Reinforcement Learning-based Orchestration of XR applications in Distributed 6G Cloud Infrastructures

Javad Sameri*‡, José Santos*, Sam Van Damme*, Susanna Schwarzmann†, Qing Wei†,
Riccardo Trivisonno†, Filip De Turck*, Maria Torres Vega‡
* IDLab, Department of Information Technology, Ghent University - imec, Ghent, Belgium
† Huawei Technologies, Germany
‡ eMedia Research Lab, Department of Electrical Engineering (ESAT), KU Leuven, Belgium
Email: Javad.Sameri@UGent.be

***Abstract*—eXtended Reality (XR) and holographic telepresence place stringent Quality of Service (QoS) demands on network infrastructure, requiring ultra-low latency, high throughput, and reliable connectivity. Meeting such QoS demands is critical in dynamic, distributed cloud environments, but does not always guarantee a satisfactory user experience. Quality of Experience (QoE) captures the user's perception of service performance, which may be influenced by factors not fully reflected in system-level metrics. Thus, novel orchestration strategies must consider both QoS and QoE. This paper proposes a Reinforcement Learning (RL)-driven approach to edge-cloud orchestration capable of adapting to dynamic network conditions, leveraging a multi-objective reward function, including both QoS and QoE aspects, to guide service placement decisions. Evaluation shows that our RL approach reaches a 21.3% QoE gain over heuristics and 14.7% over balanced strategies, with 100% request acceptance. The results highlight the robustness and scalability of RL-driven orchestration, particularly for latency-sensitive 6G applications. Our findings also reveal the limitations of traditional heuristics under complex objectives and highlight the potential of RL as a transformative tool for intelligent network and service management in next-generation communication systems.**

***Index Terms*—Quality of Experience, Reinforcement Learning, Orchestration, Network Management, Cloud Computing**

I. Introduction

The growing adoption of emerging applications, such as eXtended Reality (XR) and holographic tele-presence, imposes ultra-low latency and high bandwidth requirements. Thus, ensuring seamless immersive experiences is a major challenge [1]. These demands expose the limitations of today's cloud infrastructure, which spans heterogeneous and distributed computing resources, from edge nodes to the centralized cloud Data Centers (DCs) [2].

While traditional orchestration strategies often optimize for system-level Quality of Service (QoS) metrics, such as latency and throughput, these do not always guarantee a satisfactory Quality of Experience (QoE). Small variations in network conditions can lead to unexpected degradations in user experience [3]. This QoS-QoE discrepancy arises from the fact that QoE is influenced not only by network performance but also by other factors, including device capabilities, user context, and individual expectations, making it more nuanced and difficult to predict. As such, there is a growing need for novel orchestration strategies that jointly optimize for QoS and QoE, enabling a more human-centric approach to system management. This is especially relevant in dynamic 6G Compute Continuum (CC) environments, where fluctuating workloads and network conditions require adaptive user-aware orchestration, typically supported by cloud-native platforms such as Kubernetes (K8s) to enable distributed scalable deployments [4].

To address this challenge, this paper presents a Reinforcement Learning (RL)-based approach for the orchestration of XR applications in distributed 6G CC infrastructures. It uses a multi-objective reward function that balances latency, cost, fairness, and user QoE. A key novelty is integrating QoE models from prior subjective studies to guide the RL orchestrator toward efficient and user-centric policies. The main contributions of this work are: *1)* the ***gym-qoe* framework**: an open-source RL framework for QoE-aware service placement in 6G CC enabling to evaluate orchestration strategies under diverse scenarios; *2)* featuring a **Comprehensive Reward Function Design** that incorporates trade-offs among cost, latency, fairness and QoE, allowing quantification of conflicting objectives; *3)* an **Extensive Evaluation** of RL under different reward strategies and benchmark it against heuristic baselines, demonstrating the advantages of RL in managing dynamic orchestration within the CC.

The remainder of this paper is organized as follows: Sec. II reviews related work relevant to QoE-aware orchestration in 6G CC environments. Sec. III presents an overview of the envisioned 6G system architecture and its integration with cloud-native technologies. Sec. IV details the design of the RL approach, including the observation state, action space, and multi-objective reward formulation. Sec. V describes the experimental setup, including performance metrics. Sec. VI presents the results obtained under different reward strategies and compares RL with heuristic approaches. Finally, Sec. VII concludes this paper.

II. Background & Related Work

Cloud-Native (CN) XR [5] merges XR technologies with cloud infrastructures to deliver immersive experiences over the internet. Unlike traditional XR relying on local infrastructure, CN XR leverages scalability and elasticity by offloading computationally intensive tasks (e.g., sampling, encoding) to cloud servers, such that end users can access high-quality

978-3-903176-75-1 ©2025 IFIP

XR content on a wide range of devices without requiring specialized hardware [5]. **K8s** [6] is central to this paradigm, providing automated deployment, scaling, and management of containerized workloads, while ensuring high availability, fault tolerance, and efficient resource usage. Leveraging K8s allows XR workloads to be distributed across multiple nodes, to adapt dynamically to changing user demands, and to be continuously updated with minimal disruption, ultimately enhancing the scalability and reliability of immersive XR delivery [5].

Modeling QoE in interactive XR is challenging due to the multidimensional nature of human perception [7]. Prior work [8], [4] shows that QoE depends on a mix of technological, contextual, and physiological factors rather than a single parameter. While network Key Performance Indicators (KPIs) capture aspects, such as performance and device capability, human responses are crucial for reflecting subjective QoE dimensions.

QoE-aware orchestration has attracted increasing attention in edge-cloud computing and networked applications [9]. Prior studies explored user-centric network management using perception metrics [10]–[15]. Gramaglia et al. [10] designed and validated a multi-service 5G network with QoE-aware orchestration, stressing open-source solutions and highlighting multi-slice orchestration, Radio Access Network (RAN) slicing, and local breakout for diverse services. Alencar et al. [12] proposed Fog4VR, which selects optimal fog nodes based on delay, migration time, and resource use, improving cost, fairness, and QoE. Bagaa et al. [11] introduced a zero-touch Software-Defined Networking (SDN)/Network Function Virtualization (NFV)-based orchestration system for Internet of Things (IoT), reducing delay and ensuring trusted deployments.

Despite these advancements, existing solutions lack holistic orchestration mechanisms that jointly consider network conditions, computing resources, and user experience. In contrast, this paper proposes an RL-driven orchestration approach to dynamically deploy XR microservices based on network conditions, available computing resources, and predicted QoE scores. We incorporate multiple performance factors into the reward function, including deployment cost, latency, inequality, and QoE, to quantify the trade-offs among these objectives.

III. System Overview

Fig. 1 shows the envisioned 6G CC infrastructure built on the K8s orchestration platform. This cloud-native architecture deploys and manages XR applications while optimizing resources across edge, fog, and cloud. Unlike traditional clouds, fog and edge provide intermediate zones closer to users, reducing latency and improving responsiveness for XR.

In this work, XR applications are decomposed into microservices, each developed, deployed, and maintained independently. These run in K8s as *Pods*, the smallest deployable units. At the core of the architecture is an RL-based orchestration component that serves as the central decision-maker for runtime service placement and handling incoming XR requests. The RL orchestrator leverages a real-world dataset from our previous work [8], where the environment is dynamically updated with network conditions and subjective quality metrics. This dataset stems from a collaborative VR pizza-baking task, where participants coordinated actions via avatars under controlled network impairments (e.g., burst traffic, delays) to emulate latency spikes, jitter, and synchronization issues. QoE was measured on a 5-point Likert scale [16] based on participants' reported immersion.

On each XR request, the RL orchestrator decides whether to admit it and, if accepted, selects the optimal K8s cluster. A QoE assessment module, based on Machine Learning (ML) models trained on the dataset [8], estimates user-perceived quality to guide these decisions. This enables the RL agent to anticipate satisfaction, adapt policies, and establish a QoE-driven feedback loop aligned with user expectations. The next section provides further details about the RL design.

IV. Reinforcement Learning Design

The *gym-qoe* framework enables scalable and cost-effective training of RL algorithms, building on recent open-source environments for network orchestration [17]–[19]. Traditional OpenAI Gym-based environments are typically designed to speed up the training process of RL agents. As such, our *gym-qoe* framework supports multiple RL algorithms focused on generating an orchestration strategy using dynamic information from the collected dataset [8] as input. The framework updates the RL environment with network data (throughput, packet size, inter-arrival times) collected from a collaborative Virtual Reality (VR) pizza-baking experiments and dynamically adjusts CPU and memory at each location based on admitted requests and RL actions. The *gym-qoe* framework mimic the behavior of deployment requests within an edge-cloud scenario, thereby providing the RL agent with pertinent information about network conditions collected by previous experiments. Each XR request in the framework represents a collaborative pizza-baking session from the dataset, where the microservice allocation corresponds to the provisioning of an XR service instance (further details in [8]) that needs to be deployed to a given location if the request is admitted. The requests are randomly generated based on a given set of requests detailed in Sec. V. These requests have different resource requirements and latency thresholds.

A. Observation Space

Table I exhibits the observation space applied in the *gym-qoe* framework to characterize the environment at each step. It contains four main sets of metrics: *Request*, *Location*, *QoE*, and *Network*. **The *Request* set** defines the application's deployment requirements, including Central Processing Unit (CPU) and memory demands (ω_{cpu} and ω_{mem}), the latency threshold (Δ_r) that must be satisfied by the hosting location, and the expected duration (T_r) of each request r. Since the RL agent is triggered with each incoming deployment request, the time interval between consecutive steps varies dynamically. To account for this variability, T_r is integrated into the observation space, allowing the agent to learn and adapt to the environment's temporal dynamics, as shown in

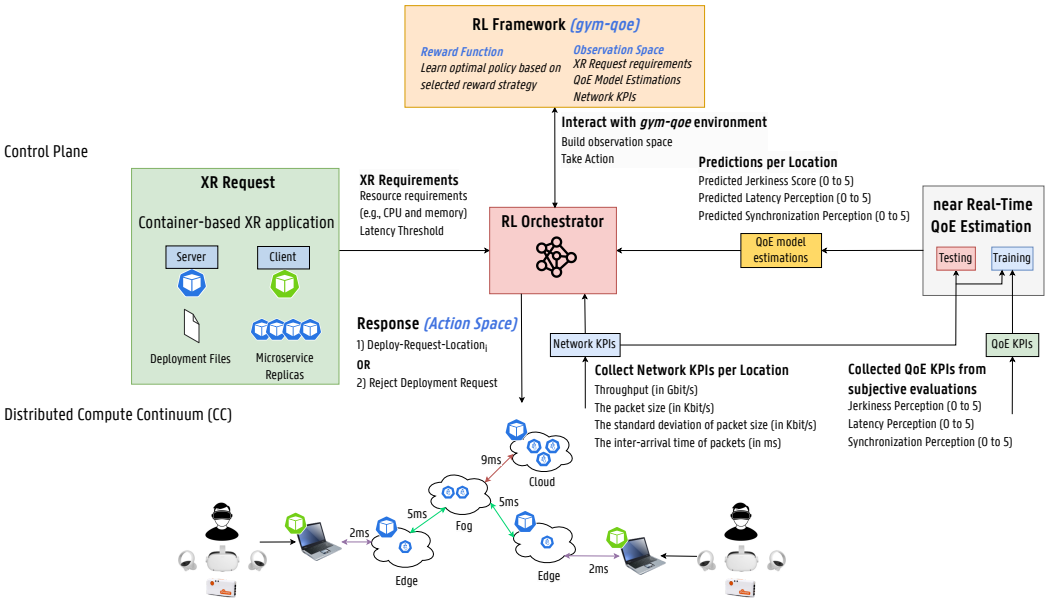


Fig. 1: High-level view of the deployment of a container-based XR application within the 6G CC.

TABLE I: Observation Space structure in *gym-qoe*.

Set	Metric	Description
Request	ω_{cpu}	The CPU request (in millicpu (m)).
	ω_{mem}	The memory request (in mebibyte (MiB)).
	Δ_{r}	The latency threshold (in ms).
	T_{r}	The execution time (in ms).
Location	Π_{cpu}	The location's cpu capacity (in m).
	Π_{mem}	The location's memory capacity (in MiB).
	Θ_{cpu}	The CPU allocated in the location (in m).
	Θ_{mem}	The memory allocated in the location (in MiB).
	δ_{l}	The processing latency of the location (in ms).
QoE	I_{jerk}	Jerkiness score (between 0-5).
	I_{sync}	Synchronization score (between 0-5).
	I_{lat}	Latency perception score (between 0-5).
Network	δ	The throughput (in Gbit/s).
	S	The packet size (in Kbits).
	S_{std}	The standard deviation of packet size (in Kbits).
	P	The inter-arrival time of packets (in ms).

TABLE II: The hardware configuration of each location based on Amazon EC2 On-Demand Pricing (Frankfurt Region - Europe) [20].

Location Type	Latency (β_l)	Amazon ($/h)	Cost (τ_c)	CPU (in m)	RAM (in MiB)
Cloud-T2	10 ms	2XL (0.3072)	64.0	8.0	32.0
Cloud-T1	8.5 ms	XL (0.1536)	32.0	4.0	16.0
Fog-T2	7.6 ms	L (0.0768)	16.0	2.0	8.0
Fog-T1	5.0 ms	M (0.0384)	8.0	2.0	4.0
Edge-T3	2.5 ms	S (0.0192)	4.0	2.0	2.0
Edge-T2	1.5 ms	Micro (0.0096)	2.0	1.0	1.0
Edge-T1	1.0 ms	Nano (0.0048)	1.0	2.0	0.5

previous work [21]. By explicitly incorporating inter-arrival times, the framework captures fluctuations in resource consumption across different locations, enhancing the ability of the agent to make informed allocation decisions based on evolving system states. The RL framework models a queuing system in which deployment requests arrive and are processed over time, following common practices used in previous works [18], [21]. Inter-arrival and service times follow exponential distributions to reflect realistic workload dynamics. The arrival time of the request is generated by adding a sampled inter-arrival interval to the current time, while the departure time is determined by adding a service duration to the arrival time. This queuing mechanism ensures realistic request behavior and efficient handling without overlapping requests.

The *Location* set represents the metrics describing the current state of the infrastructure, capturing both resource availability and network conditions. These metrics include the total CPU and memory capacities (Π_{cpu} and Π_{mem}), the currently allocated CPU and memory resources (Θ_{cpu} and Θ_{mem}), 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. Additionally, the processing latency (δ_l) at each location is influenced by the number of hosted requests, increasing by a predefined factor (i.e., 2 ms per active request). By incorporating these detailed infrastructure metrics, the *gym-qoe* framework provides a comprehensive view of resource availability and network conditions, enabling the RL agent to make more effective and adaptive orchestration decisions.

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

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

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 $[2.0, 64.0]$ 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 (β_l) that refers to the average latency between users and the selected deployment location. In our experiments, this variable varies between 1 to 10 milliseconds, depending on the type of the chosen location to host that particular request.

The *QoE* set adds perception-related metrics to the observation space, helping the RL agent choose deployment locations that maximize user QoE. Since QoE is subjective—it reflects the user's perceived experience rather than a direct system measurement. Thus, we adopt an estimation approach based on the QoE perception models proposed in [4]. These models predict user experience through distinct metrics—jerkiness (I_{jerk}), latency (I_{lat}), and synchronization (I_{sync})—each requiring a specific set of objective features, including network-level metrics such as throughput (δ), packet inter-arrival time (P), packet size (S) and its standard deviation (S_{std})— **the *Network* set**. Throughput measures the average data rate exchanged between server and clients during playthroughs at that location, while packet inter-arrival time indicates the average interval between consecutive packet transmissions.

Current QoE estimates alone ignore temporal dynamics, preventing the RL agent from recognizing trends and anticipating fluctuations. To capture history, we extend the observation space with a recursive moving average over the current state s_i and the previous state s_{i-1}, tracking the evolution of jerkiness, latency, and synchronization:

$$I_{\text{jerk}} = \frac{(i-1)\cdot I_{\text{jerk},s_{i-1}} + I_{\text{jerk},s_i}}{i} \tag{3}$$

$$I_{\text{lat}} = \frac{(i-1)\cdot I_{\text{lat},s_{i-1}} + I_{\text{lat},s_i}}{i} \tag{4}$$

$$I_{\text{sync}} = \frac{(i-1)\cdot I_{\text{sync},s_{i-1}} + I_{\text{sync},s_i}}{i} \tag{5}$$

By incorporating past observations, the agent gains a deeper understanding of QoE fluctuations over time, allowing it to anticipate variations and adapt its decisions accordingly. This integration of real-time and historical QoE makes orchestration more robust and user-centric.

B. Action Space

In *gym-qoe*, actions are discrete, with only a single action executed per timestep. The RL agent may *deploy* or *reject* a 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 number of possible actions equals the number of locations, with penalties applied for invalid choices [22]. A more advanced method, action masking [23], prevents the agent from selecting actions that are infeasible in the current state s. For example, the action masks for the location l and state s can be defined as:

$$\text{mask}(s)[l] = \begin{cases} \text{True} & \text{If location } l \text{ has enough resources} \\ \text{False} & \text{Otherwise} \end{cases} \tag{6}$$

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.

C. Reward function

The reward function is critical for steering the RL agent towards maximizing rewards by selecting optimal actions based on the current observation state. In this work, a multi-objective reward function (7) has been designed to incorporate four distinct objectives: cost-aware (8), latency-aware (9), inequality-aware (10), and QoE-aware (11). When the agent chooses to accept a deployment request, it is rewarded positively according to these objectives, with respective weights (ω_{cost}, ω_{lat}, ω_{ineq}, and ω_{qoe}), 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. The penalty mechanism is designed to strongly penalize the agent for avoidable rejections, encouraging more admission decisions.

$$r = \begin{cases} \omega_{\text{cost}} \cdot r_{\text{cost}} + \omega_{\text{lat}} \cdot r_{\text{lat}} + \\ \omega_{\text{ineq}} \cdot r_{\text{ineq}} + \omega_{\text{qoe}} \cdot r_{\text{qoe}} & \text{if request is accepted.} \\ -1 & \text{if request is rejected.} \end{cases} \tag{7}$$

$$r_{\text{cost}} = 1.0 - \Gamma_d \text{ with } \Gamma_d = \text{expected deployment cost} \tag{8}$$

$$r_{\text{latency}} = 1.0 - \lambda_d \text{ with } \lambda_d = \text{expected total latency} \tag{9}$$

$$r_{\text{inequality}} = 1.0 - G \text{ with } G = \text{Gini coefficient} \tag{10}$$

$$r_{\text{QoE}} = Q \text{ with } Q = \text{perceived QoE by accepted user} \tag{11}$$

The **cost-aware** function guides the agent to deploy requests or migrate in a manner that minimizes allocation costs. Cloud locations usually have higher deployment costs and latency than fog or edge types. To maximize reward, the RL agent prioritizes edge and fog locations, but their smaller resource capacity compared to the cloud can limit the number of supported requests. The **latency-aware** function aims to minimize the expected latency of a deployment request. The expected latency (λ_d) is computed based on several latency factors, considering the specific location hosting the request. The latency-aware function minimizes overall latency (12) by favoring locations with low processing and access delays.

$$\underbrace{\lambda_d}_{\text{Total Latency (in ms)}} = \underbrace{\delta_l}_{\text{Proc. Latency (in ms)}} + \underbrace{\beta_l}_{\text{Access Latency (in ms)}} \tag{12}$$

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) and 1 indicates perfect inequality (all requests deployed

TABLE III: The evaluated reward strategies.

Name	ω_{cost}	ω_{lat}	ω_{ineq}	ω_{qoe}
Cost	1.0	0.0	0.0	0.0
Latency	0.0	1.0	0.0	0.0
Inequality	0.0	0.0	1.0	0.0
Qoe	0.0	0.0	0.0	1.0
Balanced	0.25	0.25	0.25	0.25

in one location). A lower G indicates then a more equitable distribution. As such, G is an accurate measure of inequality in a distribution, calculated as:

$$G = \frac{\sum_{i=1}^{l} \sum_{j=1}^{l} |L_i - L_j|}{2l^2 \bar{L}} \tag{13}$$

where:

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.

Including G in the reward promotes a fairer distribution of resources between deployment locations. The RL agent may favor high-QoE or low-cost locations, causing imbalance and under-utilization elsewhere. By adding inequality to the reward, we capture the trade-offs between competing objectives.

Lastly, the **QoE-aware** function guides the agent to deploy requests towards maximizing the QoE of each user request by incorporating the multiple QoE criteria, including perceived latency (I_{latency}), perceived jerkiness (I_{jerk}), and perceived synchronization (I_{sync}), with all being equally weighted $w_{\text{l}} = w_{\text{j}} = w_{\text{s}} = 1$.

$$\underbrace{Q}_{\text{Per. QoE}} = w_{\text{l}}.\underbrace{(5 - I_{\text{lat}})}_{\text{Perc. lat.}} + w_{\text{j}}.\underbrace{(5 - I_{\text{jerk}})}_{\text{Perc. jerk.}} + w_{\text{s}}.\underbrace{(I_{\text{sync}})}_{\text{Perc. sync.}} \tag{14}$$

where:

$w_{\text{l}}, w_{\text{j}}, w_{\text{s}}$ are the weighting coefficients.

I_{lat} is the user's reported perceived latency (0-5).

I_{jerk} is the user's perception of motion smoothness (0-5).

I_{sync} is the user's perception of synchronization (0-5).

Q is the perceived QoE by the user (0-15).

V. Experimental Setup

The *gym-qoe* framework is developed in Python for seamless integration with both the OpenAI Gym and Stable Baselines 3 libraries, enabling efficient RL-based experimentation and model training. Each evaluation episode spans 100 steps, with the RL agent maximizing rewards over multiple deployment requests. When the agent selects an available location to host the request, the CPU and memory usage of that location increase. When a request is completed, resource usage drops based on its mean service duration. The expected duration of the request (T_r) is then randomized around the mean service duration, ensuring that the RL agent encounters varied request patterns across consecutive episodes. During training, four locations are considered, with the type of each location being randomized. The RL algorithm is trained on over 2000 episodes, utilizing a 14-core Intel i7-12700H CPU @ 4.7 GHz processor with 16 GB of memory.

Multiple reward strategies are evaluated, as shown in Table III. Some strategies target a single objective—such as *Cost*, *Latency*, *Inequality*, or *QoE*—while a balanced strategy (*Balanced*) assigns equal weight to all four objectives. Through this comparative evaluation, the *gym-qoe* framework aims to assess how different reward formulations influence deployment decisions in distributed computing environments. **One notable RL algorithm**, known as Maskable Proximal Policy Optimization (MaskPPO) [23], is evaluated based on its reliable implementation in Python within the stable baselines 3 framework [24]. MaskPPO extends the standard PPO algorithm by incorporating invalid action masking, allowing the RL agent to ignore infeasible actions while preserving the core behavior of PPO. In this study, additional RL algorithms are evaluated, but our findings reveal that MaskPPO consistently outperforms the other RL methods. Thus, we focus our evaluation on assessing MaskPPO performance under various reward strategies. The objective is to analyze the effectiveness and robustness of several strategies, thereby providing a nuanced understanding of their applicability in real-world deployment scenarios. Through this evaluation, we aim to offer valuable insights for researchers and practitioners seeking to leverage RL techniques for efficient deployment strategies within the CC.

Various application requests are used in the evaluation of the *gym-qoe* framework, representing a broad range of use cases, ensuring that the RL agent encounters diverse patterns in computing demands (i.e., CPU from $[0.015, 0.20]$ millicpu and RAM from $[0.016, 0.70]$ MiB) and latency thresholds (i.e., from $[20, 200]$ milliseconds). This enables a thorough assessment of the RL agent's capacity to handle diverse computational loads and latency demands in dynamic environments. These requirements are designed to be generic and applicable to a wide range of use cases. **Several performance metrics** are used to evaluate the performance of the RL algorithm:

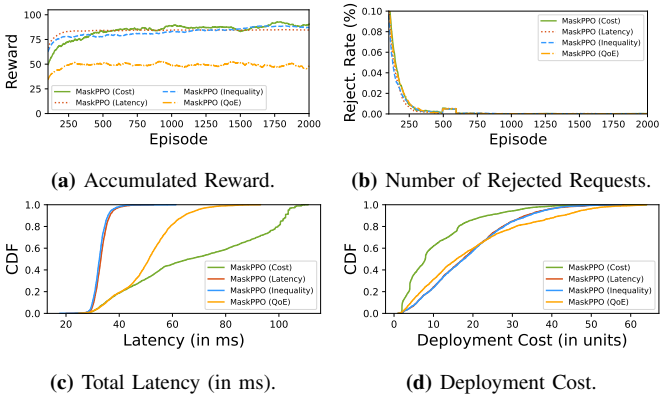
- **Accumulated reward** during each episode. It refers to the total sum of rewards obtained by MaskPPO.
- **Proportion of rejected requests** expressed as $[0, 1]$.
- **Average total latency (in ms)** for each accepted request.
- **Average deployment cost** incurred for each request.
- **G (Gini Coefficient)** highlighting the inequality of the deployment scheme, with values ranging from 0 (perfect equality) to 1 (maximum inequality), helping to assess fairness in the distribution of requests.
- **Accumulated QoE (0-15)** experienced by each user.

Three heuristic baselines are assessed to compare its performance against the RL algorithm:

- **CPU-Greedy**: assigns the request to the location with the lowest resource consumption in terms of CPU usage.
- **Latency-Greedy**: assigns the request to a random location while adhering to the specified latency threshold, focusing on both the access and processing latency.

TABLE IV: The different configurations tested in the experiments.

Name	Description
False – False (FF)	Both QoE and network KPIs are excluded.
False – True (FT)	QoE KPIs are excluded.
True – False (TF)	Network KPIs are excluded.
True – True (TT)	Both QoE and network KPIs are included.



(a) Accumulated Reward. **(b)** Number of Rejected Requests.

(c) Total Latency (in ms). **(d)** Deployment Cost.

Fig. 2: The training results for *False – False* scenario.

- **Cost-Greedy**: assigns the request to the location with the lowest deployment cost according to the type of location.

Various observation spaces are evaluated to understand the performance of the RL agent when QoE or network KPIs are unavailable, as shown in Table IV. The evaluation focuses on the following key research questions:

- ***RQ1 - Does RL perform better when both QoE and network KPIs are included in the observation space?*** This study aims to identify the most effective configuration of the observation space to maximize QoE while maintaining a high acceptance rate.
- ***RQ2 - Do network KPIs provide benefit in improving the performance of RL when maximizing QoE?*** This work investigates whether including network-level metrics enhances the decision-making of the RL agent, or if QoE metrics alone are sufficient for effective orchestration.

VI. Results

This section presents the performance results under different reward strategies (Table III) and observation state configurations (Table IV). Fig. 2 shows the performance of different reward strategies over 2000 training episodes in the FF configuration, smoothed with a 100-episode window. Most strategies converge between the 300th and 600th episode, with slight reward gains beyond. MaskPPO (*Cost*) achieves the highest average reward (Fig. 2a) and effectively minimizes rejections across all strategies (Fig. 2b), reaching nearly 0% by the end. *Latency* strategy yields consistently lower total latency (Fig. 2c), while *Cost*- and *QoE*-driven strategies incur higher latency. In contrast, *Cost* strategy results in the lowest deployment costs at the end (Fig. 2d). Regarding the other observation state configurations (FT, TF, TT), training results were similar to the FF case. Due to space constraints, we have omitted these plots and focused on the testing phase to highlight key differences. Notably, when network or QoE KPIs

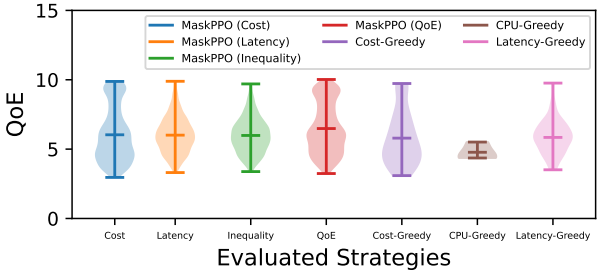


Fig. 3: The average QoE obtained for the TT configuration.

are included, the RL agent learns more effective placement decisions that optimize QoE compared to the FF configuration.

During the testing phase, each strategy was executed over 100 episodes using the saved configuration from the training phase (over 2000 episodes). Table V reports performance across all observation state configurations, showing that RL strategies consistently outperform heuristics in balancing objectives. The *Cost* strategy yielded the lowest deployment cost (6.4–8.3), confirming its strength in cost optimization. However, it also exhibited high latency (78.9-92.3 ms) and low fairness (0.6-0.7), indicating that minimizing cost can compromise both user experience and equitable resource distribution. The *Latency* strategy reduced latency and kept 0% rejections, but resulted in higher costs (15.7–16.4). The *Inequality* strategy offered one of the most balanced results, combining low latency, comparable to the *Latency* strategy, with excellent fairness (0.1). In the TT configuration, the *QoE* strategy achieved the highest score (7.0) but with high cost (20.1) and moderate latency (65.9), as concentrating requests at optimal locations increased resource contention. In contrast, **greedy baselines** showed extreme trade-offs: Latency-Greedy achieved the lowest latency (26.5 ms) but had a 20% rejection rate due to resource exhaustion. Cost-Greedy was the most cost-efficient (4.5 units) but suffered from high latency (80.1 ms) and low QoE (5.8). The CPU-Greedy baseline performed worst in fairness and QoE, demonstrating the limitations of purely resource-driven policies.

Overall, RL-based strategies demonstrated versatility, with specific configurations excelling in different performance aspects, such as low latency, cost efficiency, equitable resource distribution, and high QoE. The Cost-Greedy heuristic remains the most cost-effective but at the expense of increased latency, while the Latency-Greedy heuristic achieves the lowest latency but with a slightly higher request rejection rate. In addition, incorporating network and QoE KPIs in the observation space as in the TT configuration consistently achieved higher QoE without rejections, and more balanced trade-offs, affirming the value of RL strategies for orchestration in the 6G CC.

In summary, the experimental results confirm that the proposed RL-based approach effectively optimizes XR orchestration in 6G CC. The **reward function design** balanced cost, latency and QoE, with the agent adapting decisions to optimize its main objective while maintaining other metrics acceptable. RL-based orchestration also achieved clear QoE

TABLE V: Summary of RL agent performance across different observation space configurations. Incorporating network and QoE KPIs in the observation space consistently achieved higher QoE compared to their configurations.

Strategy	False – False (FF)				False – True (FT)				True – False (TF)				True – True (TT)			
	QoE	Cost	Tot. Latency	G	QoE	Cost	Tot. Latency	G	QoE	Cost	Tot. Latency	G	QoE	Cost	Tot. Latency	G
Cost	5.9 ± 0.4	**6.4 ± 1.0**	92.3 ± 3.6	0.7 ± 0.0	6.1 ± 0.4	**8.3 ± 1.5**	85.6 ± 4.2	0.7 ± 0.0	6.0 ± 0.4	8.3 ± 1.3	79.5 ± 4.4	0.6 ± 0.0	5.8 ± 0.4	**7.4 ± 1.2**	78.9 ± 3.7	0.6 ± 0.0
Latency	6.2 ± 0.3	16.0 ± 1.9	**32.3 ± 0.5**	**0.1 ± 0.0**	6.1 ± 0.2	15.8 ± 1.8	32.0 ± 0.5	**0.1 ± 0.0**	5.9 ± 0.2	15.7 ± 1.8	32.1 ± 0.5	**0.1 ± 0.0**	6.1 ± 0.2	15.8 ± 1.8	31.9 ± 0.5	**0.1 ± 0.0**
Inequality	**6.3 ± 0.2**	16.5 ± 2.0	**31.2 ± 0.5**	**0.1 ± 0.0**	5.9 ± 0.2	16.4 ± 1.9	**31.6 ± 0.5**	**0.1 ± 0.0**	5.9 ± 0.2	16.1 ± 1.9	**31.5 ± 0.5**	**0.1 ± 0.0**	6.0 ± 0.2	16.4 ± 1.9	31.5 ± 0.5	**0.1 ± 0.0**
QoE	5.8 ± 0.3	18.5 ± 2.9	51.1 ± 1.2	0.5 ± 0.0	**6.7 ± 0.4**	19.9 ± 3.8	84.6 ± 3.4	0.7 ± 0.0	**6.5 ± 0.4**	20.7 ± 3.9	79.0 ± 3.9	0.6 ± 0.0	**7.0 ± 0.4**	20.1 ± 3.4	65.9 ± 3.8	0.5 ± 0.0
Balanced	6.0 ± 0.3	11.3 ± 1.2	36.1 ± 1.0	0.2 ± 0.0	6.0 ± 0.3	10.6 ± 1.2	37.3 ± 1.2	0.3 ± 0.0	6.1 ± 0.3	11.7 ± 1.3	36.0 ± 0.9	0.2 ± 0.0	6.1 ± 0.3	11.0 ± 1.2	35.8 ± 0.9	0.2 ± 0.0
CPU-Greedy	QoE: 4.8 ± 0.6 Cost: 48.8 ± 24.3 Tot. Latency: 93.0 ± 28.8 G: 0.7 ± 0.1															
Latency-Greedy	QoE: **5.8 ± 0.2** Cost: 18.2 ± 2.0 Tot. Latency: **26.5 ± 0.3** G: **0.1 ± 0.0**															
Cost-Greedy	QoE: **5.8 ± 0.4** Cost: **4.5 ± 1.0** Tot. Latency: 80.1 ± 5.0 G: 0.6 ± 0.0															

gains over heuristics, with MaskPPO consistently delivering higher QoE for admitted users. In addition, the agent maintained a 100% acceptance rate, proving its robustness and scalability. Regarding the research questions, including both QoE and network KPIs in the observation space improved decision-making, yielding a 21.3% QoE gain over heuristics and 14.7% over the Balanced strategy. While QoE is the main driver of user satisfaction, network KPIs ensured stable and efficient allocation, preventing suboptimal choices under dynamic conditions.

Overall, these findings provide a foundational understanding of how QoE interacts with network KPIs in RL-based orchestration, informing future research directions in intelligent network and service management.

VII. Conclusions

As holographic and XR advance immersive experiences, optimizing end-user QoE in 6G networks presents both challenges and opportunities. This work proposes a RL-driven orchestration strategy that integrates QoE and network KPIs to support low-latency service delivery for upcoming 6G applications. Through extensive evaluation, RL shows strong adaptability while achieving a 21.3% QoE improvement over heuristics and 14.7% over a balanced placement strategy, while maintaining a 100% acceptance rate. These results highlight the performance of RL in handling conflicting objectives under dynamic network conditions. Future work will explore alternative RL algorithms, such as multi-agent RL, to evaluate the performance of these techniques when multiple RL agents can either cooperate or compete, collectively optimizing service placement decisions.

Acknowledgment

José Santos is funded by the Research Foundation Flanders (FWO), grant nrs. 1299323N and 1253226N. This work is partly funded by FWO WaveVR project, grant nr. G034322N.

References

[1] A. M. Aslam *et al.*, "Metaverse for 6g and beyond: The next revolution and deployment challenges," *IEEE Internet of Things Magazine*, vol. 6, no. 1, pp. 32–39, 2023.

[2] J. Santos *et al.*, "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] T. Hoßfeld *et al.*, "Qoe beyond the mos: an in-depth look at qoe via better metrics and their relation to mos," *Quality and User Experience*, vol. 1, pp. 1–23, 2016.

[4] J. Santos *et al.*, "Leveraging user perception for 6g edge-cloud orchestration of networked extended reality," *IEEE Communications Magazine*, 2025.

[5] J. Santos, "Towards cloud-native virtual reality applications: State-of-the-art and open challenges," in *NGMSE2024, the Next-Generation Multimedia Services at the Edge: Leveraging 5G and Beyond workshop, co-located with ISCC2024*, 2024.

[6] M. Luksa, *Kubernetes in action*. Simon and Schuster, 2017.

[7] S. Van Damme *et al.*, "Human-centric quality management of immersive multimedia applications," in *2020 6th IEEE Conference on Network Softwarization (NetSoft)*, 2020, pp. 57–64.

[8] S. Van Damme et al, "Impact of latency on qoe, performance, and collaboration in interactive multi-user virtual reality," *Applied Sciences*, vol. 14, no. 6, p. 2290, 2024.

[9] A. A. Barakabitze *et al.*, "Qoe management of multimedia streaming services in future networks: A tutorial and survey," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 1, pp. 526–565, 2019.

[10] M. Gramaglia *et al.*, "Design and validation of a multi-service 5g network with qoe-aware orchestration," in *Proceedings of the 12th International Workshop on Wireless Network Testbeds, Experimental Evaluation & Characterization*, 2018, pp. 11–18.

[11] M. Bagaa *et al.*, "Qos and resource-aware security orchestration and life cycle management," *IEEE Transactions on Mobile Computing*, vol. 21, no. 8, pp. 2978–2993, 2020.

[12] D. Alencar *et al.*, "Dynamic microservice allocation for virtual reality distribution with qoe support," *IEEE Transactions on Network and Service Management*, vol. 19, no. 1, pp. 729–740, 2021.

[13] M. Bosk *et al.*, "Using 5g qos mechanisms to achieve qoe-aware resource allocation," in *2021 17th International Conference on Network and Service Management (CNSM)*. IEEE, 2021, pp. 283–291.

[14] H. Luo *et al.*, "Resource orchestration at the edge: Intelligent management of mmwave ran and gaming application qoe enhancement," *IEEE Transactions on Network and Service Management*, vol. 20, no. 1, pp. 385–399, 2022.

[15] M. Carvalho *et al.*, "Qoe estimation across different cloud gaming services using transfer learning," *IEEE Transactions on Network and Service Management*, 2024.

[16] R. Likert, "A technique for the measurement of attitudes," *Archives of Psychology*, vol. 22, no. 140, pp. 1–55, 1932.

[17] J. Santos *et al.*, "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.

[18] A. F. Ocampo *et al.*, "Reinforcement learning-driven service placement in 6g networks across the compute continuum," in *2024 20th International Conference on Network and Service Management (CNSM)*. IEEE, 2024, pp. 1–9.

[19] J. Santos *et al.*, "Hephaestusforge: Optimal microservice deployment across the compute continuum via reinforcement learning," *Future Generation Computer Systems*, p. 107680, 2025.

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

[21] J. Santos *et al.*, "Efficient microservice deployment in kubernetes multi-clusters through reinforcement learning," in *NOMS 2024-2024 IEEE Network Operations and Management Symposium*. IEEE, 2024, pp. 1–9.

[22] 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, 2021.

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

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