Reinforcement Learning-Driven Service Placement in 6G Networks across the Compute Continuum

Andres F. Ocampo*‡, José Santos†‡,
* SimulaMet – Simula Metropolitan Center for Digital Engineering, Pilestredet 52, N-0167 Oslo, Norway
Email: {andres}@simula.no
† IDlab, Department of Information Technology at Ghent University - imec, 9000 Ghent, Belgium
Email: {josepedro.pereiradossantos}@UGent.be

***Abstract*—The advent of 6G networks promises unprecedented advancements in communication technologies, demanding innovative solutions for service placement across the Compute Continuum (CC), where computing resources are distributed across the network area, from edge to cloud. This paper explores a novel approach for service placement in 6G networks using Reinforcement Learning (RL) techniques. By leveraging the dynamic decision-making capabilities of RL, this work addresses the complexities of distributing services across heterogeneous computing resources to enhance network performance, reduce latency, and improve resource utilization. An extensive evaluation, based on a real dataset collected from commercial 4G/5G networks, was conducted under various network conditions and workloads to evaluate the effectiveness of the proposed approach. Results highlight the adaptability of our RL-driven model to dynamic network environments, demonstrated by its capacity to optimize for multiple objectives simultaneously. Our analysis also reveals that while single-objective heuristics can outperform RL in specific, limited scenarios, these struggle to handle increasing complexity. These findings highlight the viability of RL as a powerful tool for intelligent service management in next-generation communication systems, paving the way for more resilient and efficient 6G network architectures.**

***Index Terms*—Compute Continuum, Reinforcement Learning, Orchestration, Service Placement, 6G, Cloud Computing**

I. Introduction

The emergence of the Compute Continuum (CC), fueled by the demands of 6G and real-time applications, presents a paradigm shift in cloud computing [1], [2]. The CC envisions a seamless integration of diverse computing resources, from powerful cloud Data Centers (DCs) to resource-constrained edge devices, to execute applications closer to users and minimize latency. However, this heterogeneity and the dynamic nature of resource availability in the CC pose significant challenges for service placement and resource orchestration in 6G networks [3], [4].

Traditional approaches, including rule-based policies and heuristic algorithms, struggle to keep pace with the dynamic conditions of the CC [4], [5]. Rules quickly become outdated as network conditions change, and heuristics, while computationally efficient, often fail to capture the complexities of real-world deployments. This leads to suboptimal service placement and inefficient resource utilization, hindering the CC's full potential.

Reinforcement Learning (RL) represents a promising approach to overcome these limitations in dynamic service placement within the CC as demonstrated by recent works [6], [7]. RL agents can learn adaptive orchestration strategies by interacting with the environment and receiving feedback on their decisions, potentially handling the CC's dynamic nature more effectively. However, a key challenge in applying RL to the CC lies in the need for realistic and scalable training environments. Existing RL-based approaches often rely on simplified simulators or synthetic datasets, which may not accurately capture the complexities and dynamics of real-world CC environments [6], [8]. This discrepancy between training and real-world deployment can limit the effectiveness and generalizability of the learned policies.

This paper proposes the *gym-nne* framework to bridge the gap between the potential of RL and the need for realistic training environments in service placement for the CC. The *gym-nne* framework is specifically designed to enable scalable and cost-effective training of RL algorithms for service placement in the CC using real-world network data. Unlike traditional OpenAI Gym-based environments [9] that prioritize training speed over environmental realism, *gym-nne* incorporates a comprehensive dataset from commercial mobile operators (the NortNetEdge (NNE) dataset [10]). This dataset provides RL agents with realistic network dynamics, including latency variations and bandwidth fluctuations. This data-driven approach allows training in a setting that closely replicates real-world CC deployments. Consequently, *gym-nne* fosters the development of more robust and generalizable service orchestration strategies.

Our key contributions are two-fold:

- ***gym-nne* framework:** We investigate the application of *gym-nne* for service placement within the CC. We demonstrate how this framework, specifically designed for RL, can be adapted to model the interactions between services and resources throughout the CC. The framework has been open-sourced [1], enabling researchers to leverage *gym-nne* to evaluate their placement strategies.

‡ Andres F. Ocampo and José Santos contributed equally to this work.

[1] https://github.com/jpedro1992/gym-nne

- **Extensive Evalution:** We evaluate the performance of *gym-nne* using a variety of reward strategies that capture different deployment objectives, such as minimizing cost, latency, or resource inequality. We compare the performance of *gym-nne* with traditional heuristic approaches to service placement, highlighting the advantages of RL in handling dynamic resource allocation and service orchestration challenges within the CC.

The remainder of this paper is structured as follows. Section II reviews related work on service placement. Section III introduces the system and data models used in our evaluation. Section IV details our proposed RL-based service placement algorithm. Section V presents our evaluation methodology and results, comparing our approach to traditional methods. Finally, Section VII concludes the paper and outlines future research directions.

II. Related work

Efficient service orchestration in dynamic fog/edge environments is crucial for emerging applications with stringent latency and bandwidth demands. Traditional approaches [11]–[13], often reliant on static rules or heuristics, struggle to adapt to real-time changes in network conditions, resource availability, and user demands. This has led to growing interest in leveraging RL to develop more adaptive and dynamic service placement strategies. RL agents learn through interaction with the environment, making them well-suited for handling the dynamic and unpredictable nature of fog/edge systems [8].

Various studies [6], [14], [15] have explored leveraging value-based RL methods for optimal service placement. For example, Dehuri et al. [14], employed Deep Q-Network (DQN) to learn policies that map system states to actions, aiming to minimize latency and resource consumption. Similarly, Talpur et al. [15] investigated the use of Double DQN, a variant of DQN, to improve the stability and performance of service placement in fog environments. Researchers have also investigated policy-based methods, including Deep Deterministic Policy Gradients [16] and Proximal Policy Optimization (PPO) [17], for their ability to handle continuous action spaces and complex system dynamics often encountered in fog environments. Furthermore, hybrid approaches combining value-based and policy-based methods have been proposed in [18], aiming to capitalize on the strengths of both methodologies. These studies have demonstrated the potential of RL for optimizing various performance factors of service placement, including latency minimization [19], cost optimization [20], and resource allocation [14].

However, a significant limitation of existing research is the reliance on simplified simulations that do not capture the complexities of real-world fog environments within the CC. Furthermore, existing evaluations often focus on a narrow set of performance metrics, limiting the ability to comprehensively assess the effectiveness of different RL algorithms. In contrast to these limitations, *gym-nne* provides a more realistic and comprehensive evaluation platform for service placement strategies. Unlike existing works that rely on simplified simulation environments, *gym-nne* leverages real-world network traces to capture the dynamic nature of the CC, including fluctuations in latency, bandwidth availability, and user demands. By focusing on real-world data, our work enables a more accurate assessment of how different approaches would perform in practical deployments.

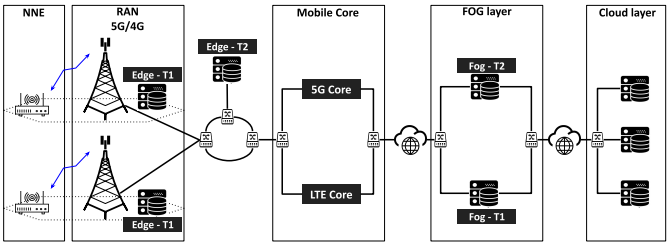


Fig. 1: System model for 6G service placement, illustrating the mobile system, CC, and NNE data collection.

III. System Overview and Data Models

This section provides an overview of the CC architecture and the data models used to evaluate our proposed RL-driven service placement approach for 6G networks. The aim is to realistically representing the complexities of the CC to enable a robust and insightful evaluation.

A. Modeling a Realistic Compute Continuum (CC)

The envisioned CC for 6G presents significant challenges for service placement due to its heterogeneous and dynamic nature. Our system model, illustrated in Fig. 1, captures these complexities by incorporating:

Multi-Operator, Multi-Technology Access Users can access services hosted at various locations within the CC (i.e., edge, fog, and cloud) through different operators and access technologies (4G and 5G). This introduces variations in terms of latency, bandwidth, and deployment cost.

Dynamic Resource Availability Computing Resources within the CC (e.g., the available CPU and memory at each location) fluctuate over time due to factors such as user demand and request arrival times. Our model accounts for these dynamics, requiring placement decisions to adapt to changing resource availability.

This realistic system model requires real-world data for meaningful evaluation. By incorporating datasets such as the NNE described in the next section, we can effectively capture the complexities and variations of real deployments, thus providing more reliable insights for optimizing service placement in 6G networks.

B. NortNetEdge (NNE) Dataset

To evaluate service placement strategies under realistic network conditions, we leverage the NNE dataset, a large-scale measurement platform deployed across Norway [10] that continuously captures 4G/5G network performance. This work uses data collected from November 2023 to February 2024, aggregating various Key Performance Indicators (KPIs) between network probes and placement nodes in 5-minute

intervals. This approach balances the need to capture network dynamics while maintaining computational feasibility during RL training. The KPIs measured include:

Round-trip time (RTT) 90th Percentile: Calculated from UDP ping measurements, representing the typical upper bound of latency experienced within each 5-minute window.

Downlink and Uplink Data Rate, Jitter: These metrics are derived from speed tests, aggregated, and mapped to align with 5-minute evaluation intervals. These provide insights into the bandwidth and stability of the connection.

Provider and network interface availability (4G/5G) varies across nodes in the NNE dataset. Nodes may access up to two of three providers, though some have access to only one. This applies to both 4G and 5G connectivity. With diverse measurement points, dynamic conditions, and multiple KPIs, the NNE dataset provides a robust basis for evaluating our RL-based service placement approach.

IV. Reinforcement Learning Methodology

A. The gym-nne framework

The *gym-nne* framework has been developed to enable scalable and cost-effective training of RL algorithms, as shown in Fig. 2. Traditional OpenAI Gym-based environments are typically designed to speed up the training process of RL. As such, our *gym-nne* framework supports multiple RL algorithms focused on generating an orchestration strategy using dynamic information from the NNE dataset as input. For instance, the framework continuously updates the RL environment with real-time data, such as RTT, and uplink and downlink bandwidth measurements for each provider, all sourced from the NNE dataset. Additionally, it dynamically adjusts the allocated CPU and memory resources for each node, reflecting the number of requests deployed at each node after each action selected by the RL algorithm. The *gym-nne* framework has been developed to mimic the behavior of deployment requests within the Compute Continuum, thereby providing the RL agent with pertinent information about network conditions collected by NNE nodes. Further details about our framework are detailed in the next subsections.

B. Observation Space

Table I exhibits the observation space applied in the *gym-nne* framework to characterize the environment at each step. It contains two main sets of metrics: *Request* and *Location*. The *Request* set corresponds to the application's deployment requirements, including the CPU and memory demands (ω_{cpu} and ω_{mem}), the latency threshold (Δ_r) that must be met by the hosting location, and the expected duration (T_r) of each request r, measured in time units. Given that RL is invoked with each new service request, the time interval between consecutive steps varies. Thus, T_r is incorporated into the observation space to enable the RL agent to learn the dynamics of the environment. This explicit inclusion of inter-arrival time between successive requests highlights changes in resource consumption across locations from one state to the next.

TABLE I: The structure of the Observation Space in the *gym-nne* framework for each location within the Cloud Continuum.

Set	Metric	Description
Request	ω_{cpu}	The CPU request (in millicpu).
	ω_{mem}	The memory request (in mebibyte).
	Δ_r	The latency threshold (in ms).
	T_r	The execution time (in time units).
Location	Π_{cpu}	The location's cpu capacity.
	Π_{mem}	The location's memory capacity.
	Θ_{cpu}	The CPU allocated in the location.
	Θ_{mem}	The memory allocated in the location.
	P_p	The location's provider identifier.
	I_i	The location's interface identifier.
	U_l	The uplink capacity (in Mbit/s).
	D_l	The downlink capacity (in Mbit/s).
	$RQ90_l$	The RTT 90% quartile (in ms).
	J_l	The jitter (in ms).
	δ_l	The processing latency of location l.

TABLE II: The hardware configuration of each location based on Amazon EC2 On-Demand Pricing [21].

Type	Latency (β_l)	Amazon ($/h)	Cost (τ_c)	CPU	RAM
Cloud	10	2XL (0.2688)	16.0	8.0	32.0
Fog-T2	7.5	XL (0.1344)	8.0	4.0	16.0
Fog-T1	5.0	L (0.0672)	4.0	2.0	8.0
Edge-T2	2.5	M (0.0336)	2.0	2.0	4.0
Edge-T1	1.0	S (0.0168)	1.0	2.0	2.0

The *Location* set corresponds to the metrics related to the current status of the infrastructure, detailing both available resources and network conditions. These metrics include the total capacities for CPU and memory (Π_{cpu} and Π_{mem}), the amounts of CPU and memory currently allocated (Θ_{cpu} and Θ_{mem}), provider and interface identifiers (P_p and I_i), uplink and downlink capacities (U_l and D_l), jitter (J_l), the 90th percentile of RTT ($RQ90_l$), and the processing latency at each location (δ_l). The allocation of CPU and memory resources for each location is dynamically adjusted based on the number of hosted requests. As the number of deployed requests grows, the allocated CPU and memory increase proportionally to the specified requirements for each request. Similarly, when a request is terminated, the allocated resources are adjusted to reflect the freed CPU and memory. The available free CPU (Ω_{cpu}) and memory (Ω_{mem}) resources are defined by equations (1) and (2), respectively. However, these metrics are excluded from the observation space as our experiments showed that including them did not enhance the performance of the algorithms since the agents already have access to the total capacity and the currently allocated amount of resources. The processing latency (δ_l) is also influenced by the number of hosted requests at each location, increasing by a specific factor (i.e., 2 ms per hosted request). By incorporating these comprehensive metrics, the *gym-nne* framework provides a detailed view of resource availability and network conditions across various locations, thereby enabling more efficient orchestration strategies by the RL agent.

$$\underbrace{\Omega_{cpu}}_{\text{free CPU}} = \underbrace{\Pi_{cpu}}_{\text{CPU Capacity}} - \underbrace{\Theta_{cpu}}_{\text{CPU Allocated}} \tag{1}$$

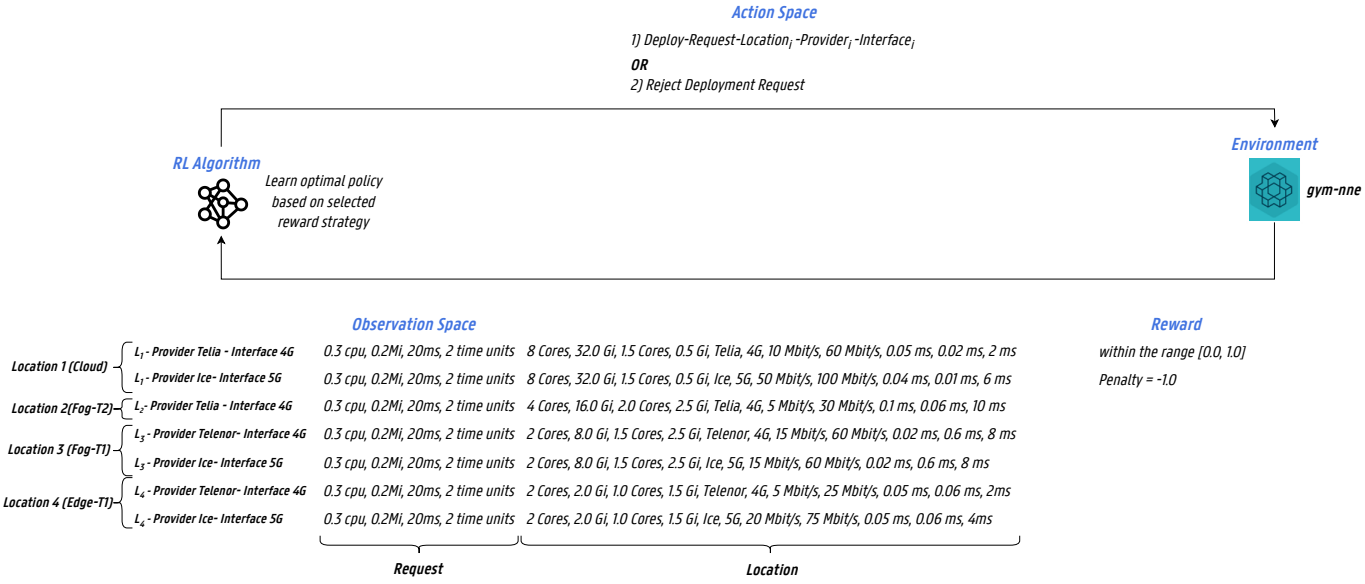


Fig. 2: Overview of the *gym-nne* framework.

TABLE III: The structure of the Action Space.

Action Name	Description
Deploy-l-p-i	Deploy request in loc. l. Use prov. p with interf. i.
Reject	The agent rejects the request. Nothing is deployed.

$$\underbrace{\Omega_{mem}}_{\text{free Memory}} = \underbrace{\Pi_{mem}}_{\text{Memory Capacity}} - \underbrace{\Theta_{mem}}_{\text{Memory Allocated}} \tag{2}$$

Table II provides a comprehensive overview of the resource capacities for each location type along with their corresponding deployment costs. Resource capacities are quantified within a range of $[2.0, 32.0]$ units, and allocated resources are initiated within $[0.0, 0.2]$ units, to account for the reserved resources allocated to background services such as monitoring. Each location type is also associated with an access latency, which varies between $[0.0, 10.0]$ ms, contingent upon the specific type of the location. This information aids the RL agent to select adequate actions at a given moment from the action space described next, ensuring an efficient orchestration strategy.

C. Action Space

Table III outlines the action space designed for *gym-nne* as a discrete set of possible actions, with a single action executed at each timestep. For any given request, the RL agent has the option to either deploy or reject the request. Rejection is allowed when computational resources are insufficient, and no location can fulfill the request's requirements. In such cases, the agent is not penalized, acknowledging the constraint of limited resources. The total size of the action space depends on the total number of locations and corresponding providers (i.e., Telia, Telenor, Ice) and interfaces (i.e., 4G and 5G) available in the NNE dataset. Assuming a CC scenario with l locations, the action space is computed as $l \times 3 \times 2 = l \times 6$, where 3 represents the number of providers and 2 represents the number of interfaces. However, it is important to note that not all providers and interfaces are available for every location in the NNE dataset, as previously discussed in Section III.

Regarding penalties (i.e., negative rewards), a straightforward approach, often adopted in the literature [20], is to penalize the agent for selecting invalid actions. These invalid actions are typically pre-determined based on the available computing resources, and if the specific provider and interface are supported. An alternative and more advanced approach is known as action masking [22], which familiarizes the agent that, depending on the current state s, certain actions are invalid. Recent studies have demonstrated that action masking significantly enhances performance and sample efficiency compared to the penalty method. The action masks for each location l in state s can be defined as follows:

$$mask(s)[l] = \begin{cases} \textit{True}, & \text{If location } l \text{ has enough resources,} \\ & \text{and supports prov. } p \text{ and interf. } i \\ \textit{False}, & \text{Otherwise.} \end{cases} \tag{3}$$

For rejection actions, the action mask is always set to true, ensuring the agent is not locked out in scenarios where all other actions are invalid.

D. Reward Function

The reward function is critical for steering the RL agent towards maximizing accumulative rewards by selecting optimal actions based on the current observation state. In this study, a multi-objective reward function (4) has been designed to incorporate four distinct objectives: cost-aware (5), latency-aware (6), bandwidth-aware (7), and inequality-aware (8). When the agent chooses to accept a deployment request, it is rewarded positively according to these objectives, with respective weights (ω_c, ω_l, ω_b and ω_i), and the rewards are normalized within the range $[0.0, 1.0]$. In contrast, if the agent

rejects the request despite sufficient computing resources being available, it incurs a penalty of -1.

$$r = \begin{cases} \omega_c \times r_{cost} + \omega_l \times r_l + \\ \omega_b \times r_{band} + \omega_i \times r_{ineq} & \text{If the request is accepted.} \\ -1 & \text{If the request is rejected.} \end{cases} \tag{4}$$

$$r_{cost} = 1.0 - \Gamma_d \quad \text{where:} \quad \Gamma_d = \text{Expected Deployment Cost} \tag{5}$$

$$r_{latency} = 1.0 - \lambda_d \quad \text{where:} \quad \lambda_d = \text{Expected Total Latency.} \tag{6}$$

$$r_{bandwidth} = 1.0 - B \quad \text{where:} \quad B = U_l + D_l \tag{7}$$

$$r_{inequality} = 1.0 - G \quad \text{where:} \quad G = \textit{Gini Coefficient} \tag{8}$$

The **cost-aware** function guides the agent to deploy requests in locations within the CC in a manner that minimizes allocation costs (denoted as τ_c). Locations classified as *cloud* generally incur higher deployment costs compared to *fog* and *edge* types, and also often offer higher latency. Thus, the RL agent will prefer to deploy requests on edge or fog locations to maximize the accumulated reward. However, fog and edge locations typically have fewer computing resources compared to cloud locations, which may limit their ability to accommodate multiple requests. The deployment cost for a request, denoted as Γ_d, is determined based on the allocation cost τ_c of the selected location.

The **latency-aware** function aims to minimize the expected latency of a deployment request. The expected latency (λ_d) is computed based on multiple latency factors, considering the specific location hosting the request. The latency-aware function seeks to find placement schemes that minimize the overall latency (9) by favoring locations that focus not only on meeting the latency threshold of the request but also providing low RTT, low processing latency, and a low access latency.

$$\underbrace{\lambda_d}_{\text{Total Latency (in ms)}} = \underbrace{RQ90_l}_{\text{RTT 90\% percentile}} + \underbrace{\delta_l}_{\text{Processing Latency}} + \underbrace{\beta_l}_{\text{Access Latency}} \tag{9}$$

The **bandwidth-aware** function aims to select a location with the highest available downlink and uplink bandwidth. The reward is computed based on the combined bandwidth capacity, taking into account both uplink (U_l) and downlink (D_l). This function influences the RL agent to prioritize locations offering the maximum bandwidth at a given time.

Lastly, the **inequality-aware** function directs the RL agent to choose deployment strategies that evenly distribute requests across the available locations. The reward is calculated based on the *Gini Coefficient* (G) that ranges from $[0.0, 1.0]$, where 0 indicates perfect equality (each location hosts an equal number of requests), and 1 indicates perfect inequality (all requests deployed in one location). A lower *Gini Coefficient* indicates then a more equitable distribution. The *Gini Coefficient* is an accurate measure of inequality in a distribution, calculated using the formula:

$$G = \frac{\sum_{i=1}^{l} \sum_{j=1}^{l} |L_i - L_j|}{2l^2 \bar{L}} \tag{10}$$

where:

G is the Gini coefficient.

l is the number of locations.

L_i is the number of requests deployed by location i.

$\bar{L}$ is the average number of requests across all locations.

TABLE IV: The evaluated reward strategies.

Name	ω_c	ω_l	ω_b	ω_i
Cost	1.0	0.0	0.0	0.0
Latency	0.0	1.0	0.0	0.0
Bandwidth	0.0	0.0	1.0	0.0
Inequality	0.0	0.0	0.0	1.0
LatCost	0.5	0.5	0.0	0.0
LatIneq	0.0	0.5	0.0	0.5
CostIneq	0.5	0.0	0.0	0.5
BandLat	0.0	0.5	0.5	0.0
BandCost	0.5	0.0	0.5	0.0

TABLE V: Deployment properties applied in the evaluation based on different deployment type requests.

Deployment Type	CPU	RAM	Latency Threshold
type-1	0.20	0.30	50 ms
type-2	0.20	0.64	150 ms
type-3	0.20	0.50	20 ms
type-4	0.20	0.50	40 ms
type-5	0.20	0.75	200 ms
type-6	0.015	0.25	50 ms
type-7	0.05	0.016	45 ms
type-8	0.20	0.20	60 ms
type-9	0.015	0.25	100 ms
type-10	0.20	0.60	35 ms

V. Evaluation Setup

This section presents an overview of the several reward strategies used to validate the *gym-nne* framework, followed by specific deployment requirements used in the evaluation, and explanations about the dynamics of the applied RL environment.

Reward Strategies Table IV details eight distinct reward strategies considered in the evaluation of the *gym-nne* framework. By evaluating numerous reward strategies, the *gym-nne* framework aims to provide insights into the effectiveness of different allocation strategies in the deployment of requests across distributed computing environments.

RL Algorithm One notable RL algorithm, known as Maskable (Mask) PPO [22], which supports discrete action spaces has been evaluated based on its reliable implementation in Python within the stable baselines 3 framework [23]. MaskPPO incorporates invalid action masking into the PPO algorithm, maintaining a similar behavior to the core PPO algorithm while adding support for action masking. In this study, additional RL algorithms have been evaluated, but our

findings revealed that MaskPPO consistently outperformed the other RL methods. Thus, in this paper, we decided to focus our evaluation on assessing the performance of MaskPPO under various reward strategies. The objective is to analyze the effectiveness and robustness of these strategies, thereby providing a nuanced understanding of their applicability in real-world deployment scenarios. Through this extensive evaluation, we aim to offer valuable insights for researchers and practitioners seeking to leverage RL techniques for efficient deployment strategies within the CC.

Application Requests Table V shows the deployment requirements for various application requests used in the evaluation of the *gym-nne* framework. These diverse types represent a broad range of use cases, ensuring that the RL agents encounter diverse patterns in computing resource demands and latency thresholds. It allows for a comprehensive assessment of the RL agent's ability to handle varying levels of computational intensity and time sensitivity, thereby validating its effectiveness in dynamic environments.

***gym-nne* Framework** has been implemented in Python to facilitate seamless interaction with both the OpenAI Gym and Stable Baselines 3 libraries. In our evaluation, an episode consists of 100 steps during which the RL agent aims to maximize the reward based on the current deployment request. When the agent selects an available location to host the request, the CPU and memory usage of that location increase. In contrast, if a request is terminated based on a mean service duration (default is one unit), the CPU and memory usage decrease accordingly. The expected duration of the request (T_r) is then randomized around the mean service duration, ensuring that RL agents encounter varied request patterns across consecutive episodes. During training, four locations have been considered, with the type of each location (Table II) being randomized. The same randomization process was applied during testing. The RL algorithms have been trained over 2000 episodes, utilizing a 14-core Intel i7-12700H CPU @ 4.7 GHz processor with 16 GB of memory. The performance of the RL agents has been evaluated based on the following metrics:

- **Accumulated reward** during each episode. It refers to the total sum of rewards obtained by an RL agent throughout each episode as it interacts with the *gym-nne* environment.
- **Percentage of rejected requests** expressed in the range $[0, 100]$.
- **Average deployment cost (in units)** incurred for deploying each request.
- **Average processing latency (in ms)** for each accepted request based on the number of hosted requests.
- **Average access latency (in ms)** for each accepted request based on the location type.
- **Average RTT** for each accepted request.
- **Average total latency (in ms)** for each accepted request based on the previously presented formula.
- **Average uplink and donwlink bandwidth (in Mbit/s)** for each accepted request.
- **Percentage of requests in each provider** expressed in the range $[0, 100]$.
- **Gini Coefficient** highlighting the inequality of the deployment scheme, with values ranging from 0 (perfect equality) to 1 (maximum inequality), helping to assess the fairness in request distribution.

TABLE VI: The execution time per episode during training for the *gym-nne* framework indicates that RL algorithms are significantly slower than common heuristics, due to the need for policy learning and adaptation.

Algorithm	Execution Time per episode (in s)	Execution Time for 2000 episodes
MaskPPO (*Cost*)	0.82 ± 0.11	27.3 minutes
MaskPPO (*Latency*)	0.63 ± 0.08	21.0 minutes
MaskPPO (*Bandwidth*)	0.94 ± 0.13	31.3 minutes
MaskPPO (*Inequality*)	1.07 ± 0.15	35.7 minutes
Cost-Greedy	0.076 ± 0.004	2.53 minutes
Bandwidth-Greedy	0.066 ± 0.001	2.20 minutes
Latency-Greedy	0.073 ± 0.002	2.43 minutes

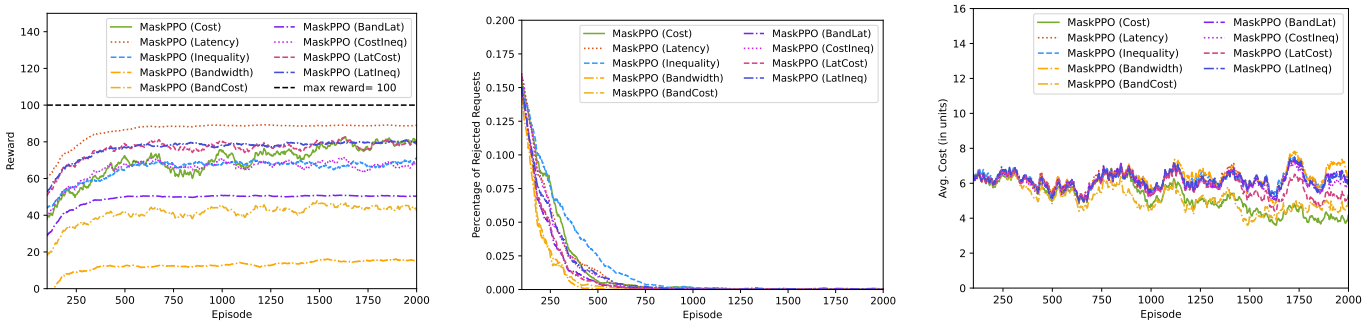
Three heuristic-based baselines have also been evaluated to compare against the RL-based methods:

- **Cost-Greedy**: assigns the request to the location with the lowest deployment cost based on the type of location. It prioritizes minimizing cost without considering other factors such as latency or bandwidth.
- **Bandwidth-Greedy**: assigns the request to the location with the highest downlink bandwidth, aiming to optimize for network throughput.
- **Latency-Greedy**: assigns the request to the location with the lowest RTT. However, this heuristic does not consider other latency sources.

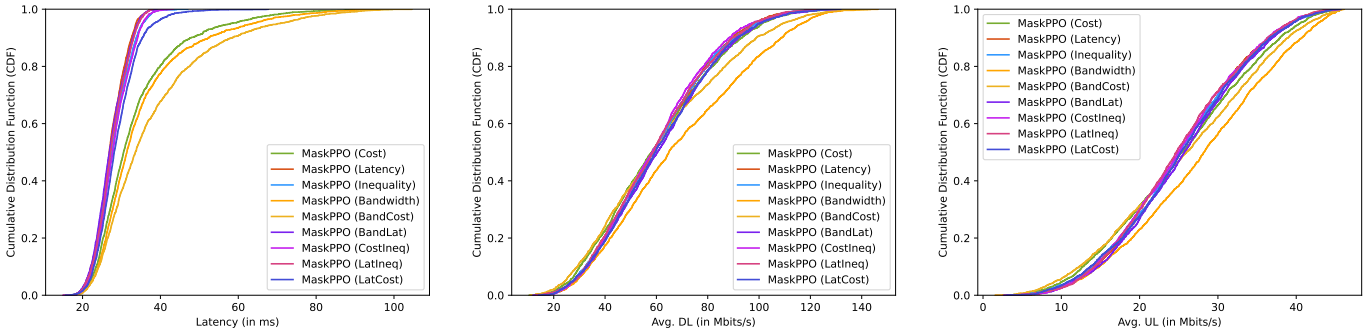
VI. Results

RL versus Heuristic Execution Time The time complexity for the RL agents and heuristics has been assessed based on their execution times during the training phase, as detailed in Table VI. The results highlight that training RL agents in environments similar to the *gym-nne* framework can significantly accelerate the training process compared to online learning in live operational settings [24]. However, RL methods are considerably slower than all heuristic baselines. Specifically, RL algorithms require on average 21 to 35 minutes to complete the training phase over 2000 episodes. The high execution time is required for policy training and adaptation, as well as for making necessary adjustments to the environment based on the NNE dataset. In contrast, heuristic methods complete 2000 episodes in about 2 to 3 minutes, as these do not require policy training.

RL Training Fig. 3 illustrates the performance of various reward strategies during the training phase over 2000 episodes. To mitigate spikes, a smoothing window of 100 episodes has been applied to all curves in these figures. Despite fluctuations, all strategies converge between the 300th and 600th episodes, with some showing slight improvements in rewards beyond this point. Most strategies achieve high accumulated rewards with MaskPPO (*Latency*) converging at a slightly higher state, resulting in higher rewards on average. Notably, MaskPPO effectively learns to minimize rejections for all strategies, as

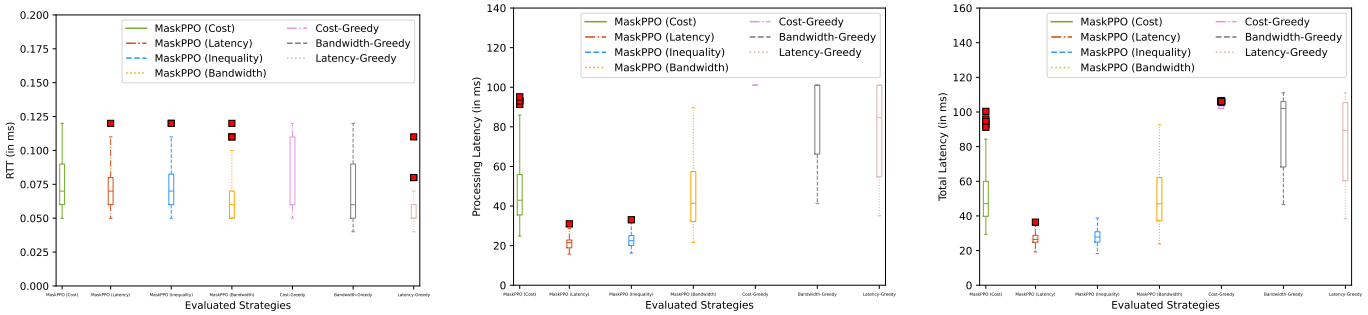


(a) Accumulated Reward. (b) Number of Rejected Requests. (c) Deployment Cost.

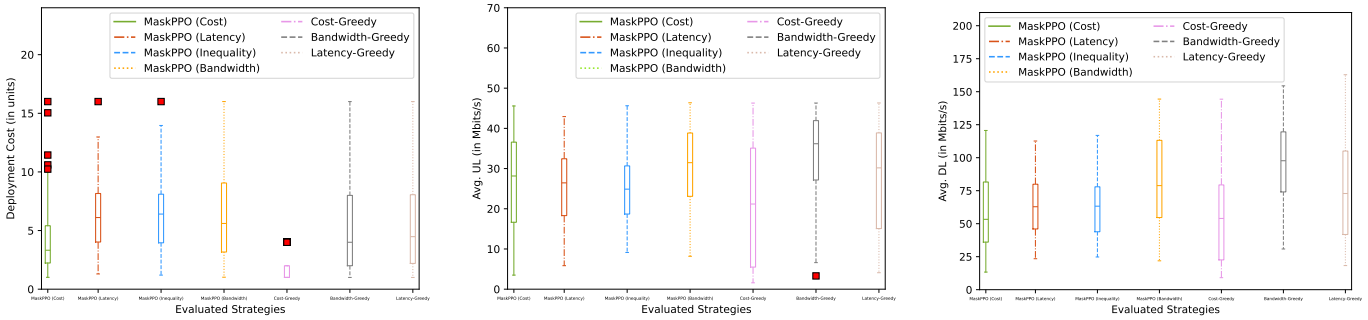


(d) Total Latency (in ms). (e) Downlink bandwidth (in Mbit/s). (f) Uplink bandwidth (in Mbit/s).

Fig. 3: The training results for the MaskPPO agent evaluated for the multiple reward strategies.



(a) RTT (in ms). (b) Processing Latency (in ms). (c) Total Latency (in ms.



(d) Deployment Cost. (e) Uplink bandwidth (in Mbit/s). (f) Downlink bandwidth (in Mbit/s).

Fig. 4: The testing results for the MaskPPO agent evaluated for the multiple reward strategies.

evidenced by the pursuit of higher rewards. By the end of the training, all strategies reach an almost 0% rejection rate. In terms of deployment cost, cost-focused strategies typically yield lower values compared to other strategies by the end of the training phase. Additionally, latency-aware strategies consistently result in lower total latency, as demonstrated by the Cumulative Distribution Function (CDF) in Fig. 3d. In contrast, cost and bandwidth-focused strategies tend to incur higher latency. Lastly, Fig. 3e and Fig. 3f illustrate that bandwidth-aware approaches provide the highest downlink and uplink data rate, respectively.

RL Testing During the testing phase, each strategy has

TABLE VII: Results obtained during the testing phase for the multiple RL agents and heuristics - Part I.

Algorithm	Strategy	Acc. Reward	Deployment Cost (in units)	Rejected Requests (in %)	Processing Latency (in ms)	Access Latency (in ms)	RTT (in ms)	Total Latency (in ms)
MaskPPO	*Cost*	78.2 ± 3.9	4.2 ± 0.6	0.02 ± 0.03	47.4 ± 3.3	4.0 ± 0.6	0.07 ± 0.004	51.6 ± 3.3
MaskPPO	*Latency*	89.1 ± 0.3	6.3 ± 0.6	**0.0**	**21.6 ± 0.7**	5.3 ± 0.4	0.07 ± 0.003	**26.9 ± 0.7**
MaskPPO	*Bandwidth*	15.6 ± 1.1	6.4 ± 0.8	0.04 ± 0.04	45.5 ± 3.2	5.3 ± 0.5	0.06 ± 0.003	50.8 ± 3.2
MaskPPO	*Inequality*	67.9 ± 2.5	6.3 ± 0.6	0.02 ± 0.03	22.9 ± 0.7	5.3 ± 0.4	0.07 ± 0.003	28.3 ± 0.9
MaskPPO	*LatCost*	76.8 ± 2.0	5.6 ± 0.6	0.02 ± 0.03	33.4 ± 1.8	4.9 ± 0.4	0.07 ± 0.003	38.5 ± 1.8
MaskPPO	*LatIneq*	78.7 ± 1.4	6.5 ± 0.6	0.01 ± 0.02	22.6 ± 0.7	5.4 ± 0.4	0.08 ± 0.003	28.2 ± 0.7
MaskPPO	*CostIneq*	68.4 ± 2.5	6.1 ± 0.6	0.07 ± 0.06	22.8 ± 0.8	5.2 ± 0.4	0.08 ± 0.003	28.1 ± 0.9
MaskPPO	*BandLat*	50.4 ± 0.4	6.4 ± 0.6	0.01 ± 0.02	**22.4 ± 0.7**	5.4 ± 0.4	0.08 ± 0.003	**27.9 ± 0.8**
MaskPPO	*BandCost*	44.2 ± 2.2	4.6 ± 0.6	0.01 ± 0.02	56.9 ± 4.1	4.3 ± 0.4	0.08 ± 0.005	61.3 ± 4.1
Cost-Greedy	-	-	**1.7 ± 0.2**	**0.0**	101.0 ± 0.01	**2.0 ± 0.3**	0.08 ± 0.006	103.1 ± 0.3
Bandwidth-Greedy	-	-	5.4 ± 0.9	**0.0**	84.2 ± 4.1	4.8 ± 0.6	0.07 ± 0.005	89.1 ± 4.1
Latency-Greedy	-	-	6.1 ± 0.8	**0.0**	78.7 ± 4.2	5.2 ± 0.5	**0.05 ± 0.001**	83.9 ± 4.3

TABLE VIII: Results obtained during the testing phase for the multiple RL agents and heuristics - Part II.

Algorithm	Strategy	Uplink (in Mbit/s)	Downlink (in Mbit/s)	Jitter (in ms)	Gini Coeff.	Telia (in %)	Telenor (in %)	Ice (in %)
MaskPPO	*Cost*	26.3 ± 2.2	60.3 ± 5.4	3.6 ± 0.5	0.43 ± 0.02	44.7 ± 5.7	31.4 ± 5.0	23.9 ± 6.0
MaskPPO	*Latency*	25.6 ± 1.8	62.7 ± 4.5	3.3 ± 0.3	**0.30 ± 0.03**	45.9 ± 3.2	35.1 ± 3.5	19.0 ± 3.7
MaskPPO	*Bandwidth*	30.4 ± 1.9	81.7 ± 6.4	**3.1 ± 0.3**	0.46 ± 0.03	58.2 ± 5.7	30.1 ± 5.2	11.7 ± 3.6
MaskPPO	*Inequality*	25.0 ± 1.72	61.7 ± 4.3	3.6 ± 0.4	**0.30 ± 0.03**	35.9 ± 3.6	**38.9 ± 3.9**	25.2 ± 4.5
MaskPPO	*LatCost*	27.2 ± 2.0	63.8 ± 5.0	3.6 ± 0.6	0.38 ± 0.03	42.4 ± 4.6	**42.9 ± 4.8**	14.7 ± 3.8
MaskPPO	*LatIneq*	25.1 ± 1.6	58.4 ± 4.1	3.5 ± 0.5	**0.28 ± 0.03**	38.5 ± 3.5	34.8 ± 4.2	**26.7 ± 4.0**
MaskPPO	*CostIneq*	25.1 ± 1.7	59.6 ± 4.3	3.8 ± 0.6	**0.26 ± 0.03**	39.1 ± 3.3	35.6 ± 3.3	25.3 ± 4.0
MaskPPO	*BandLat*	25.7 ± 1.6	60.2 ± 3.9	3.7 ± 0.8	0.31 ± 0.03	47.2 ± 4.0	29.2 ± 3.3	23.6 ± 4.4
MaskPPO	*BandCost*	25.7 ± 2.3	62.3 ± 5.7	3.9 ± 1.3	0.50 ± 0.03	42.7 ± 6.6	29.3 ± 5.7	**28.0 ± 7.5**
Cost-Greedy	-	22.0 ± 3.0	56.8 ± 6.9	3.3 ± 0.4	0.67 ± 0.02	57.0 ± 9.7	14.0 ± 6.8	**29.0 ± 8.9**
Bandwidth-Greedy	-	**33.5 ± 2.1**	**96.7 ± 5.4**	**2.9 ± 0.5**	0.61 ± 0.02	67.4 ± 8.0	8.2 ± 4.5	24.4 ± 7.6
Latency-Greedy	-	27.6 ± 2.6	75.7 ± 6.9	**3.0 ± 0.3**	0.64 ± 0.01	**72.2 ± 8.0**	27.1 ± 8.0	0.7 ± 1.1

been executed over 100 episodes, using the saved configuration from the training phase (over 2000 episodes). Fig. 4 shows the performance of various reward strategies, with box plots showcasing their variations. Additionally, Table VII and Table VIII summarize the results based on the selected performance metrics, providing a detailed comparison of each strategy's effectiveness. The highest accumulated reward was achieved by the *Latency* strategy, with a value of 89.1. This strategy demonstrated superior performance in minimizing overall latency, obtaining an average of 26.9 ms, which is crucial for latency-sensitive applications requiring low response times. The *BandLat* strategy also performed well, recording a total latency of 27.9 ms, closely following the *Latency* strategy.

Regarding deployment costs, the Cost-Greedy heuristic exhibited exceptional cost efficiency, with an average deployment cost of 1.7 units, the lowest among all tested strategies. This greedy approach is particularly suitable for scenarios where reducing operational costs is a primary concern. However, it is worth noting that the Cost-Greedy heuristic incurred higher processing and total latency, indicating a trade-off between cost efficiency and latency minimization. All heuristics handle all incoming requests without any rejections, achieving a 0% rejection rate. In contrast, among the MaskPPO strategies, only the *Latency* strategy achieved a 0% rejection rate, while other strategies maintained a minimal rejection rate between 0.01% and 0.04%. The Bandwidth-Greedy heuristic achieved the highest uplink and downlink rates, at 33.5 Mbit/s and 96.7 Mbit/s, respectively, significantly outperforming other strategies in terms of bandwidth. Additionally, it recorded the lowest average jitter at 2.9 ms, indicating the most stable and predictable data transmission. This makes it an ideal choice for applications requiring high data throughput. The *Bandwidth* strategy also performed well, with uplink and downlink rates of 30.4 Mbit/s and 81.7 Mbit/s, respectively.

Regarding request distribution, all MaskPPO strategies that aim for equality demonstrated low Gini coefficients, ranging from 0.26-0.30, indicating a commitment to a balanced distribution. In contrast, the Cost-Greedy, Latency-Greedy, and Bandwidth-Greedy heuristics exhibited the highest Gini coefficients, 0.67, 0.64, and 0.61 respectively. This indicates greater disparities in request distribution, potentially leading to unequal network performance across users. The Latency-Greedy heuristic showed a pronounced preference for deploying requests to the Telia provider (72.2%), potentially indicating a reliance on this operator, which could lead to congestion and overuse. All MaskPPO strategies demonstrated versatility, with specific configurations excelling in various metrics such as low jitter, equitable resource distribution, and balanced operator usage.

In conclusion, our extensive evaluation revealed distinct advantages among the tested strategies and heuristics. The *Latency* strategy excelled in reducing total latency, while the Bandwidth-Greedy heuristic achieved the highest uplink and downlink rates. The Cost-Greedy heuristic offered the lowest deployment cost but also incurred the highest total latency. Heuristics performed well for simple objectives, but these may encounter limitations with increased complexity, especially when multiple performance factors come into play.

In contrast, RL is particularly effective when optimizing for multiple objectives simultaneously. Our findings highlight that RL can find an efficient policy based on different performance factors as demonstrated by our results.

VII. Conclusions

This study investigates RL methods for service placement within 6G networks, offering significant advancements in managing the distribution of services across the CC. The paper presents an extensive evaluation under varying network conditions and workloads, confirming that RL can develop efficient placement strategies by considering multiple performance factors such as deployment costs, latency, and bandwidth. The results highlight the adaptability of our RL-driven model to dynamic network environments, showcasing its effectiveness through a multi-objective reward function. This adaptability is crucial for low latency applications in 6G networks, emphasizing the potential of RL as a transformative tool for intelligent service management in next-generation communication systems. Our findings also reveal that while traditional heuristics remain effective for simple objectives, they tend to falter as complexity increases. RL is particularly effective when optimizing for multiple objectives simultaneously as demonstrated by our results. Future work will explore multi-agent RL and integrate energy consumption models to evaluate efficiency in service placement strategies, where multiple RL agents can either cooperate or compete, collectively optimizing service placement decisions. This work showcases the potential of intelligent RL systems in the future of communication networks, providing valuable insights for developing more robust and efficient 6G management capabilities.

Acknowledgment

José Santos is funded by the Research Foundation Flanders (FWO), grant number 1299323N.

References

[1] M.-J. Montpetit and N. Crespi, "Computing in the network: The core-edge continuum in 6g network," *Shaping Future 6G Networks: Needs, Impacts, and Technologies*, pp. 133–166, 2021.

[2] J. Santos, T. Wauters, B. Volckaert, and F. De Turck, "Towards low-latency service delivery in a continuum of virtual resources: State-of-the-art and research directions," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 4, pp. 2557–2589, 2021.

[3] P. Soumplis, P. Kokkinos, A. Kretsis, P. Nicopolitidis, G. Papadimitriou, and E. Varvarigos, "Resource allocation challenges in the cloud and edge continuum," in *Advances in Computing, Informatics, Networking and Cybersecurity: A Book Honoring Professor Mohammad S. Obaidat's Significant Scientific Contributions*. Springer, 2022, pp. 443–464.

[4] J. Santos, M. Zaccarini, F. Poltronieri, M. Tortonesi, C. Sleianelli, N. Di Cicco, and F. De Turck, "Efficient microservice deployment in kubernetes multi-clusters through reinforcement learning," in *NOMS 2024-2024 IEEE Network Operations and Management Symposium*, 2024, pp. 1–9.

[5] K. Fu, W. Zhang, Q. Chen, D. Zeng, and M. Guo, "Adaptive resource efficient microservice deployment in cloud-edge continuum," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 8, pp. 1825–1840, 2021.

[6] J. Santos, T. Wauters, B. Volckaert, and F. De Turck, "Resource provisioning in fog computing through deep reinforcement learning," in *2021 IFIP/ieee international symposium on integrated network management (IM)*. IEEE, 2021, pp. 431–437.

[7] A. M. Maia and Y. Ghamri-Doudane, "A deep reinforcement learning approach for the placement of scalable microservices in the edge-to-cloud continuum," in *GLOBECOM 2023-2023 IEEE Global Communications Conference*. IEEE, 2023, pp. 479–485.

[8] M. Goudarzi, M. Palaniswami, and R. Buyya, "A distributed deep reinforcement learning technique for application placement in edge and fog computing environments," *IEEE Transactions on Mobile Computing*, vol. 22, no. 5, pp. 2491–2505, 2023.

[9] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016, accessed on 04 July 2024. [Online]. Available: https://github.com/openai/gym.

[10] A. Kvalbein, D. Baltrunas, K. Evensen, J. Xiang, A. Elmokashfi, and S. Ferlin-Oliveira, "The nornet edge platform for mobile broadband measurements," *Comput. Netw.*, vol. 61, no. C, p. 88–101, mar 2014. [Online]. Available: https://doi.org/10.1016/j.bjp.2013.12.036

[11] J. Santos, T. Wauters, B. Volckaert, and F. De Turck, "Towards network-aware resource provisioning in kubernetes for fog computing applications," in *2019 IEEE Conference on Network Softwarization (NetSoft)*. IEEE, 2019, pp. 351–359.

[12] A. M. Maia, Y. Ghamri-Doudane, D. Vieira, and M. F. de Castro, "An improved multi-objective genetic algorithm with heuristic initialization for service placement and load distribution in edge computing," *Computer networks*, vol. 194, p. 108146, 2021.

[13] Z. N. Samani, N. Mehran, D. Kimovski, S. Benedict, N. Saurabh, and R. Prodan, "Incremental multilayer resource partitioning for application placement in dynamic fog," *IEEE Transactions on Parallel and Distributed Systems*, vol. 34, no. 6, pp. 1877–1896, 2023.

[14] C. K. Dehury, S. Poojara, and S. N. Srirama, "Def-drel: Towards a sustainable serverless functions deployment strategy for fog-cloud environments using deep reinforcement learning," *Applied Soft Computing*, vol. 152, p. 111179, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1568494623011973

[15] A. Talpur and M. Gurusamy, "Drld-sp: A deep-reinforcement-learning-based dynamic service placement in edge-enabled internet of vehicles," *IEEE Internet of Things Journal*, vol. 9, no. 8, pp. 6239–6251, 2022.

[16] A. Qadeer and M. J. Lee, "Deep-deterministic policy gradient based multi-resource allocation in edge-cloud system: A distributed approach," *IEEE Access*, vol. 11, pp. 20 381–20 398, 2023.

[17] J. Wang, J. Hu, G. Min, A. Y. Zomaya, and N. Georgalas, "Fast adaptive task offloading in edge computing based on meta reinforcement learning," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 1, pp. 242–253, 2021.

[18] Z. Cheng, M. Liwang, N. Chen, L. Huang, X. Du, and M. Guizani, "Deep reinforcement learning-based joint task and energy offloading in uav-aided 6g intelligent edge networks," *Computer Communications*, vol. 192, pp. 234–244, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0140366422002195

[19] S. Wang, C. Yuen, W. Ni, Y. L. Guan, and T. Lv, "Multiagent deep reinforcement learning for cost- and delay-sensitive virtual network function placement and routing," *IEEE Transactions on Communications*, vol. 70, no. 8, pp. 5208–5224, 2022.

[20] H. Sami, A. Mourad, H. Otrok, and J. Bentahar, "Demand-driven deep reinforcement learning for scalable fog and service placement," *IEEE Transactions on Services Computing*, vol. 15, no. 5, pp. 2671–2684, 2021.

[21] Amazon AWS, "Amazon ec2 on-demand pricing." accessed on 28 September 2023. [Online]. Available: https://aws.amazon.com/ec2/pricing/on-demand/.

[22] S. Huang and S. Ontañón, "A closer look at invalid action masking in policy gradient algorithms," *arXiv preprint arXiv:2006.14171*, 2020.

[23] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021.

[24] J. Santos, T. Wauters, B. Volckaert, and F. De Turck, "gym-hpa: Efficient auto-scaling via reinforcement learning for complex microservice-based applications in kubernetes," in *NOMS 2023-2023 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2023, pp. 1–9.