GaiaRL: Multi-Objective RL for Sustainable VNF Placement in the Compute Continuum

Mattia Zaccarini†, José Santos*, Filippo Poltronieri†, Nicola Di Cicco‡,
Mauro Tortonesi†, Cesare Stefanelli†, Filip de Turck*
† Distributed Systems Research Group, University of Ferrara, Ferrara, Italy
* Ghent University - imec, IDLab, Department of Information Technology, Gent, Belgium
‡ Optit S.r.l., Bologna, Italy

***Abstract*—The increasing deployment of Virtual Network Functions (VNFs) across the Compute Continuum (CC)—spanning cloud, fog, and edge environments— has introduced new challenges in orchestrating network services efficiently and sustainably. As network infrastructures scale to support latency-sensitive and resource-intensive applications, such as eXtended Reality (XR) and Generative Artificial Intelligence (GenAI) applications, existing orchestration strategies often prioritize performance while overlooking energy consumption and carbon footprint, leading to environmentally costly outcomes. Motivated by the urgent need for greener network management solutions, this work proposes *GaiaRL*, a Multi-Objective Reinforcement Learning (MORL) framework for sustainable VNF placement in distributed edge-to-cloud environments. GaiaRL formulates the orchestration problem as a sequential decision-making problem, balancing conflicting objectives such as deployment cost, latency, energy consumption, and fair resource utilization. We implement and evaluate multiple MORL algorithms—alongside a novel Geometric Pareto Q-Learning (GeoPQL) method—and compare them against single-objective baselines from previous work. Experiments using realistic 5G deployments across compute continuum scenarios show that MORL algorithms better balance the trade-offs between different objectives and drive energy-aware orchestration in modern computing scenarios, paving the way toward sustainable network management in 6G and beyond.**

***Index Terms*—Multi-Objective Optimization, Reinforcement Learning, Virtual Network Functions, Compute Continuum**

I. Introduction

The rapid evolution of network infrastructures towards cloud-native and edge computing paradigms has revolutionized the deployment of network services [1]. Virtual Network Functions (VNFs) play a critical role in this transformation, enabling flexible, programmable, and scalable service deployment across heterogeneous distributed clouds [2]. In addition, emerging latency-sensitive and resource-intensive applications, such as eXtended Reality (XR) [3] and Generative Artificial Intelligence (GenAI) [4], are placing unprecedented demands on network infrastructure and increasing pressure on orchestration frameworks to make intelligent, efficient service placement decisions.

In addition, the escalating energy footprint of digital infrastructure and the pressing need for sustainability reinforce the urgency of integrating sustainability into network orchestration strategies [5]. Data Centers (DCs) and networking equipment already account for approximately 2% of global energy consumption, with projections estimating this number could rise to at least 8% by 2030 [6]. This trend is largely driven by the rapid proliferation of AI workloads, immersive media, and the growing demand for ubiquitous connectivity. Without energy-aware orchestration mechanisms, such growth threatens to undermine global carbon reduction efforts and significantly increase operational expenditures for service providers. Beyond its environmental relevance, sustainable orchestration offers tangible advantages, including energy efficiency and extended hardware lifespans, positioning sustainability as a strategic and economic necessity [7].

Traditional orchestration approaches have predominantly focused on optimizing a single objective, such as minimizing latency or deployment cost. However, this narrow focus often results in suboptimal decisions when considering other critical performance factors, leading to increased operational expenses and inefficient resource utilization. The orchestration problem is inherently multi-faceted, requiring the joint optimization of multiple, often conflicting, objectives, including latency, cost, energy consumption, and resource fairness [8]. This complexity calls for Multi-Objective Optimization (MOO) techniques that can effectively steer trade-offs among these dimensions [9]. Although linear scalarization methods are commonly employed to reduce the problem's dimensionality [10], they frequently fail to capture the diversity of trade-offs present in the solution space. In contrast, MOO offers a more expressive decision-making framework by identifying the Pareto Front (PF), which represents the set of non-dominated solutions that achieve optimal balance across competing objectives.

This paper proposes *GaiaRL*[1], a Multi-Objective Reinforcement Learning (MORL) framework designed to address the complex trade-offs inherent in Virtual Network Function (VNF) placement. *GaiaRL* aims to harmonize conflicting objectives, including deployment cost, latency, energy consumption, and fair resource utilization to achieve sustainable orchestration practices in distributed cloud environments—the Compute Continuum (CC) [11]. Our approach formulates

[1]*Gaia*, the primordial Greek goddess personifying the Earth. As a symbol of life, balance, and natural sustainability, *Gaia* reflects the core objective of our work: achieving efficient and environmentally responsible orchestration of network functions.

VNF orchestration as a sequential decision-making problem. We leverage MORL algorithms to learn policies that balance performance and sustainability objectives, adapting to dynamic network conditions and diverse workload profiles. We evaluate *GaiaRL* against single-objective Reinforcement Learning (RL) baselines from our prior work [8], [9]. In a realistic 5G core deployment scenario, we demonstrate that *GaiaRL* achieves substantial gains – minimizing power consumption while maintaining, or even improving, key performance metrics.

The main contributions of this work are the following:

- **Implementation of MORL algorithms**: We investigate the practical application of two MORL algorithms, evaluating their applicability and implementation maturity. Also, we propose a novel algorithm, Geometric Pareto Q-Learning (GeoPQL), which leverages geometric principles within the Pareto Q-Learning (PQL) framework to enhance its suitability for real-world optimization problems. The MORL framework, *GaiaRL*[2], has been open-sourced to enable other researchers to build upon our work.
- **Extensive Evaluation:** We conduct a comprehensive evaluation of the VNF placement problem using the open-source 5G core[3] VNFs, considering realistic computational and latency requirements. Furthermore, we compare the performance of single-objective RL algorithms with MORL methods across both *3*-objective and *4*-objective scenarios.

The remainder of this paper is organized as follows: Sec. II discusses relevant works in the field of MOO for network and service management. Sec. III provides a background on MORL and Sec. IV presents the *GaiaRL* framework to solve the VNF orchestration problem. Then, Sec. V details the experimental setup, while Sec. VI presents the results for three (*3*-) and four (*4*-) objective scenarios. Lastly, Sec. VII concludes this paper.

II. Background & Related Work

MOO for Network Management. The orchestration of VNFs in distributed cloud environments has gained significant attention in recent years, driven by the growing complexity and scale of modern networked services [12]–[17]. VNFs must be efficiently deployed, migrated, and scaled across heterogeneous infrastructures to meet diverse requirements such as latency, reliability, energy consumption, and operational cost. Given the inherent trade-offs between objectives, MOO enables the exploration of solutions that satisfy different stakeholder preferences rather than a single global optimum. Traditional techniques such as Integer Linear Programming (ILP) and metaheuristics have shown success in small- to medium-scale scenarios [9]. For example, Kaur et al. [12] employed ϵ-NSGA-II to balance energy consumption and deployment efficiency in multi-domain Software Defined Networking (SDN) environments, while Yao et al. [13] used a best-fit

[2]https://github.com/mattiazaccarini/morl_baselines_karmada
[3]https://free5gc.org/

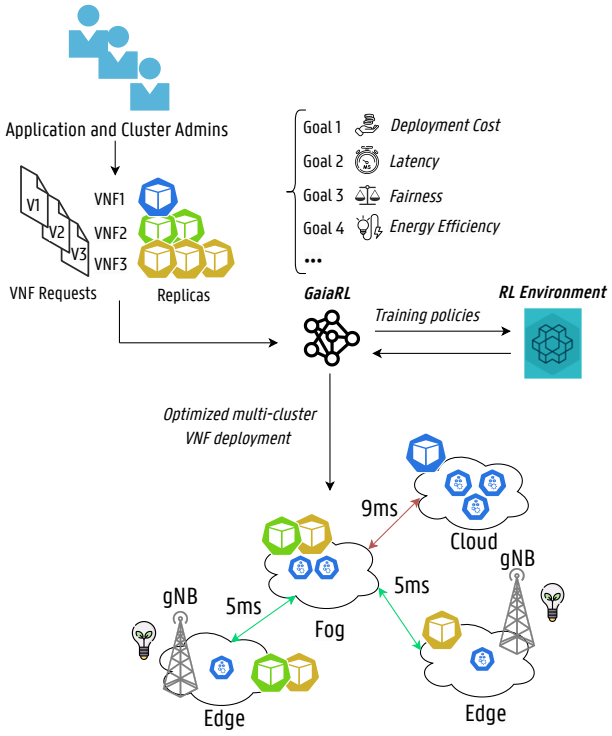


Fig. 1. An overview of the proposed *GaiaRL* framework.

decreasing heuristic for reliability and cost optimization in fog computing. However, ILP-based approaches struggle with scalability due to their high computational overhead, especially under real-time constraints or large-scale deployments [14]. To mitigate this, hybrid metaheuristics have been proposed, such as Particle Swarm Optimization (PSO) combined with Artificial Fish Swarm optimization [15] and Grey Wolf Optimization (GWO)-Ant Colony Optimization (ACO) variants for energy-aware Virtual Machine (VM) placement [17]. Although effective in specific environments, these solutions often rely on static approximations (e.g., weighted sums, lexicographic orderings) and are ill-suited for dynamically evolving network conditions or highly variable workloads. Moreover, most existing work focuses on isolated DC environments and lacks a holistic view of sustainability in orchestration.

RL in Network Orchestration. Recent advances in RL have opened new avenues for dynamic network management, where orchestration decisions are made through interaction with the environment rather than pre-defined rules based on thresholds. In the past few years, RL has been successfully applied to VNF placement [18]–[20], network slicing [21], and traffic engineering [22], demonstrating the potential for adaptive and real-time decision-making. In particular, the authors in [18] use federated RL to improve resource utilization in Network Function Virtualization (NFV)-enabled networks, while [19] employ Deep Learning (DL) models with the twist of ensemble techniques for automated VNF orchestration. Despite their promise, these approaches are typically single-objective, aiming to optimize a single metric (e.g., latency), and often neglect critical concerns such as energy efficiency or

carbon footprint. Recent efforts have begun to explore MORL to approximate the PF of non-dominated solutions, allowing for trade-offs between conflicting goals. However, applications of MORL in the context of sustainable VNF orchestration remain limited [23], [24].

To extend the current state-of-the-art, this paper introduces *GaiaRL* (illustrated in Fig .1), a MORL framework for sustainable VNF placement that explicitly balances deployment cost, latency, energy consumption, and resource fairness. Building further on our previous works that either simplified Multi-Objective (MO) problems via scalarization [25] or focused solely on performance factors [8], [9], *GaiaRL* leverages MORL's capacity to provide flexible policies and diverse trade-offs, enabling operators and cloud providers to adapt to different goals without re-engineering the learning pipeline. In summary, *GaiaRL* addresses several limitations of existing methods:

- **Scalability to high-dimensional objectives and constraints:** Traditional reward shaping often requires extensive domain knowledge and manual tuning, which becomes impractical for problems with many objectives or dynamic constraints. MORL algorithms, and GeoPQL in particular, systematically explore the PF without relying on hand-crafted reward combinations.
- **Structured exploration of the PF**: Unlike standard MORL algorithms that may converge to only a subset of Pareto-optimal policies depending on reward weights, our approach ensures more uniform coverage of the PF, enabling better decision-making in scenarios with competing objectives.
- **Adaptability to changing environments:** Reward shaping typically encodes static trade-offs, whereas GeoPQL can dynamically adapt to changes in the environment or objectives, making it more robust in online or real-time settings.

III. Multi-Objective Reinforcement Learning

MORL enables the definition of multi-objective reward functions, which are much more informative than a single scalar reward, where multiple components need to be scalarized as in single-objective RL. This leads to the creation of multiple optimized policies that can be selected according to the specific needs, such as latency minimization or cost minimization. To use MORL, a MO variant of a Markov Decision Process (MDP), called MOMDP must be defined [26]. Specifically, a MOMDP is described as a tuple

$$< S, A, T, R, \mu, \gamma >$$

where:

- S is a finite set of states.
- A is a finite set of actions.
- $T : S \times A \times S \rightarrow [0, 1]$ is a transition function which specifies the probability of the next state for each state, action, and next state.
- $R : S \times A \times S \rightarrow \mathbb{R}^n$ is a reward function that defines a vector of *n* rewards (one for each objective). This differs from the scalar values that are typically used as rewards in single-objective definitions.
- $\mu : S \rightarrow [0, 1]$ is the probability distribution over the initial states.
- $\gamma \in [0, 1)$ is the discount factor that specifies the relevance of immediate rewards.

Similarly to a MDP, the expected cumulative discounted reward can be calculated through a value function, defined as follows:

$$V^{\pi} = E[\sum_{k=0}^{\inf} \gamma^k r_{k+1} | \pi]$$

where π corresponds to a policy and r_t is the rewards vector received at time *t*. If the single-objective formulation is considered, the ordering is complete for any state, i.e., given $V^{\pi}(s)$ and $V^{\pi'}(s')$, one must be less than, equal to, or greater than the other. This is no longer true regarding the MOMDP formulation, where the vector structure of $V^{\pi}(s)$ provides only a partial ordering. $V^{\pi}(s)$ may be greater than $V^{\pi'}$ considering an objective *i*, but the opposite for a different objective *j*. Therefore, it is not possible to identify an optimal value without additional information on how to prioritize objectives. To address this problem, users typically rely on a function called *utility function* $U : \mathbb{R}^n \rightarrow \mathbb{R}$, that maps vector-valued returns to scalar values, enabling an explicit preference over different and contrasting objectives. Depending on the characteristics of the scenario, a priori scalarization might be possible or not, thus leading to different planning and learning phases in which either a single policy or a solution set of multiple policies is found. [27].

Since we aim to obtain as much value as possible from each objective, we can assume monotonicity in the utility function as one of the minimal assumptions for MORL. Specifically, a monotonically increasing utility function *u* could be defined as follows:

$$(\forall i : V_i^{\pi} \geq V_i^{\pi'}) \wedge (\exists i : V_i^{\pi} > V_i^{\pi'}) \rightarrow u(V_i^{\pi}) \geq u(V_i^{\pi'})$$

In other words, if a policy increases for one or multiple of its objectives without decreasing any of the other objectives, the relative utility function will have the same behavior. It has been proven that if the utility function *u* is monotonically increasing, the corresponding PF is a non-dominated set *U* [26], i.e., a subset of all possible policies Π where there exists at least one utility function that maximizes the scalar value:

$$U(\Pi) = \{\pi \in \Pi | \exists u, \forall \pi' \in \Pi : u(V_i^{\pi}) \geq u(V_i^{\pi'}\}$$

$$PF(\Pi) = \{\pi \in \Pi | \nexists \ \pi' \in \Pi : V_i^{\pi'} \succ_P V_i^{\pi}\}$$

where the dominance relation is analogous to dominance in multi-objective optimization [9]. As a result of these considerations, the PF corresponds to a set that includes policies such that there exist no other policies with better or equal values in all objectives.

Because there might exist multiple policies which can lead to the same value function, our objective is to consider the

smallest set of policies whose value functions correspond to the PF, called *Pareto Coverage Set*:

$$PCS(\Pi) = \{\pi \in PF(\Pi) | \forall \pi' \in \Pi, \exists \pi : V^{\pi} \succ_P V^{\pi'}\}$$

While many algorithms generate estimates of value vectors, it is important to note that most of the time RL leverages approximate and stochastic evaluations to do so [28]. These aspects can lead to high variance or systematic bias, especially when using techniques such as neural networks to perform function approximations. In fact, they tend to introduce approximation errors or uncertainties, making the identification of reliable Pareto-optimal policies more difficult. Therefore, the algorithms that we included in this work extract the policies that form the $PCS(\Pi)$ and then execute a separate policy evaluation to mitigate estimation errors and make policy selection more reliable.

A. MORL Algorithms

Multi-Policy Multi-Objective Q-Learning (MPMOQ) [29]: This algorithm is based on a basic Q-learning strategy for MO settings. Typically, these kinds of approach gather the $\hat{\text{Q}}$-values for each objective and action. Then, a scalarization function, which could be both linear or non-linear, is employed to adjust the MO problem into a single-objective one and store the evaluations required to select actions in a specific state properly. Once scalarized, one of the easiest and most effective ways to implement the action selection mechanism is to adopt an ϵ-greedy strategy [30]. In the MPMOQ algorithm, the typical Q-table is extended with values that are updated for each objective individually according to the following rule:

$$\begin{aligned}\hat{Q}(s,a,o) \leftarrow \hat{Q}(s,a,o) + \alpha[r^{\rightarrow}(s,a,o) \\ + \gamma \hat{Q}(s', greedy_{a'}(s'), o) - \hat{Q}(s,a,o)]\end{aligned} \tag{1}$$

where $r^{\rightarrow} \in \mathbb{R}^{\rightarrow}$ is the reward vector and $greedy_{a'}(s')$ is obtained through a Chebyshev scalarization function [29]. To make the algorithm Multi-Policy (MP), we conducted several runs with different weight settings, and for each weight combination we reserved the same amount of step budget.

PQL [31]: Differently from the previous algorithm, this is MP by design. Specifically, it integrates the Pareto dominance principles into a RL approach to find multiple policies in a single run. By splitting the expected immediate rewards related to an action selection and the set of future discounted reward vectors, this algorithm is able to profitably use the learned policies consistently all over the state space. Between the different mechanisms available, we chose the hypervolume metric to evaluate the sets of vectors and balance exploration and exploitation. Since this algorithm is not limited to the convex hull, it is suitable for use in episodic environments with both deterministic and stochastic transition functions. Differently from single-objective settings, selecting actions based only on locally non-dominated Q-vectors can lead to suboptimal solutions because of no guaranties about cumulative episodic Pareto optimality. Therefore, PQL defines a global greedy policy that consistently tracks a target return vector with the decomposition mechanism described previously. In each state, the agent selects the action that minimizes the difference between the target vector and the sum of the average immediate reward and expected discounted future rewards, updating the tracking target as the agent goes through different states. In the experiments described in the next section, we leveraged the Manhattan distance between the terms, as exploited in the original work [31].

GeoPQL: Despite the proven effectiveness of PQL, the version presented is limited to a tabular representation, where each state-action pair stores a specific Q_{set}. In fact, if the number of state-action pairs grows significantly (as in most real-world problems), the memory requirements also grow, making the use of the algorithm more complicated. Therefore, a new version was implemented for this work, based on a geometric approach to fit the elements in each set. Since the environments treated in the experiments consisted of a *3*- and *4*-objectives optimization, the vectors were fitted first on a plane and then on a hyperplane, respectively. The form of the PF and its geometric shape are learned and updated iteratively through local iterations. Alg. 2 (provided in the Appendix) shows an overview of the behavior of GeoPQL considering a *3*-objectives optimization problem (the considerations are the same also for other amounts of objectives, except for the geometric model and the number of parameters involved). Specifically, for each state-action pair, a set of non-dominated points $D_{s,a}$ and a parameter vector $\theta_{s,a}$ that approximate the front are stored. Each time a new $q_{new} \in \mathbb{R}^d$ for (s,a) is found and is not dominated by the current set, it is added to the related $D_{s,a}$ and the model is fitted using a least-squares approach in $D_{s,a}$ to obtain the updated $\theta_{s,a}$. Since it requires a smaller amount of support points $D_{s,a}$ and a parameter vector θ_{s_a} to define the geometric shape depending on the amount of objectives (e.g., curve for two objectives or plane for three) instead of a potentially big list of non-dominated Q-Vectors for each state-action pair, we consider this version of the algorithm more suitable to deal with real-world problems. This results only in local interactions and updates, without the need to involve the entire Q-Table. In this work, we adopt an initial implementation of GeoPQL, which already demonstrates performance comparable to existing algorithms. As part of future work, we plan to further enhance the implementation by conducting hyperparameter optimization and robustness analyses.

IV. *GaiaRL* Environment

GaiaRL is implemented in Python to ease the interaction with online libraries, such as OpenAI Gym, and can be considered as an extension of the *Hephaestusforge* framework[4] for MOO. In particular, we integrated *morl-baselines* library [32] to perform MOO with RL. GeoPQL, the novel algorithm proposed in this paper, has been implemented using the tools provided by *morl-baselines*, facilitating direct comparison with other MORL algorithms.

[4]https://github.com/jpedro1992/HephaestusForge

To simulate dynamic conditions, when a replica is deployed on a cluster, the latency variables for that cluster are increased by a random factor of up to 15%. Conversely, when a replica is terminated after its mean service duration (one time unit as default), the latency variables decrease by a random factor of up to 15%. This ensures that latency varies dynamically and uniquely in each episode, avoiding static patterns. This dynamic variation is crucial for the RL agent to learn that adding replicas to a cluster will result in increased latency [8]. The linear model assumes that each additional replica incrementally affects latency, a reasonable approximation for CC environments where network bandwidth or congestion increases with additional deployments. Similar relationships have been widely used in the literature to simplify environments and facilitate the learning process of RL algorithms [8], [33]. During the training and evaluation processes, the multi-cluster ecosystem is formed by eight clusters picked randomly at the beginning of each episode among the types listed in Table I. Once again, this is to avoid static patterns and enable RL algorithms to learn a larger number of potential trade-offs between the conflicting objectives.

Observation Space Table II shows the observation space considered for the MO orchestration problem. It includes two sets of metrics: *VNF* and *Cluster*. The first set *VNF* corresponds to the deployment requirements of the VNF, including the number of requested replicas that need to be deployed (R), the CPU and memory requests for each replica (ω_{cpu} and ω_{mem}), the latency threshold (in milliseconds) that the cluster or clusters hosting the required replicas should respect (Δ_r), and the expected inter-arrival time (T_r) of the request r measured in time units, representing the time interval between successive VNF requests. The CPU and memory requests for VNFs are based on the resource requirements reported for Free5GS and specified in its deployment files [34]. Since the RL agent is triggered upon each service request arrival, the time interval between consecutive decision steps may vary. To help the agent better learn the environment dynamics and accurately capture changes in resource consumption across clusters between successive observations, the elapsed time since the previous step, denoted as T_r, is incorporated into the observation space. The second set, *Cluster*, includes metrics that describe the current state of the infrastructure. These metrics include resource capacities (Π_{cpu} and Π_{mem}), currently allocated resources (Θ_{cpu} and Θ_{mem}), and the current latency of the cluster. The latency is computed based on multiple measurements, which vary according to the number of available clusters in the environment. It represents the mean latency from cluster c to all other clusters. These latency values are initialized randomly within the range of 1 to 1000 ms at the start of each episode, introducing diverse network conditions across episodes. This variability enables the RL agent to experience different cluster dynamics during training, thus enhancing its ability to generalize across varying scenarios. The allocation of CPU and memory resources within each cluster depends directly on the number of replicas deployed. As more replicas are instantiated, the allocated CPU and memory increase proportionally to the resource requirements specified per replica. In contrast, when replicas are removed, the allocated resources decrease accordingly to reflect the available capacity.

TABLE I
THE HARDWARE CONFIGURATION OF EACH CLUSTER, REPORTING THE CPU (NUMBER OF CORES), THE MEMORY, AND THE COST PER DEVICE.

Cores (vCPU)	Memory (GB)	Cost (Units)	Device
4.0	1.0	2.0	Raspberry Pi3
4.0	4.0	4.0	Raspberry Pi4
4.0	16.0	16.0	Shuttle
4.0	16.0	32.0	Intel NUC
8.0	48.0	64.0	VWall

Regarding power consumption, we extracted data from two previous studies [35], [36]. Specifically, we derived power consumption profiles for five different cluster types based on the results presented in these works, as shown in Fig. 2. This approach enabled us to capture a wide spectrum of systems, ranging from low-power edge clusters to more energy-demanding fog and cloud clusters. Since both studies primarily presented power consumption as a function of CPU utilization through visualizations, we digitized the data using a CPU utilization scale in steps of 10, employing the numpy interpolation function[5] to extract the values. The obtained points were then used to construct a piecewise-linear function of CPU utilization and power consumption. During the evaluation, the framework interpolates the power consumption based on the current CPU utilization of each cluster. Also, Table I summarizes the hardware configurations used in our study. Each entry reports the number of CPU cores, memory capacity, and associated cost per device type. The device types correspond to commonly used edge and experimental platforms: Raspberry Pi 3 and 4 (lightweight edge devices), Shuttle and Intel NUC (edge servers), and hardware nodes available in VWall [37], [38]. The cost values are normalized units in the range $[2.0, 64.0]$, reflecting relative differences in hardware and operational expenses rather than absolute prices. The initial allocation of resources starts within the range $[0.0, 0.2]$ to account for reserved resources dedicated to background services (e.g., monitoring). By incorporating this information into the observation space, the RL agent becomes aware of the infrastructure conditions, allowing it to make informed decisions when selecting deployment actions from the action space.

Action Space Upon receiving a deployment request, the RL agent must decide whether to allocate all replicas to a single cluster, distribute the replicas across multiple clusters, or reject the request entirely. The spread policy aims to consider all available clusters for deployment, ensuring that at least two clusters are used to deploy all replicas of a request. The size of the action space depends on the total number of clusters in the CC scenario. Specifically, for a system with C clusters, the action space comprises $C + 2$ actions: allocating all replicas to any one of the C clusters, deploying replicas across mul-

[5]https://numpy.org/devdocs/reference/generated/numpy.interp.html

TABLE II
THE STRUCTURE OF THE OBSERVATION SPACE.

Set	Metric	Description
VNF	R	The number of requested replicas.
	ω_{cpu}	The CPU request of the VNF (in millicpu).
	ω_{mem}	The memory request (in mebibyte).
	Δ_r	The latency threshold of the request (in ms).
	T_r	The inter-arrival time (in time units).
Cluster	Π_{cpu}	The cluster's cpu capacity.
	Π_{mem}	The cluster's memory capacity.
	Θ_{cpu}	The CPU allocated in the cluster.
	Θ_{mem}	The memory allocated in the cluster.
	δ_c	The average latency of cluster c.

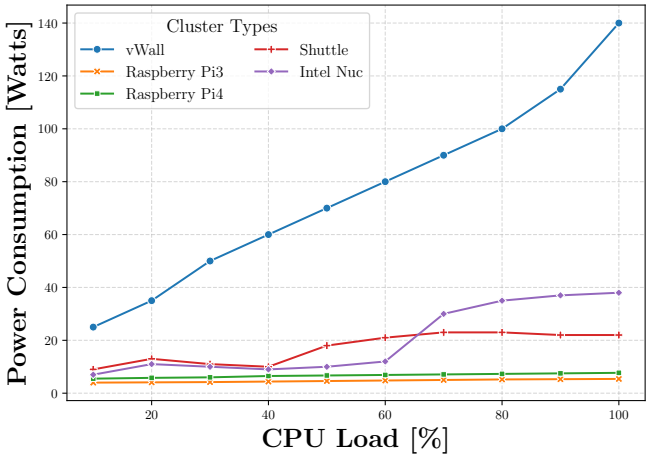


Fig. 2. The power consumption profile for the different cluster types. Edge devices show a conservative profile, while VWall is a representative example of a cloud location.

tiple clusters, or rejecting the request. Rejection is explicitly allowed to account for scenarios where resource availability is insufficient to satisfy the request. In such cases, the RL agent is not penalized, as rejection is a valid and necessary action under resource constraints. Regarding invalid actions (e.g., the chosen action will exceed available resources), these are often penalized in RL frameworks [39]. However, action masking, a technique applied in this study, has shown better performance and sample efficiency by guiding the agent toward feasible decisions by dynamically filtering out invalid actions based on the current state [8], [40].

For the spread action, *GaiaRL* adopts a First Fit Decreasing (FFD) strategy [8] as detailed in Alg. 1 in the Appendix. In particular, clusters are first sorted in descending order of available computing resources. For each cluster, a factor (f) is computed as the minimum ratio between the available CPU and memory and the replica's corresponding requirements. This factor determines how many replicas can be placed in the cluster, based on the most restrictive resource dimension. A threshold (Δ) is updated to reflect the lowest f across all clusters, capturing the capacity of the most constrained cluster. The algorithm then iterates over the sorted clusters, placing replicas while respecting both the CPU and memory limits. If replicas remain after the initial distribution loop ($DistLoop$), the loop is repeated until all replicas are deployed. The FFD heuristic is widely adopted for its simplicity and near-optimal performance, with a time complexity of $O(c \log c)$, where c is the number of clusters.

Reward Function A MO reward function has been designed based on four different objectives: **Cost**, **Latency**, **Inequality**, and **Power**. As mentioned in the previous section, each objective is a component of a reward vector that the agent uses as a guide for its decisions. Specifically, each vector component corresponds to a single reward computed by adding normalized values related to each objective obtained at every step performed by the agent, with the following overall definition:

$$r^{\rightarrow} = [r_{cost}, r_{latency}, r_{inequality}, r_{power}]$$

$$r_{cost} = 1 - \Gamma_d$$

$$r_{latency} = 1 - \lambda_d$$

$$r_{inequality} = 1 - G$$

$$r_{power} = 1 - \rho$$

If the agent accepts the deployment request, it receives positive reward components based on these strategies. Otherwise, the agent is penalized by a 5% factor for each vector component if it decides to reject the request despite having a sufficient amount of resources available to deploy all requested VNF replicas.

Cost encourages the agent to minimize deployment costs by prioritizing clusters with lower allocation expenses. Cloud clusters are significantly more costly than edge clusters, although they may also exhibit higher latency. As a result, the RL agent is incentivized to favor edge clusters, which offer higher rewards due to their lower costs. However, edge clusters often have limited computing resources and may be unable to accommodate all replicas of a given deployment request. The deployment cost of a request, denoted as Γ_d, is computed as the average cost of deploying all replicas within the request, defined as:

$$\Gamma_d = \frac{1}{R} \sum_{i=1}^{R} \tau_{c_i} \tag{2}$$

where:

Γ_d is the deployment cost of the VNF request.

R is the number of replicas within the request.

τ_{c_i} represents the allocation cost of the i-th replica.

Latency encourages the agent to minimize the expected latency of deployment requests. The overall latency is calculated from the latency of the clusters hosting the requested replicas, with each replica contributing to the expected latency of the VNF request. Accordingly, the expected latency of a deployment request, denoted as λ_d, is computed as the average latency across all replicas, taking into account the specific clusters where they are placed:

$$\lambda_d = \frac{1}{R} \sum_{i=1}^{R} \delta_{c_i} \tag{3}$$

where:

λ_d is the expected latency of the VNF request.

R is the number of replicas within the request.

δ_{c_i} represents the latency of the i-th replica.

Inequality aims that the RL agent distributes replicas evenly across available clusters. The reward is based on the *Gini Coefficient* (G) [41], which ranges from 0 to 1, where 0 represents perfect equality (all clusters host an equal number of replicas) and 1 denotes maximum inequality (all replicas are deployed in a single cluster). Thus, a lower *Gini Coefficient* reflects a more balanced distribution. The *Gini Coefficient* is a well-established metric for measuring inequality and is calculated as follows:

$$G = \frac{\sum_{i=1}^{c} \sum_{j=1}^{c} |L_i - L_j|}{2c^2 \bar{L}} \tag{4}$$

where:

G is the Gini coefficient.

c is the number of clusters.

L_i is the number of replicas deployed by cluster i.

$\bar{L}$ is the average number of replicas across all clusters.

Power Consumption encourages the RL agent to minimize the power usage of a given deployment request. Each cluster defines a power consumption profile that describes its power usage at various CPU loads. The environment tracks the power consumed by each cluster to guide the agent toward more sustainable placement decisions by minimizing the power consumption (ρ) of the VNF request. The overall goal is to prioritize computing clusters with lower power profiles, thereby increasing the sustainability of the deployment.

V. Evaluation

To evaluate the performance of MORL algorithms, we defined a scenario in which deployment requests correspond to the instantiation of VNF components of the free5GC platform, an open-source implementation of the 5G core. Each VNF specifies a minimum and maximum CPU requirement, reflecting its processing needs under varying load conditions. For example, lightweight services such as the Binding Support Function (BSF) or Network Slice Selection Function (NSSF) require only 0.1–0.3 CPU units, while more demanding components like the User Plane Function (UPF) can require up to 2.0 CPU units. This heterogeneity in resource demands enables a realistic and diverse evaluation of placement strategies. To quantify the performance of the different algorithms, we evaluate the **Acceptance Rate** (ratio of accepted to total requests, in $[0, 1]$), **Deployment Cost** (in units), expected **Latency** for accepted requests, the **Gini coefficient**, and the **Power Consumption** (in Watts) resulting from the selected deployment actions.

Then, to increase the relevance of the evaluation, we include in the comparison **two single-objective RL-based baselines** to verify if MORL methods can overcome these methods in the case of multiple and conflicting objectives. Specifically, we will consider **DeepSets (DS)-Proximal Policy Optimization (PPO)** and **DeepSets (DS)-Deep Q-Network (DQN)**. Both algorithms integrate DS as their Neural Network (NN) architecture and have shown strong performance in [8].

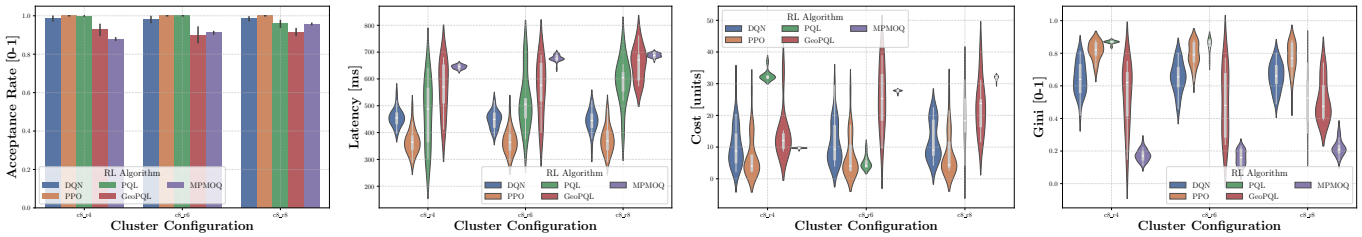
In the experiments, we adopted a balanced approach in which all objective weights were set equally for the single-objective RL baselines (i.e, 0.33 for *3*-objectives and 0.25 for *4*-objectives). During training and testing, the multi-cluster infrastructure consists of eight clusters. Each deployment request corresponds to a VNF that needs between four and eight replicas, which are then assigned to one or more clusters depending on the availability of resources and the agent's placement decision. PQL and GeoPQL have been trained over 10K steps (each episode has 250 steps), MPMOQL for 20K steps. MORL algorithms were tested for 50 episodes, and the results were presented with a confidence interval of 95%. Instead, DS-DQN and PPO-DS were trained for 200K steps as in [8]. Note that we empirically validated configurations with a higher number of steps for MORL algorithms, but observed no significant gains.

VI. Results

This section presents two distinct evaluations of single-objective RL versus MORL methodologies: *1) 3*-objective scenario and *2) 4*-objective scenario by incorporating power consumption into the reward formulation.

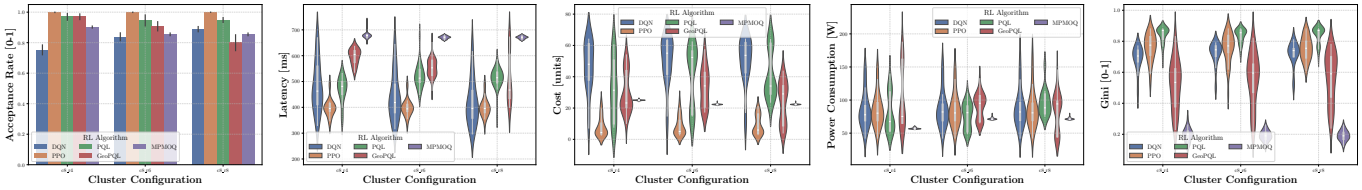
A. 3-*Objective Evaluation - Cost, Latency and Gini*

Fig. 3 presents the results for the *3*-objective scenario. The violin plots (3b - 3d) help visualize the distribution and the kernel density function. Fig. 3a demonstrates that all algorithms obtain an acceptance rate between 80% and 100% for all configurations, with both DS RL and PQL approaches overcoming the others. Focusing on MORL techniques, MPMOQ tends to prefer spreading strategies to distribute VNFs across computing resources at the cost of a higher rejection rate and a higher average latency (Fig. 3b). In fact, it is the algorithm with the lowest Gini values across all configurations (Fig. 3d). Both PQL and GeoPQL show an increasing trend in average latency and a decreasing one considering the inequality when the number of replicas per cluster increases. Furthermore, PQL reaches the lowest value of overall deployment costs when the maximum number of replicas is set to six, while both MPMOQ and GeoPQL perform worse. While GeoPQL can achieve low Gini values in certain solutions, it does so at the expense of either lower acceptance rates or higher costs and latencies. In contrast, both DS-PPO and DS-DQN appear largely unaffected by changes in configuration across the evaluated metrics, demonstrating strong generalization capabilities. However, their performance in minimizing the Gini index lags behind other MORL methods, as they tend to consolidate replicas into fewer clusters. For clarity, readers can refer to Table III in the appendix for the numerical results.



(a) Acceptance Rate. (b) Latency. (c) Deployment Cost. (d) Gini Index.

Fig. 3. Performance evaluation of multiple RL algorithms during the testing phase for the *3*-Objective scenario.



(a) Acceptance Rate. (b) Latency. (c) Deployment Cost. (d) Power Consumption. (e) Gini Index.

Fig. 4. Performance evaluation of multiple RL algorithms during the testing phase for the *4*-Objective scenario.

B. 4-Objective Evaluation - Power Consumption impact

Fig. 4 compares the performance of theRL algorithms for the *4-objective* evaluation. Introducing power consumption as an optimization objective has an immediate impact on the acceptance rate (Fig.4a), leading to a general decline in performance, except for DS-PPO and PQL, which remain largely unaffected. MPMOQ continues to prioritize VNF spreading strategies, achieving the lowest inequality (Fig.4e) and power consumption (Fig.4d) across all configurations. This demonstrates its effective utilization of edge clusters over VWall resources. The results of PQL and GeoPQL remain broadly consistent with the earlier *3*-objective evaluation. DS-PPO focuses on solutions with minimal costs (Fig. 4c) and latencies (Fig. 4b), but this comes at the expense of a poorer VNF distribution. In contrast, GeoPQL prioritizes fairness, similar to MPMOQ, but this leads to some of the worst outcomes in terms of both power consumption and average latency. This highlights a clear trade-off, with PQL striking a balance between fairness and efficiency across competing objectives. For clarity, readers can refer to Table IV in the appendix for the numerical results.

C. Trade-offs analysis

In summary, the trade-off analysis confirms that MORL methods provide greater flexibility in balancing competing objectives compared to single-objective RL approaches. While scalarized techniques generally converge towards fixed priorities, MORL algorithms, particularly PQL, seem capable of discovering diverse PF policies. This adaptability is an added value in dynamic CC environments, where multiple trade-offs must be simultaneously considered. Note that the open-source *GaiaRL* framework facilitates reproducibility and further research by providing an extensible platform for MORL, allowing researchers to easily implement and evaluate various RL algorithms across a wide range of scenarios.

VII. Conclusions

This paper explored the application of MORL techniques to the sustainable placement of VNFs within a CC environment. We demonstrated how MORL approaches enable more informed a posteriori decision-making by explicitly capturing the trade-offs among conflicting objectives, such as latency, cost, fairness, and energy consumption. To assess the effectiveness of MORL methods, we compared them against traditional single-objective RL algorithms that rely on scalarized reward functions through weighted sums. Our results revealed that as the number of objectives increased, MORL methods (especially PQL) can find more balanced trade-offs, thus achieving more sustainable solutions. However, DS-PPO still obtains interesting trade-offs, confirming the strong exploration capabilities of this algorithm. Considering the maturity of MORL libraries is still limited, future work will focus on studying multi-agent RL techniques and advancing MORL methodologies to better manage high-dimensional trade-offs and achieve more robust and sustainable solutions for CC environments.

Acknowledgment

This work has been partially supported by the Spoke 1 “FutureHPC & BigData” of the Italian Research Center on High-Performance Computing, Big Data and Quantum Computing (ICSC) funded by MUR Missione 4 - Next Generation EU (NGEU). José Santos is funded by the Research Foundation Flanders (FWO), grant number 1253226N.

REFERENCES

[1] 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.

[2] A. Laghrissi *et al.*, "A survey on the placement of virtual resources and virtual network functions," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 2, pp. 1409–1434, 2018.

[3] M. Liubogoshchev *et al.*, "Adaptive cloud-based extended reality: Modeling and optimization," *IEEE Access*, vol. 9, pp. 35 287–35 299, 2021.

[4] Y.-C. Wang *et al.*, "An overview on generative ai at scale with edge–cloud computing," *IEEE Open Journal of the Communications Society*, vol. 4, pp. 2952–2971, 2023.

[5] S. Che *et al.*, "Global insights on the impact of digital infrastructure on carbon emissions: a multidimensional analysis," *Journal of Environmental Management*, vol. 368, p. 122144, 2024.

[6] N. Jones *et al.*, "How to stop data centres from gobbling up the world's electricity," *nature*, vol. 561, no. 7722, pp. 163–166, 2018.

[7] P. N. Balasooriya *et al.*, "Green cloud computing and economics of the cloud: Moving towards sustainable future," *GSTF Journal on Computing (JoC)*, vol. 5, no. 1, p. 15, 2016.

[8] J. Santos *et al.*, "Hephaestusforge: Optimal microservice deployment across the compute continuum via reinforcement learning," *Future Generation Computer Systems*, vol. 166, p. 107680, 2025. Available at: https://www.sciencedirect.com/science/article/pii/S0167739X24006447

[9] N. Di Cicco *et al.*, "Multi-objective scheduling and resource allocation of kubernetes replicas across the compute continuum," in *2024 20th International Conference on Network and Service Management (CNSM)*, 2024, pp. 1–9.

[10] V. Noghin, "Linear scalarization in multi-criterion optimization," *Scientific and Technical Information Processing*, vol. 42, pp. 463–469, 2015.

[11] A. Al-Dulaimy *et al.*, "The computing continuum: From iot to the cloud," *Internet of Things*, vol. 27, p. 101272, 2024.

[12] K. Kaur *et al.*, "An energy-driven network function virtualization for multi-domain software defined networks," 2019, p. 121 – 126.

[13] J. Yao *et al.*, "Fog resource provisioning in reliability-aware iot networks," *IEEE Internet of Things Journal*, vol. 6, no. 5, p. 8262 – 8269, 2019.

[14] L. N. Vijouyeh *et al.*, "Efficient application deployment in fog-enabled infrastructures," in *2020 16th International Conference on Network and Service Management (CNSM)*. IEEE, 2020, pp. 1–9.

[15] M. Gao *et al.*, "The multi-objective cloud tasks scheduling based on hybrid particle swarm optimization," 2020, p. 1 – 5.

[16] P.-Y. Kong *et al.*, "Vnf orchestration and power-disjoint traffic flow routing for optimal communication robustness in smart grid with cyber-physical interdependence," *IEEE Transactions on Network and Service Management*, vol. 19, no. 4, pp. 4479–4490, 2022.

[17] R. Keshri *et al.*, "Energy-efficient communication-aware vm placement in cloud datacenter using hybrid aco–gwo," *Cluster Computing*, vol. 27, no. 9, p. 13047 – 13074, 2024.

[18] H. Huang *et al.*, "Scalable orchestration of service function chains in nfv-enabled networks: A federated reinforcement learning approach," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 8, pp. 2558–2571, 2021.

[19] M. Emu *et al.*, "Towards 6g networks: ensemble deep learning empowered vnf deployment for iot services," in *2021 IEEE 18th Annual Consumer Communications & Networking Conference (CCNC)*. IEEE, 2021, pp. 1–4.

[20] L. Rui *et al.*, "Sfc orchestration method for edge cloud and central cloud collaboration: Qos and energy consumption joint optimization combined with reputation assessment," *IEEE transactions on parallel and distributed systems*, vol. 34, no. 10, pp. 2735–2748, 2023.

[21] V. Sciancalepore *et al.*, "Rl-nsb: Reinforcement learning-based 5g network slice broker," *IEEE/ACM transactions on networking*, vol. 27, no. 4, pp. 1543–1557, 2019.

[22] J. Santos *et al.*, "Towards optimal load balancing in multi-zone kubernetes clusters via reinforcement learning," in *2024 33rd International Conference on Computer Communications and Networks (ICCCN)*. IEEE, 2024, pp. 1–9.

[23] F. Song *et al.*, "Evolutionary multi-objective reinforcement learning based trajectory control and task offloading in uav-assisted mobile edge computing," *IEEE Transactions on Mobile Computing*, vol. 22, no. 12, pp. 7387–7405, 2022.

[24] B. Kruekaew *et al.*, "Multi-objective task scheduling optimization for load balancing in cloud computing environment using hybrid artificial bee colony algorithm with reinforcement learning," *IEEE Access*, vol. 10, pp. 17 803–17 818, 2022.

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

[26] D. M. Roijers *et al.*, "A survey of multi-objective sequential decision-making," *J. Artif. Int. Res.*, vol. 48, no. 1, p. 67–113, Oct. 2013.

[27] C. F. Hayes *et al.*, "A practical guide to multi-objective reinforcement learning and planning," *Autonomous Agents and Multi-Agent Systems*, vol. 36, no. 1, p. 26, Apr 2022. Available at: https://doi.org/10.1007/s10458-022-09552-y

[28] B. Jang *et al.*, "Q-learning algorithms: A comprehensive classification and applications," *IEEE Access*, vol. 7, pp. 133 653–133 667, 2019.

[29] K. Van Moffaert *et al.*, "Scalarized multi-objective reinforcement learning: Novel design techniques," in *2013 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*, 2013, pp. 191–199.

[30] "Improved sarsa and dqn algorithms for reinforcement learning," *Theoretical Computer Science*, vol. 1027, p. 115025, 2025. Available at: https://www.sciencedirect.com/science/article/pii/S030439752400642X

[31] K. Van Moffaert *et al.*, "Multi-objective reinforcement learning using sets of pareto dominating policies," *J. Mach. Learn. Res.*, vol. 15, no. 1, p. 3483–3512, Jan. 2014.

[32] F. Felten *et al.*, "A toolkit for reliable benchmarking and research in multi-objective reinforcement learning," in *Proceedings of the 37th Conference on Neural Information Processing Systems (NeurIPS 2023)*, 2023.

[33] N. D. Cicco *et al.*, "Drl-forch: A scalable deep reinforcement learning-based fog computing orchestrator," in *2023 IEEE 9th International Conference on Network Softwarization (NetSoft)*, 2023, pp. 125–133.

[34] G. FIORENTINO, "Evaluation and comparison of free5gc in 5g networks: a comparative analysis," 2022.

[35] C. Centofanti *et al.*, "Impact of power consumption in containerized clouds: A comprehensive analysis of open-source power measurement tools," *Computer Networks*, vol. 245, p. 110371, 2024. Available at: https://www.sciencedirect.com/science/article/pii/S1389128624002032

[36] N. D. Cicco *et al.*, "Scalable and energy-efficient service orchestration in the edge-cloud continuum with multi-objective reinforcement learning," *IEEE Transactions on Network and Service Management*, pp. 1–1, 2025.

[37] M. Jain *et al.*, "Convolutional neural networks for real-time object detection with raspberry pi," 2021.

[38] Virtual Wall, "The virtual wall emulation environment." accessed on 2 September 2024. [Online]. Available: https://doc.ilabt.imec.be/ilabt/virtualwall/index.html.

[39] 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.

[40] 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.

[41] K. Blesch *et al.*, "Measuring inequality beyond the gini coefficient may clarify conflicting findings," *Nature human behaviour*, vol. 6, no. 11, pp. 1525–1536, 2022.

APPENDIX

Algorithm 1 First Fit Decreasing (FFD) [8]

Input: R, the number of requested replicas. C, the number of clusters. $\omega_{cpu,mem}$, the replica's requested cpu/memory. $\Omega_{cpu,mem}$, the cluster's amount of free cpu/memory.
Output: α, the distribution of replicas across all clusters

```
if R = 1 then
    penalty ← true                                  ▷ Penalize the agent
    return α = 0
end if
min ← 1, max ← R, Δ ← R ▷ Get min and max replicas
for each c ∈ C do                                   ▷ Calculate min factor
    f ← min(Ω_cpu[c]/ω_cpu[c], Ω_mem[c]/ω_mem[c])
    Δ ← min(f, Δ)
end for
if Δ ≥ R then
    Δ ← R − 1                                       ▷ To really distribute replicas
end if
S ← sorted(Ω_cpu)                                   ▷ Sort by decreasing order of CPU
for each c ∈ S do                                   ▷ DistLoop: distribute replicas
    if R = 0 then
        break
    else if R > 0 & Δ < R & (ω_cpu × Δ < Ω_cpu[c]) & (ω_mem × Δ < Ω_mem[c]) then
        α[c] = α[c] + Δ
        R = R − Δ
    else if R > 0 & (ω_cpu < Ω_cpu[c]) & (ω_mem < Ω_mem[c]) then
        α[c] = α[c] + min
        R = R − min
    end if
end for
if R = 0 then
    return α
else if R ≠ 0 then
    repeat
        DistLoop                                    ▷ Repeat the DistLoop
    until R = 0
end if
```

A. Algorithms additional details

Alg. 1 and Alg. 2 show the definitions of the FFD heuristic used for the spread action in the action space of *GaiaRL*, and the GeoPQL algorithm considered in this manuscript. As visible in Alg. 1, the sorting is based on the amount of free CPU. The main reason behind this decision lies in the fact that prioritizing free CPU yielded the highest average rewards for the RL agent in our previous research [8]. Since this design decision outperformed multiple heuristics for cluster sorting, including free memory and a balanced combination of both free CPU and free memory, it is used in *GaiaRL*.

Alg. 2 defines the behavior of GeoPQL, a MORL algorithm implemented for this work to realize a solution that could be more suitable for high-dimensional problems. As mentioned

Algorithm 2 GeoPQL

```
Initialize D[(s,a)] ← ∅, θ[(s,a)] ← [0,0,0], avg_reward[(s,a)] ← 0 for all (s,a)
Set hyperparameters γ, ϵ, ϵ_decay, ref_point, counters
while global_step < total_timesteps do
    Reset environment, get initial state s
    while episode not terminated or truncated do
        Select action a:
        if uniform(0,1) < ϵ then
            choose random a
        else
            compute scores via hypervolume, choose a with highest score
        end if
        Execute a, observe next state s′, reward vector r
        Update running mean: avg_reward[(s,a)] ← avg_reward[(s,a)] + (1/N)(r − avg_reward[(s,a)])
        Update non-dominated points in s′ over all actions
        for each action a′ do
            for each q_next ∈ D[(s′,a′)] do
                q_new ← avg_reward[(s,a)] + γ · q_next
                Insert q_new into D[(s,a)]
                Remove dominated points
                Fit geometric model, update θ[(s,a)]
            end for
        end for
        if D[(s′,∗)] empty then
            Insert immediate reward into D[(s,a)]
        end if
        Decay ϵ, s ← s′
    end while
    if global_step mod log_every = 0 then
        Evaluate all local Pareto solutions via policy tracking
    end if
end while
return local Pareto coverage set at initial state
```

previously, the PQL algorithm already showed promising results for different use cases. However, it adopts a tabular approach, which becomes increasingly complicated as the dimensionality of the problem grows. In contrast, GeoPQL requires very few parameters–basically only those required to define the approximated geometric shape of the obtained front–making it more suitable for problems of this nature.

B. Additional results

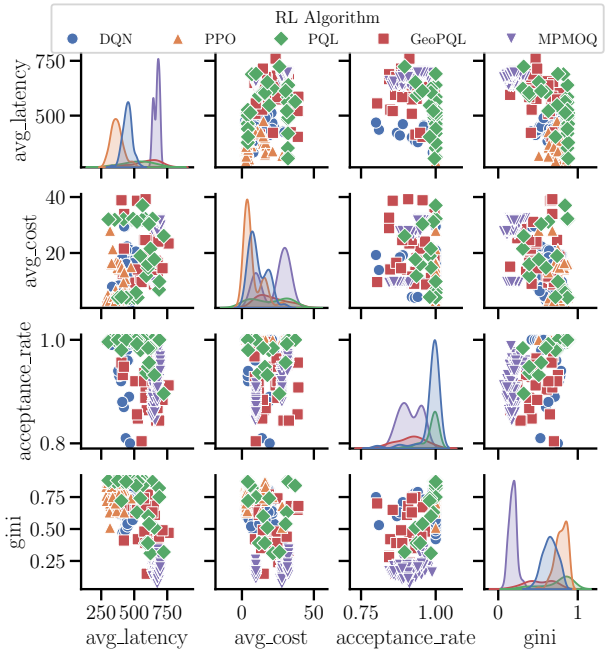
Tables III and IV summarize the average values and corresponding confidence intervals for each metric across the evaluated algorithms. In addition, Fig. 5a and Fig. 5b show the relationship between two performance metrics for the *3*-objectives and *4*-objectives scenarios, respectively. In the former, it is easy to note how DS-PPO maintains the lowest average costs by influencing Gini. DS-DQN demonstrated a more balanced profile among possible trade-offs, albeit with slightly worse performance on individual objectives. PQL

TABLE III
NUMERICAL RESULTS RELATED TO PERFORMANCE EVALUATION SHOWN IN FIG 3.

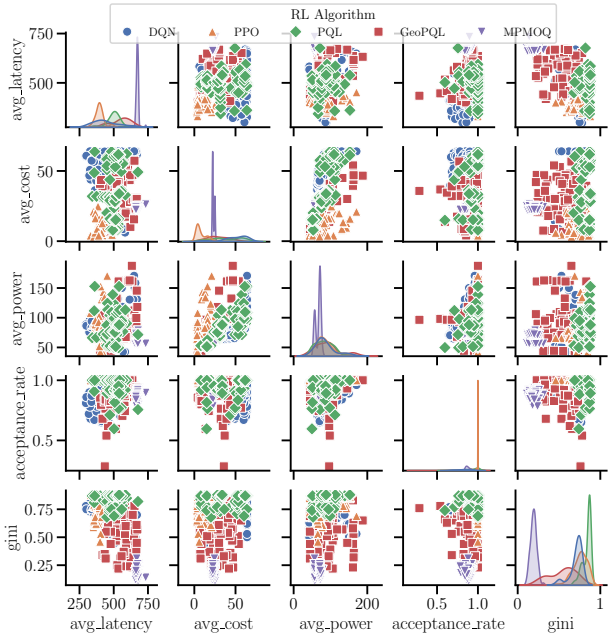
Alg.	Cluster Configuration	Acceptance Rate (in %)	Latency (in ms)	Deployment Cost (in units)	Gini Index
DQN	*8 cl. 4 replicas*	98.80 ± 1.37	455.42 ± 12.12	10.31 ± 2.40	0.65 ± 0.04
	8 cl. 6 replicas	98.37 ± 1.64	448.02 ± 11.91	11.22 ± 2.43	0.65 ± 0.03
	8 cl. 8 replicas	98.53 ± 1.13	442.93 ± 15.03	13.90 ± 5.28	0.54 ± 0.14
PPO	*8 cl. 4 replicas*	100.00 ± 0.0	365.23 ± 16.45	6.94 ± 2.38	0.81 ± 0.01
	8 cl. 6 replicas	100.00 ± 0.0	366.88 ± 16.71	7.12 ± 2.40	0.79 ± 0.03
	8 cl. 8 replicas	100.00 ± 0.0	374.73 ± 18.38	7.67 ± 2.46	0.77 ± 0.03
MPMOQ	*8 cl. 4 replicas*	87.88 ± 1.70	644.82 ± 4.10	9.67 ± 0.07	0.17 ± 0.01
	8 cl. 6 replicas	91.13 ± 1.70	677.24 ± 4.10	27.67 ± 0.13	0.16 ± 0.02
	8 cl. 8 replicas	95.72 ± 0.98	688.26 ± 2.84	31.91 ± 0.24	0.22 ± 0.02
PQL	*8 cl. 4 replicas*	99.92 ± 0.30	470.96 ± 85.60	32.55 ± 1.13	0.87 ± 0.01
	8 cl. 6 replicas	100.00 ± 0.0	516.62 ± 72.40	5.06 ± 1.39	0.85 ± 0.03
	8 cl. 8 replicas	95.88 ± 5.52	588.37 ± 60.95	19.26 ± 5.98	0.52 ± 0.11
GeoPQL	*8 cl. 4 replicas*	92.76 ± 8.54	564.79 ± 69.18	13.90 ± 5.28	0.54 ± 0.14
	8 cl. 6 replicas	89.95 ± 12.44	565.59 ± 78.75	25.82 ± 7.59	0.50 ± 0.16
	8 cl. 8 replicas	91.56 ± 5.46	656.93 ± 43.64	22.70 ± 5.90	0.51 ± 0.09

TABLE IV
NUMERICAL RESULTS RELATED TO PERFORMANCE EVALUATION SHOWN IN FIG 4.

Alg.	Cluster Config.	Acceptance Rate (in %)	Latency (in ms)	Depl. Cost (in units)	Power Consump. (in W)	Gini Index
DQN	*8 cl. 4 replicas*	75.08 ± 3.68	479.79 ± 38.66	46.46 ± 6.58	84.51 ± 10.97	0.69 ± 0.03
	8 cl. 6 replicas	83.80 ± 2.65	442.36 ± 34.17	50.19 ± 5.85	86.03 ± 10.23	0.72 ± 0.03
	8 cl. 8 replicas	88.93 ± 1.86	419.62 ± 33.50	50.29 ± 5.66	87.06 ± 10.77	0.72 ± 0.03
PPO	*8 cl. 4 replicas*	99.90 ± 2.07	397.32 ± 12.41	7.86 ± 2.32	84.96 ± 9.48	0.76 ± 0.03
	8 cl. 6 replicas	99.93 ± 0.14	397.71 ± 11.96	8.32 ± 2.28	85.33 ± 9.63	0.76 ± 0.04
	8 cl. 8 replicas	99.97 ± 0.07	398.65 ± 12.92	9.16 ± 2.31	86.66 ± 10.69	0.75 ± 0.03
MPMOQ	*8 cl. 4 replicas*	90.25 ± 1.66	677.79 ± 4.44	24.97 ± 0.11	57.24 ± 0.43	0.20 ± 0.02
	8 cl. 6 replicas	85.56 ± 2.16	670.73 ± 2.05	22.35 ± 0.15	71.86 ± 0.62	0.19 ± 0.01
	8 cl. 8 replicas	85.56 ± 2.16	670.73 ± 2.05	22.35 ± 0.15	71.86 ± 0.62	0.19 ± 0.01
PQL	*8 cl. 4 replicas*	97.24 ± 5.68	469.97 ± 18.41	32.15 ± 7.15	79.61 ± 10.57	0.84 ± 0.02
	8 cl. 6 replicas	94.61 ± 9.72	520.91 ± 16.93	41.76 ± 7.10	75.87 ± 7.17	0.84 ± 0.02
	8 cl. 8 replicas	94.84 ± 4.00	512.85 ± 14.54	41.60 ± 5.71	96.13 ± 7.45	0.84 ± 0.02
GeoPQL	*8 cl. 4 replicas*	97.14 ± 4.75	596.12 ± 11.00	30.16 ± 4.73	102.07 ± 16.93	0.52 ± 0.06
	8 cl. 6 replicas	90.84 ± 8.06	553.91 ± 13.29	33.52± 4.62	90.61 ± 6.95	0.55 ± 0.06
	8 cl. 8 replicas	80.60 ± 13.59	509.45 ± 35.98	23.72 ± 4.84	81.70 ± 10.62	0.56 ± 0.06



(a) *3*-Objective Pairplot.



(b) *4*-Objective Pairplot.

Fig. 5. The pairplot for both scenarios, illustrating the pairwise relationships between various metrics.

appears to be one of the most balanced approaches, combining moderate deployment costs and latency with relatively low Gini and power metrics without influencing the acceptance rate. In contrast, GeoPQL and MPMOQ show a stronger bias toward fairness and energy efficiency, which seems to lead to increased latency or higher rejection rates in certain cluster configurations. In the *4*-Objective scenario (Fig. 5b), the inclusion of power consumption introduces a sharper differentiation. DS-PPO and DS-DQN remain consistent in cost and latency optimization, but perform poorly for power efficiency and Gini. Meanwhile, MPMOQ and GeoPQL adapt their policy toward low power usage and higher VNF distribution across available clusters, confirming their inclination toward energy-aware orchestration. Finally, PQL still showed a high acceptance rate and a balanced performance across all considered dimensions (i.e, objectives).

C. Artifact details

This artifact is available for review at https://github.com/mattiazaccarini/morl_baselines_karmada/.