, it is well-known that in DRL sparse (i.e., infrequent), large rewards are detrimental to convergence. As invalid actions are a priori known by the orchestrator, we leverage *invalid action masking* in policy gradient algorithms, which has been found to yield significantly better performance and sample efficiency than invalid action penalties [25]–[27]. Action masking works by setting the log-probabilities of invalid actions to $-\infty$ before sampling an action, according to a state-dependant action mask. As such, we ensure that our orchestrator never chooses invalid actions. Formally, we define the action mask of node $i \in \{0, \dots, |V|\}$ at state s as follows:

$$\text{mask}(s)[i] = \begin{cases} \textit{true} & \text{if } i \text{ has enough resources and hosts} \\ & \text{the appropriate service container} \\ \textit{false} & \text{otherwise} \end{cases} \tag{3}$$

Whereas for action $|V| + 1$, i.e., rejection, the action mask is always set to *true*. The possibility of rejection ensures that in all states, at least one action is permitted. In this way, we exclude the possibility that the agent may "lock itself" in states where all actions are invalid.

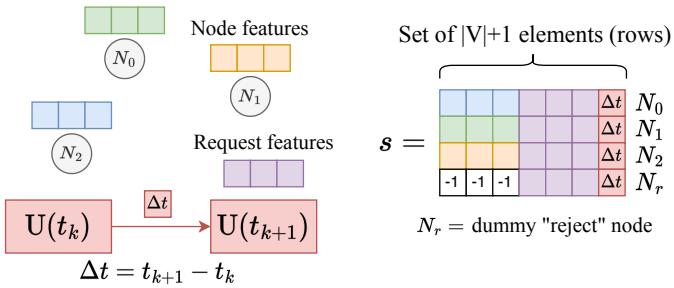


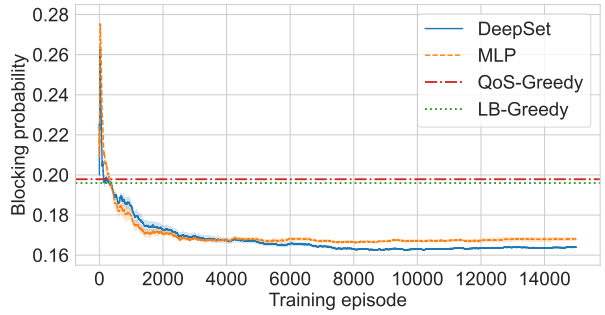
Fig. 3. Building a state for DRL-FORCH. The state s consists of a set of $|V|+1$ elements, each element having a fixed number of features m.

State Space: we consider a state space of dimension $(|V|+1)\times m$, where m is the number of features per node. For each fog node, we choose the normalized CPU, RAM, disk, bandwidth utilization, and its measured delay as features. As our Deep Set-based policy returns a categorical probability distribution over the set elements (i.e., choosing a fog node), we must also introduce a "dummy" node representing the reject action. Therefore, we include an additional "reject" node with all features set to -1. Furthermore, we consider request-specific features, namely, the CPU, RAM, disk, and bandwidth required by the current request. Finally, since the orchestrator is invoked at each service request arrival, the time interval Δt between two consecutive environment steps is not constant. Therefore, we also consider Δt as an input feature. The inclusion of Δt allows the agent to learn the traffic dynamics, as it explicitly conveys the inter-arrival time between subsequent requests, and implicitly the service duration time (by looking at the difference in resource occupation in fog nodes between two consecutive states). The request-specific features and Δt are replicated $n+1$ times and concatenated to the node-specific features. Fig. 3 illustrates how the set-based state representation is built.

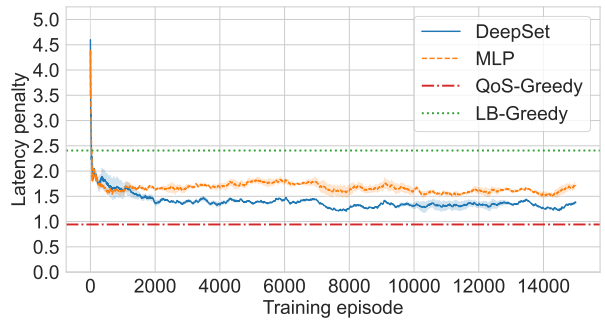
Reward Function: we consider a bi-objective optimization problem, such that i) the average blocking probability is minimized, and ii) the latency QoS requirements of each service are satisfied as much as possible. As such, upon either choosing a fog node or rejecting a service request, our reward function is shaped as follows:

$$r_t = \begin{cases} 1 & \text{if accepted and } L_{\text{node}} \leq L_{\text{service}} \\ -\frac{L_{\text{node}} - L_{\text{service}}}{L_{\text{node}}} & \text{if accepted and } L_{\text{node}} > L_{\text{service}} \\ -1 & \text{if rejected} \end{cases} \quad (4)$$

where L_{node} is the maximum latency class satisfiable by the chosen fog node, and L_{service} is the latency class required by the orchestrated service. When $L_{\text{node}} > L_{\text{service}}$, we define the latency penalty $L_{\text{pen}} = L_{\text{node}} - L_{\text{service}}$ as a measure of the QoS degradation. As such, the reward function is shaped such that it penalizes both i) blocking and ii) violation of latency service requirements. We note that, our reward shaping strategy is akin to assigning different objective priorities in multi-objective optimization, which depend on the specific application scenario [28]–[30]. In our case, we normalize the latency penalty such that the associated reward is always greater than -1, meaning that rejecting a service request is always considered more undesirable than accepting it with a degradation in the latency requirements.



(a) Training blocking probability.



(b) Training latency penalty

Fig. 4. Training blocking probability and QoS metrics for PPO-DeepSet and the considered greedy heuristics and DRL-based baselines.

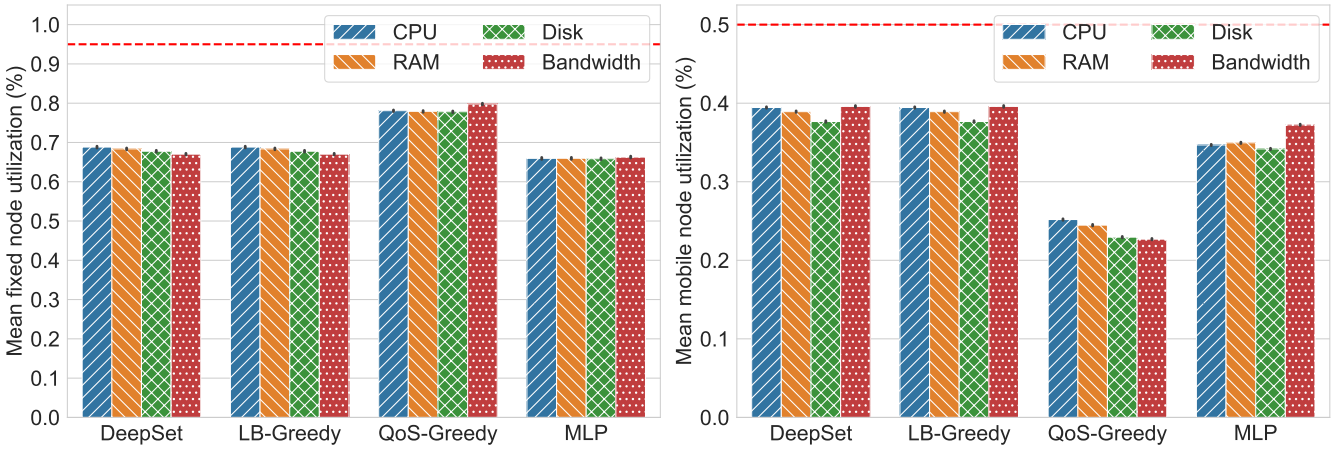
We underline that objectives i) and ii) compete with each other. For instance, the blocking probability can be minimized by a deterministic load-balancing policy that distributes the requests evenly among all the available fog nodes, but this would very likely break the QoS requirements of the most latency-constrained service requests. Conversely, one may ensure QoS satisfaction by greedily filling the fixed nodes first, but failing to exploit the additional capacity provided by mobile modes will lead to larger blocking probabilities. Therefore, learning an online orchestration policy that finds a good long-term trade-off between load balancing and latency constraints is not a trivial optimization task.

V. Illustrative Numerical Results

The goal of our illustrative numerical results is to empirically show that our Deep Set-based orchestrator can:

1) outperform standard greedy heuristic approaches in terms of QoS metrics optimization;
2) match or outperform a traditional MLP-based orchestrator with inference times up to an order of magnitude faster and lower memory occupation;
3) generalize its learned policy without the need for re-training (i.e., zero-shot generalization) to numbers of fog nodes up to two orders of magnitude larger than training.

In our simulation, for each service request, we assume the demanded computational resources (CPU, RAM, disk, and bandwidth) to be uniformly distributed in $[0.15, 0.3]$, $[0.1, 0.2]$,



(a) Average resource utilization on fixed fog nodes.

(b) Average resource utilization on mobile nodes.

Fig. 5. Average resource utilization of fixed and mobile nodes for $|V| = 100$ fog nodes, mean arrival rate $\lambda = 1000$, and mean service duration $\mu = 1$. The dashed line indicates the 0.95 and 0.5 saturation thresholds for fixed and mobile fog nodes, respectively.

$[0.01, 0.1]$, and $[0.01, 0.02]$ for FVE, SDP, APP, and LAF service types, respectively. Similarly, we assume the requested latency classes to be uniformly distributed in $[4, 10]$, $[3, 7]$, $[1, 4]$, and $[1, 4]$ for FVE, SDP, APP, and LAF service types, respectively. Our training and evaluation protocol is structured as follows.

Training: we train on a small set of 10 fog nodes, with a fixed amount of offered traffic equal to 100 service requests. We used PPO for training our DRL-based policy. Specifically, we consider an episodic learning setting with episode length equal to 100, i.e., where the policy and the value function networks are updated every 100 request arrivals. Empirically, choosing an episode length equal to 100 provided the best trade-off between the convergence times of the policy network and the long-term optimization of our objective function. For implementing PPO, we borrowed from CleanRL [31] and Stable-Baselines3 [32]. Training times were in the order of ten minutes on a Macbook Pro M1 CPU.

Evaluation: we evaluate on environments with numbers of fog nodes $|V| = 50$, 100, 500 and 1000, with a mean arrival rate $\lambda = 500$, 1000, 5000 and 10000 time-units, respectively, and a mean service duration $\mu = 1$ time-units for all cases. Thus, we evaluate zero-shot generalization of DS on instances up to two orders of magnitude larger than training.

Baselines: we compare our DS-based orchestrator with the following baseline orchestration policies:

- **LB-Greedy**: assigns the service request to the available node with the lowest current resource occupation. This policy aims at minimizing the blocking probability, at the expense of the latency penalties.
- **QoS-Greedy**: assigns the service request to the available node with the lowest measured service delay. This policy aims at minimizing the latency penalties, at the expense of the blocking probability.
- **MLP**: DRL-based orchestrator optimizing the same reward function as DS, but using an MLP neural network to implement the policy and the value networks. In contrast to DS this baseline is evaluated on the same settings in which it was trained, due to the fixed input/output dimensionalities of MLPs.

We consider average blocking probability and latency penalty (as defined in (4) as performance metrics. For DS and MLP, numerical results are aggregated over five different random training seeds.

Numerical Results: Fig. 4 illustrates the evolution during training of blocking probability and latency penalty of DS and MLP, alongside the average values of QoS-Greedy and LB-Greedy. DS attains approximately 3% lower blocking probabilities than the greedy heuristic baselines, achieving a relative 15% improvement. Moreover, the average latency penalty of DS stands lower than 1.5, which is only 0.5 away from the QoS-Greedy policy. Overall, DS achieves the best trade-off between blocking probability and latency penalty, illustrating the capability of DRL to efficiently learn an effective online optimization strategy for both QoS metrics. Moreover, while DS and MLP learn similar policies, we observe that the policy learned by DS is slightly better than MLP for both the considered metrics. This result illustrates that the choice of a DS-based policy for orchestration in FC networks is indeed more desirable than a traditional MLP. Furthermore, other than outperforming an MLP, DS can be applied to an arbitrary number of fog nodes without the need for retraining.

Fig. 5 illustrates the resource occupation (CPU, RAM, disk, and bandwidth) of DS and the considered baselines in the training environment. We observe that, as expected, QoS-Greedy tends to favor fixed nodes, as they provide on average lower latency. Conversely, LB-Greedy tends to distribute the load between mobile and fixed nodes evenly, irrespective of the latency requirements. As illustrated in Fig. 4, both greedy approaches fail to provide a competitive blocking probability, as they do not take into account the dynamics of the traffic and cannot issue tactical proactive rejections. DS and MLP, on

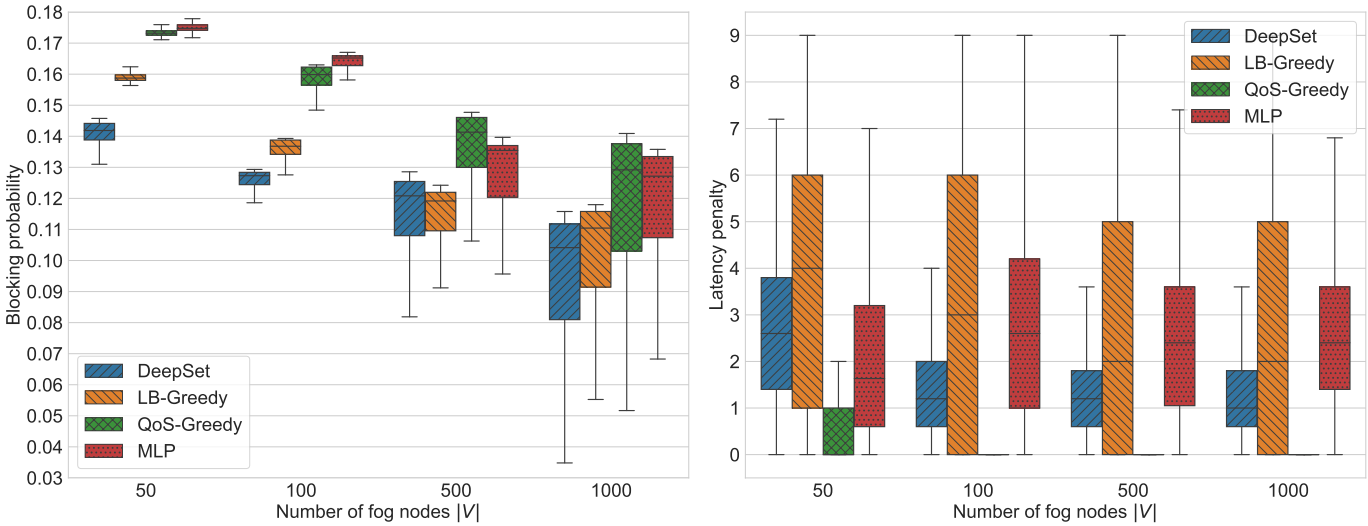


Fig. 6. Test blocking probabilities and latency penalties of DeepSet and the considered greedy heuristics and DRL-based baselines.

the other hand, make smart use of the available mobile nodes (without getting too close to overloading them), allowing them to achieve better resource utilization than both greedy heuristics. We note that the occupation of both fixed and mobile nodes is relatively far from their 0.95 and 0.50 threshold, respectively. This suggests that blocking mainly happens due to the unavailability of nodes hosting a specific containerized service rather than the unavailability of raw computational resources. This consideration reinforces the need for smart resource allocation policies in FC networks.

Fig. 6 illustrates the blocking probabilities and latency penalties of DS, MLP and the greedy baselines for test environments with $|V| = 50$, 100, 500, and 1000 fog nodes, with mean inter-arrival rates equal to $\lambda = 500$, 1000, 5000, and 10000 time-units, respectively, and service duration equal to $\mu = 1$ time-unit for all cases. We underline that for DS the same policy trained on $|V| = 10$ is reused in a zero-shot fashion, whereas for MLP a new policy must be trained from scratch due to the different dimensionalities of the input spaces, with training times spanning several hours for $|V| = 1000$. In particular, due to the growing complexity of the input space, MLP is unable to learn a competitive policy for large numbers of fog nodes. Conversely, DS extrapolates well its learned policy without the need for retraining, attaining the best trade-off between blocking probabilities and latency penalty up to $|V| = 1000$. In particular, up to $|V| = 100$, DS significantly outperforms all baselines in terms of blocking probability, and achieves latency penalties slightly worse than QoS-Greedy. For $|V| = 500$, 1000 LB-Greedy closes the gap with DS in terms of blocking probability, but DS achieves significantly lower latency penalties. We conclude that DS learned an effective and generalizable orchestration policy, achieving a smart trade-off between load balancing and satisfying QoS latency requirements.

Fig. 7 illustrates the blocking probability of DS and the other baselines for $|V| = 100$, $\mu = 1$ while varying λ. We can observe that both DS and MLP generalize well

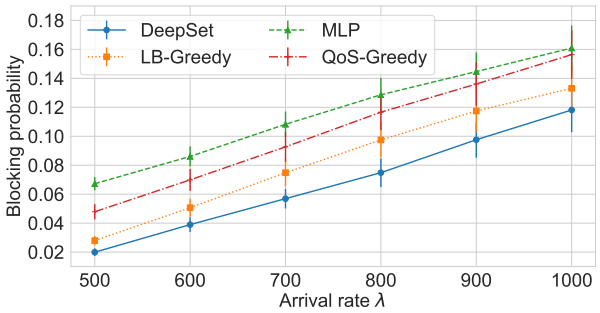


Fig. 7. Blocking probability of DeepSet and the considered baselines for $|V| = 100$ fog nodes, varying the service arrival rate λ.

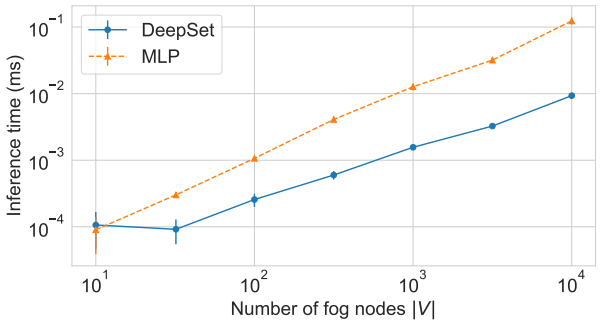


Fig. 8. Inference times (ms) of DeepSet and MLP with two hidden layers of size 512, for a varying number of fog nodes and 100 features per node.

with respect to the amount of offered traffic, with blocking probabilities scaling linearly as expected. Consistently with previous results, DS outperforms all other baselines in terms of blocking probability.

We conclude our scalability analysis by evaluating the inference times of the proposed Deep Set-based orchestrator for large numbers of fog nodes and features per node. Fig. 8 illustrates the inference times of DS-based and MLP-based policies with two hidden layers of dimension 512 and 100 features per fog node. Inference times were measured on a Macbook Pro M1 CPU. While DS and MLP show similar

inference times for smaller numbers of fog nodes, DS scales significantly better, with inference times approximately equal to 0.01ms for 10^4 fog nodes, an improvement of an order of magnitude compared to MLP. This is expected, as the size of MLP grows with the number of fog nodes, while the size of DS is independent of the number of fog nodes. As such, our DS-based policy provides excellent scalability in both time and memory complexity.

VI. Conclusion

In this work, we proposed DRL-FORCH, a Deep Reinforcement Learning-based orchestrator for Fog Computing networks built on top of the state-of-the-art FORCH orchestration system. To deal with the limitations of standard DRL-based solution for dealing with variable numbers of fog nodes, we engineered the online orchestration problem as a Machine Learning problem on sets, and we proposed the use of Deep Sets neural networks for implementing our DRL-based orchestration policy. Our illustrative numerical results demonstrate that our Deep Sets-based policy yields excellent zero-shot generalization for numbers of fog nodes up to two orders of magnitude larger than training, providing the best trade-off between blocking probability and satisfaction of service latency constraints. As future work, we plan on integrating a realistic fog node energy consumption model in the DRL reward function, and evaluating the performance of our DRL agent in large-scale smart city environments.

Acknowledgments

This work was partially supported by the European Union under the Italian National Recovery and Resilience Plan (NRRP) of NextGenerationEU, partnership on "Telecommunications of the Future" (PE00000001 - program "RESTART").

References

[1] F. Bonomi *et al.*, "Fog computing and its role in the internet of things," in *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, ser. MCC '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 13–16.

[2] Z. M. Fadlullah *et al.*, "State-of-the-art deep learning: Evolving machine intelligence toward tomorrow's intelligent network traffic control systems," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2432–2455, 2017.

[3] G. Stampa *et al.*, "A deep-reinforcement learning approach for software-defined networking routing optimization," *arXiv preprint arXiv:1709.07080*, 2017.

[4] N. Di Cicco *et al.*, "On deep reinforcement learning for static routing and wavelength assignment," *IEEE Journal of Selected Topics in Quantum Electronics*, vol. 28, no. 4: Mach. Learn. in Photon. Commun. and Meas. Syst., pp. 1–12, 2022.

[5] X. Liu *et al.*, "Resource allocation for edge computing in iot networks via reinforcement learning," in *ICC 2019-2019 IEEE international conference on communications (ICC)*. IEEE, 2019, pp. 1–6.

[6] M. R. Raza *et al.*, "Reinforcement learning for slicing in a 5g flexible ran," *J. Lightwave Tech.*, vol. 37, no. 20, pp. 5161–5169, 2019.

[7] M. Zaheer *et al.*, "Deep sets," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17, 2017, p. 3394–3404.

[8] G. Davoli *et al.*, "A fog computing orchestrator architecture with service model awareness," *IEEE Transactions on Network and Service Management*, vol. 19, no. 3, pp. 2131–2147, 2022.

[9] H. Sami *et al.*, "Demand-driven deep reinforcement learning for scalable fog and service placement," *IEEE Transactions on Services Computing*, vol. 15, no. 5, pp. 2671–2684, 2022.

[10] A. Nassar and Y. Yilmaz, "Deep reinforcement learning for adaptive network slicing in 5g for intelligent vehicular systems and smart cities," *IEEE Internet of Things Journal*, vol. 9, no. 1, pp. 222–235, 2022.

[11] Y. Sun *et al.*, "Deep reinforcement learning-based mode selection and resource management for green fog radio access networks," *IEEE IoT Journal*, vol. 6, no. 2, pp. 1960–1971, 2018.

[12] J. C. Guevara *et al.*, "Qos-aware task scheduling based on reinforcement learning for the cloud-fog continuum," in *GLOBECOM 2022-2022 IEEE Global Communications Conference*. IEEE, 2022, pp. 2328–2333.

[13] Y. Wei *et al.*, "Joint optimization of caching, computing, and radio resources for fog-enabled iot using natural actor–critic deep reinforcement learning," *IEEE IoT Journal*, vol. 6, no. 2, pp. 2061–2073, 2018.

[14] Y. Wang *et al.*, "Traffic and computation co-offloading with reinforcement learning in fog computing for industrial applications," *IEEE Trans. Ind. Infor.*, vol. 15, no. 2, pp. 976–986, 2018.

[15] Z. Wei *et al.*, "Dynamic many-to-many task offloading in vehicular fog computing: A multi-agent drl approach," in *GLOBECOM 2022-2022 IEEE Global Communications Conference*. IEEE, 2022, pp. 6301–6306.

[16] Z. Gao *et al.*, "Fast adaptive task offloading and resource allocation via multi-agent reinforcement learning in heterogeneous vehicular fog computing," *IEEE Internet of Things Journal*, 2022.

[17] S. N. Afrasiabi *et al.*, "Reinforcement learning-based optimization framework for application component migration in nfv cloud-fog environments," *IEEE Transactions on Network and Service Management*, 2022.

[18] Q. Chang *et al.*, "Cooperative edge caching via multi agent reinforcement learning in fog radio access networks," in *ICC 2022-IEEE International Conference on Communications*. IEEE, 2022, pp. 3641–3646.

[19] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. MIT Press, 2018.

[20] J. Schulman *et al.*, "Proximal policy optimization algorithms," 2017. [Online]. Available: https://arxiv.org/abs/1707.06347

[21] J. Gilmer *et al.*, "Neural message passing for quantum chemistry," in *Proceedings of the 34th International Conference on Machine Learning (ICML 2017)*, 2017.

[22] Prometheus. [Online]. Available: https://prometheus.io/

[23] G. F. Pittalà *et al.*, "Function-as-a-service orchestration in fog computing environments," in *2022 18th International Conference on Network and Service Management (CNSM)*. IEEE, 2022, pp. 364–366.

[24] B. Costa *et al.*, "Orchestration in fog computing: A comprehensive survey," *ACM Computing Surveys*, vol. 55, no. 2, pp. 1–34, 2022.

[25] O. Vinyals *et al.*, "Starcraft ii: A new challenge for reinforcement learning," 2017. [Online]. Available: https://arxiv.org/abs/1708.04782

[26] S. Huang and S. Ontañón, "A closer look at invalid action masking in policy gradient algorithms," in *Proceedings of the Thirty-Fifth International Florida Artificial Intelligence Research Society Conference, FLAIRS 2022*, 2022.

[27] J. W. Nevin *et al.*, "Techniques for applying reinforcement learning to routing and wavelength assignment problems in optical fiber communication networks," *J. Opt. Commun. Netw.*, vol. 14, no. 9, pp. 733–748, Sep 2022.

[28] N. Di Cicco *et al.*, "Optimization over time of reliable 5g-ran with network function migrations," *Computer Networks*, vol. 215, p. 109216, 2022.

[29] N. H. Thanh *et al.*, "Energy-aware service function chain embedding in edge–cloud environments for iot applications," *IEEE IoT Journal*, vol. 8, no. 17, pp. 13 465–13 486, 2021.

[30] G. Sun *et al.*, "Energy-efficient provisioning for service function chains to support delay-sensitive applications in network function virtualization," *IEEE IoT Journal*, vol. 7, no. 7, pp. 6116–6131, 2020.

[31] S. Huang *et al.*, "Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms," *Journal of Machine Learning Research*, vol. 23, no. 274, pp. 1–18, 2022.

[32] A. Raffin *et al.*, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021.