Towards Goal-Oriented Semantic Orchestration for Resource-Aware Robotic Function Offloading

Alejandro Calvillo-Fernandez, Milan Groshev, Carlos J. Bernardos

***Abstract*— Edge computing is increasingly seen as the perfect enabler for distributed robotic applications requiring low latency and high adaptability. However, current approaches offload computational tasks exclusively to edge nodes, overlooking scenarios where onboard processing can meet performance targets. Therefore, we propose a framework that unifies offloading decision and resource orchestration into a single algorithm that compares different allocation strategies under accuracy, latency, and energy goals. Results show that this approach admits more functions while maintaining strict performance targets, offering substantial gains in overall system utility.**

***Index Terms*— Robotic Services, Resource Orchestration, Offloading Strategies, Edge robotics.**

I. Introduction

Over the last decade, the increasing digitalization of industrial environments has brought edge computing as a natural evolution from cloud-based models by relocating computation closer to the physical processes [1]. This shift enables real-time data handling near to the source of generation, which can be used to dynamically adapt the operational logic to the current network conditions.

Edge computing has progressively enabled new deployment models in cyber-physical systems. Robotic systems have begun to embrace this transition by decoupling the intelligence of the robot from the physical device, allowing computationally intense functions (for example, those requiring the usage of Machine Learning (ML) models) to be offloaded to nearby edge nodes [2]. This evolution led to the concept of edge robotics, where robots operate not as isolated machines but as part of a distributed infrastructure that extends their capabilities beyond onboard hardware. In response, robotic systems can run more complex algorithms or process shared sensor data, with improved accuracy and lower latency. Within this paradigm, a robotic application (e.g., surveillance or material delivery) can be deployed as a modular software that integrates multiple distinct functions—such as visual recognition or mapping—into a cohesive task.

The distribution of robot functionalities across different edge nodes introduces new coordination challenges. While these functions are independently deployable, they must fulfill the strict network requirements (e.g., latency, reliability) of the application. As a consequence, there is a need for orchestration mechanisms to manage their placement, connectivity, and execution fulfilling the Quality of Service (QoS) requirements of the application [3].

Under the umbrella of orchestration, the concept of semantic orchestration [4] involves dynamically allocating computational and communication resources according to the semantic understanding of each application's QoS requirements. Recent advances in semantic orchestration support offloading functions to the edge server, while transmitting only function-relevant information. This strategy has been proven to optimize the resource consumption by enabling flexible resource allocation at the edge [5]–[8]. Current state-of-the-art (SoA) frameworks for semantic orchestration formulate slicing strategies that:

1) Account for the high non-linear relationship between slicing, data compression, end-to-end latency, and expected accuracy for the ML function.
2) Employ flexibility in resource assignments to balance the consumption of the different types of resources and avoid depletion of the most requested ones.

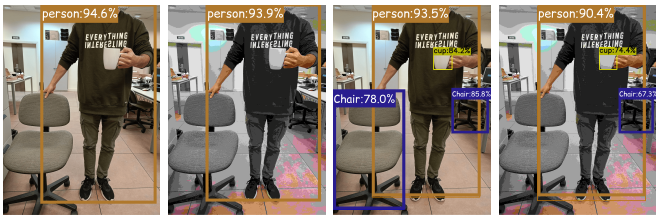

Fig. 1: Detection results using YOLOX_tiny and YOLOX_s models under two different input resolution compression factors: 0.5× and 0.01×. From left to right: YOLOX_tiny (0.5×), YOLOX_tiny (0.01×), YOLOX_s (0.5×), and YOLOX_s (0.01×).

Although recent advances in semantic orchestration have significantly improved robotic application performance by offloading robotic functionalities to the edge [9], existing approaches assume offloading as the default optimal strategy [9]–[11]. However, this assumption neglects situations where onboard robot processing could meet (or even exceed) the specific QoS targets of the robot functionality, creating a significant research gap. To illustrate this trade-off, we initially performed a preliminary evaluation for a computer vision function using two YOLOX [12] models. A light YOLOX version, named YOLOX_tiny, deployed on a resource-constrained device, to simulate the performance of a robot, and a standard YOLOX, named YOLOX_s, on an edge-server. We evaluated the impact of reducing the image quality in the inference of both models.

Our results depicted in Figure 1 shows that while YOLOX_s achieves robust detection even under heavier image quality degradation (compression factor of 0.01x),

YOLOX_tiny still reliably identifies key targets (e.g. persons), suggesting that full quality data transmission may not always be necessary, and local onboard processing can be sufficient depending on specific QoS.

Motivated by this initial validation, and the evident gap existing in literature, this paper proposes a framework that unifies the offloading decision and resource orchestration into a single algorithm. The key contributions of this work are: (1) Formulating and addressing the problem of jointly optimizing offloading decisions and resource orchestration as a unified optimization problem, accounting for latency, quality, and energy constraints. (2) Developing a light-weight greedy-based heuristic capable of evaluating different offloading strategies based on real-time conditions and application requirements. Our results show that this more holistic approach successfully admits a greater number of functions (25% more compared to the baseline) while maintaining strict quality and latency requirements, overcoming the limitations of simpler baselines and producing significant gains in overall system utility.

TABLE I: Notations and Preliminaries

Symbol	Description
π	Robotic application class
f_c	Robotic function (function f with configuration c)
z_{f_c}	Data compression factor
s_{f_c}	Vector of allocated resources for function f_c
$x^p_{f_c}$	Policy selection variable for robotic function
P, p_{f_c}	Set of policies and policy selected for robotic function
$l^p_{f_c}, q^p_{f_c}, e^p_{f_c}$	Latency, quality, energy scores for a robotic function

To facilitate the understanding of our mathematical formulation, we summarize in Table I the key notation used throughout the paper. The remainder of the paper is organized as follows. Section II presents the system model and problem formulation. Section III introduces our proposed methodology into a single algorithm. Section IV describes the experimental setup and validation. Finally, Section V concludes the paper, suggesting some directions for future investigations.

II. System Model and Problem Formulation

A. System Model

To represent the system's components, we propose a system model that captures the uniqueness of robot applications. Fig. 2 enumerates the main elements of the system under a Multi-Layer Graph (MLG). All components are detailed in the following.

- **Robotic Application Classes.** Each Robotic Application Class π is characterized by (i) a target latency L_π that states the maximum acceptable delay in between the robot and the edge; (ii) a target quality factor Q_π, (e.g., the minimum required accuracy for a computer vision task); and (iii) an application execution time T_π. Then the system will assign to each application a binary variable h_π that addresses whether these constraints are hard or soft constraints.

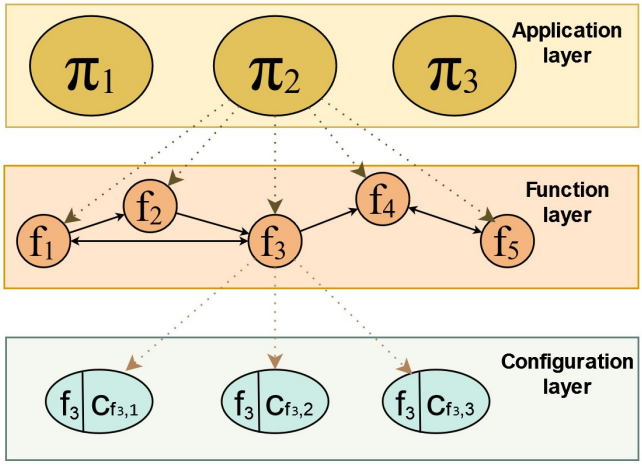

Fig. 2: System model under a Multi-Layer Graph structure which breaks down the relationships between components.

- **Functions and Configurations.** A function $f \in \mathcal{F}_\pi$ represents the fundamental building block of a robotic application, corresponding to a specific task. Therefore, robotic applications are modelled as chains of these interconnected functions. This chaining structure aligns with the concept of Service Function Chains (SFC), where functions are ordered and interconnected in a directed manner to form an end-to-end robotic service. Such dependencies can be represented by directed edges in a graph: an edge from function f to function f' indicates that f' requires the output from f (See Fig. 2). Each function can be implemented with a different configuration $c \in C_f$. For instance, an autonomous navigation function can be performed using a different set of sensors (LIDAR, depth camera...), or a computer vision task could be implemented with different ML models, each of them presenting a unique balance between accuracy and resource usage. A specific function with a given configuration defines a robotic function, denoted as $f_c = (f, c)$.
- **Data compression factor**. As different functions have different levels of sensitivity to input data degradation (See Fig.1), we define a compression factor $z_{f_c} \in (0, 1]$ to model the data stream quality. When $z_{f_c} = 1$, data quality is maximal, and will decrease proportionally as z_{f_c} decreases.
- **Radio and Computation resources.** The RAN edge server can provide our framework with radio (e.g., Radio Blocks (RBs)) and computing (e.g. CPU, GPU, and RAM) resources to run our functions. We define $K = \{1, ..., K\}$ to be the set of resources and $S = [S_1, ..S_K]$ the vector collecting the available amount of resource $k \in K$. So, we denote $s_{f_c} = [s_{f_{c,1}}, ..., s_{f_{c,K}}]$ as the amount of resource of type k selected for the robotic function f_c.
- **Offloading policies.** A strong capability that mobile

robots offer is the ability to perform some functions locally or partially locally. We define $\mathcal{P}$ to be the set of all possible policies that our system supports. Each robotic function must be implemented following one (and only one) policy, which is indicated as p_{f_c}. For example, we can have a robotic function handling a computer vision task that can be instantiated at the edge if it requires a stronger ML model to perform the inference, or in the robot, where a more lightweight ML model could be placed.

- **Battery consumption.** Depending on which policy is the most suitable, the energy trade-off will be different. Let $E_{tot} \in [0, 1]$ be the remaining battery level of the robot. When $E_{tot} = 1$ battery level is at its maximum and it gradually decreases until $E_{tot} = 0$. When a certain E_{tot} threshold limit b is reached, the system will send the robot to charge till the battery is full again. Note that the robot's internal mechanical components (e.g., wheels, end-effectors, servos) are powered by a separate integrated battery. Therefore, the energy consumption of the robotic functions depends on CPU time and the chosen offloading policy. We can write an energy function that models battery consumption for a robotic function as:

$$e^p_{f_c}(\mu_{p_{f_c}}, T_\pi) = \phi_{Robot} \cdot \mu^p_{f_c} \cdot T_\pi$$

  where $\mu_{p_{f_c}}$ is the average CPU wake-ups per second for the function f_c deployed with policy p at the robot, and ϕ_{Robot} is the battery consumed per CPU wake-up. Both values are hardware dependent. Therefore, they need to be derived by experimental validation for each policy, robotic function, and the robot hardware.
- **Application quality.** Let $q^p_{f_c}$ be defined as the quality score given from a quality function $q^p_{f_c}(z_{f_c})$. Notice, that the quality score will depend only on the compression of the input data z_{f_c}. Since a quality function is not straightforward to be derived (being influenced by the complex nonlinear responses of a ML model), incorporating an intricate mathematical model to capture it is impractical.
- **Application latency.** Similar to the quality factor, let $l^p_{f_c}$ be the latency score given from the output of the latency function $l^p_{f_c}(s_{f_c}, z_{f_c})$. In this case, a latency function is being defined by the compression of the input data, and the resources to be used.

It should be observed that the relationship between resource allocation, compression factor, and the latency and quality outcomes is not straightforward. Quality is strongly influenced by the complex, nonlinear response from a ML model, and latency is significantly affected by the specifics of the radio technology and channel conditions. Therefore, incorporating an intricate mathematical model to consider every involved factor would be unfeasible. Rather than developing a complex mathematical model, our strategy involves a data-driven approach in which functions for quality and latency are developed using a regression model, that will be given to the problem as an input.

B. Problem Formulation

Once defined the system model, we introduce the system constraints:

- **Function configuration selection.** Given $\mathcal{F}_\pi$ to be the set of functions f that composes an application. For each $f \in \mathcal{F}_\pi$, a configuration must be given from the set C_f for function f. This selection is denoted as f_c. Let x_{f_c} be defined as a binary decision variable that takes 1 if configuration c is selected for function f. We want to declare that (i) if one function from the set $\mathcal{F}_\pi$ cannot be allocated following any configuration, the application is rejected, and (ii) each function must follow only one configuration.

$$\sum_{c \in C_f} x_{f_c} \leq 1, \quad \forall f \in \mathcal{F}_\pi \tag{1}$$

- **Offloading policy selection.** The system must choose a suitable policy p from the set of supported policies P for each robotic function f_c. Let $x^p_{f_c}$ be a binary decision variable that equals 1 if policy p is selected for f_c. As the choice of a policy is crucial to decide where to allocate the robotic function (i) each robotic function must have one (and only one) policy assigned and (ii) if the system cannot determine any suitable policy to follow for a robotic function, the full application class will be rejected.

$$\sum_{f_c \in \mathcal{F}_\pi} x^p_{f_c} = x_{f_c}, \quad \forall p \in P \tag{2}$$

- **Satisfy the application requirements.** Each of the robotic functions must satisfy the target latency and quality objectives that define an application. Therefore, the system must choose for each $f_c \in \mathcal{F}_\pi$ an amount of resources s_{f_c} and a compression factor z_{f_c} in addition to an offloading policy p_{f_c} that satisfy the quality and latency objectives:

$$\min \left\{ q^p_{f_c}(z_{f_c}) \cdot x_{f_c} \right\} \geq Q_\pi \cdot h_\pi \tag{3}$$

$$\max \left\{ l^p_{f_c}(s_{f_c}, z_{f_c}) \right\} \leq L_\pi \cdot h_\pi \cdot x_{f_c} \tag{4}$$

- **Keep energy consumption limited.** As robots are battery dependent, if the expected energy usage of the robot is bigger than the current battery level, the application cannot be allocated in the system. That is, for all $f_c \in \mathcal{F}_\pi$:

$$\min \left\{ e^p_{f_c}(\mu_{p_{f_c}}, T_\pi) \cdot x_{f_c} \right\} \geq E_{tot} \tag{5}$$

- **Limit resource utilization to the total budget.** The amount of resources chosen to allocate a robotic function at the edge (S_{f_c}) must not exceed the total resource budget S.

$$\left\{ S_{f_c} \text{is feasible} \leftrightarrow S_{f_c,i} \leq S_i \quad \begin{matrix} \forall i = 1, \ldots, k \\ \forall f_c \in \mathcal{F}_\pi \end{matrix} \right\} \tag{6}$$

Then, considering all the previous constraints, the objective function of the problem is given by:

Robotic Function Offloading and Slicing Problem (R-FLOSP)

$$\max_{\mathbf{x,y,z}} \sum_{f_c \in \mathcal{F}_\pi} \sum_{p \in P} o_{f_c} x^p_{f_c} (E_{tot} - e^p_{f_c}) - \frac{1}{K} \sum_{i \leq K} \frac{S_{f_c,i}}{S_i} \quad (7)$$

s.t. (1) (2) (3) (4) (5) (6)

The objective function (7) maximizes the number of robotic functions $f_c \in \mathcal{F}_\pi$ hosted in the system according to their reward o_{f_c}, while minimizing the allocated resources at the edge, and minimize the energy usage of the robot. Theorem 1 below proves that the problem is NP-Hard.

Theorem 1. *R-FLOS is NP-hard*. We can establish the NP-hardness of the robotic application deployment, with all the rest of constraints, by a reduction from the mixed-integer programming (MIP) problem, which is known to be NP-Hard. To this end, let us review the goal of MIP: optimize a linear function subject to a set of linear restrictions, where some decision variables are restricted to be integers and others can take continuous values.

In this reduction, the binary decision variable $x^p_{f_c}$ is mapped to be an integer variable in MIP. Simultaneously, the continuous variables S_{f_c} and z_{f_c} correspond to the continuous variables in MIP. The objective function of our problem that combines the maximization of the number of implemented functions and minimization of resource usage at the edge and energy usage at the robot, aligns with a linear objective function in MIP, and our various constraints L_π, Q_π, E_{tot}, map to the linear constraints in MIP, thus ensuring a direct match between the two problems.

III. Solution

Given the NP-hardness of the R-FLOS problem, we propose a lightweight greedy-based heuristic to obtain a sub-optimal solution with reduced computational overhead. We denote this algorithm as Adaptive Resource-aware Decision Offloading (ARDO). ARDO at first sorts these robotic functions based on estimating their reward-to-cost ratio, defined for each function $f_c \in \mathcal{F}_\pi$ as:

$$priority_{f_c} = \frac{o_{f_c}}{min\_cost_{f_c}} \quad (8)$$

where min_cost is the minimum cost of assigning any feasible configuration, considering energy and resource requirements. However, determining a function priority requires identifying its corresponding resource allocation. Assuming the latency function $l^p_{f_c}(s_{f_c}, z^*_{f_c})$ increases monotonically with the compression factor, for each function f_c, ARDO computes the minimum compression factor $z^*_{f_c}$ that satisfies the application-level quality-constraint:

$$z^*_{f_c} = \min_z \quad \text{s.t.} \quad q^p_{f_c}(z_{f_c}) \geq Q_\pi \cdot h_\pi \quad (9)$$

Then, among the available options for resource allocation, it determines the resource vector $s^*_{f_c}$ that satisfies the latency constraints under the previously selected compression level, and fits within the current availability of resources:

$$s^*_{f_c} = \arg\min_s ||s|| \quad \text{s.t.} \quad l^p_{f_c}(s_{f_c}, z^*_{f_c}), s_i < S_i, \forall i \in K \quad (10)$$

Although multiple combinations of resources may satisfy both latency and quality constraints, the goal is not to minimize resource usage per function, but to maximize the total number of functions that can be admitted. Therefore, the selection of $s^*_{f_c}$ is guided by the resource availability. For instance, if radio resources are limited, the algorithm prefers to allocate fewer radio units and compensates by increasing compute resources to reduce processing delay, thereby enabling additional functions to be admitted without exhausting the bottleneck resource.

In practice, this search is limited to the number of policies the system can handle. Each policy p is then evaluated through a utility function:

$$utility^p = -l^p - e^p + q^p \quad (11)$$

where l^p, e^p, and q^p denote the estimated latency, energy consumption and quality for policy p. A lightweight stochastic comparison process based on gradient updates is run over a fixed number n of iterations, and the policy with the highest cumulative utility is selected:

$$p^* = arg \max_{p \in P} \sum_{i=1}^{n} utility^p_i \quad (12)$$

The initial step of ARDO is to (i) set the total available resource and the total energy of the robot (line 1), and (ii) include all the submitted functions of all applications in a candidate task set (line 2) Then, for each function in the set of candidate tasks, the priority is calculated according to (8) (line 3), and then sorted in descending order (line 4). Over the main loop (lines 5-18), the algorithm evaluates if there exists any feasible policy to follow for a given function. First, a random score is set for each policy (line 6). Then, for a given number of iterations, (i) a random policy is selected (line 8), (ii) metrics of latency, quality, and energy are gathered for that selected policy (line 9), (iii) the gradient is calculated (line 10), and (iv) the score is updated based on the previous score plus the gradient multiplied by a learning rate α (line 11). Once this process has finished, the algorithm takes the policy with the highest score (line 12) and checks if the solution is feasible. If so, the available resources and energy are updated, and the corresponding policy, resources, and compression factor are saved (line 16). Otherwise, the task is discarded (line 19). At the end of the algorithm, for each submitted application, if any function has been rejected, the application is rejected (line 22), and the reserved resources for the remaining functions of the application are then added to the pool (lines 24-26). In any other case, the application is admitted.

Algorithm 1: ARDO: Adaptive Resource-aware Decision Offloading

$S_{\text{avail}} \leftarrow S, \quad E_{\text{rem}} \leftarrow E_{\text{tot}}$;
$\mathcal{T} \leftarrow \bigcup_{\pi \in \Pi} \mathcal{F}_\pi$
Compute $priority_{fc}$ for each $f_c \in \mathcal{T}$:
Sort $\mathcal{T}$ in decreasing order of $priority_{f_c}$;
foreach $f_c \in \mathcal{T}$ **do**
  $score^p \sim U(0,1)$;
  for $i = 1$ **to** N **do**
    $p \sim P$;
    $(L, E, Q) \leftarrow \text{metrics}(p)$;
    $grad^p \leftarrow -L - E + Q$;
    $score^p \leftarrow score^p + \alpha \cdot grad^p$;
  $p^* \leftarrow \arg\max_p score^p$;
  if p^* *is feasible* **then**
    $S_{\text{avail}} \leftarrow S_{\text{avail}} - s^*_{f_c}$;
    $E_{\text{rem}} \leftarrow E_{\text{rem}} - e^p_{f_c}$;
    $p_{f_c} \leftarrow p^*$, $s_{f_c} \leftarrow s^*_{f_c}$, $z_{f_c} \leftarrow z^*_{f_c}$;
    $x^p_{f_c} \leftarrow 1$;
  else
    $x^p_{f_c} \leftarrow 0$
foreach $\pi \in \Pi$ **do**
  if *any* $f_c \in \mathcal{F}_\pi$ *is rejected* **then**
    $\pi \leftarrow$ Rejected;
    foreach $f_c \in \mathcal{F}_\pi$ *with* $x_{f_c} = 1$ **do**
      $S_{\text{avail}} \leftarrow S_{\text{avail}} + s^*_{f_c}$;
      $E_{\text{rem}} \leftarrow E_{\text{rem}} + e^p_{f_c}$;
      $x^p_{f_c} \leftarrow 0$;
  else
    $\pi \leftarrow$ Admitted;

IV. Performance Evaluation

We evaluate the performance of our proposed ARDO framework through extensive numerical simulations, comparing its performance against state-of-the-art offloading strategies.

Under a myriad of possible ways to process incoming robotic functions, we considered two primary offloading strategies: local processing on the robot, and offloading task entirely to the edge. Although more granular strategies (e.g., partial or cooperative offloading) exist, this binary scenario captures the core complexities of offloading decision-making to enable a clear comparison under realistic constraints

The performance of ARDO was compared against four baseline strategies: **Mixed-Greedy**, **Always Edge**, **Always Local**, and **SEM-O-RAN** [8]. The first three are greedy algorithms: Mixed-Greedy rank task according to their reward-to-cost ratio (Eq. 8), allowing task to be executed either at the edge or locally, depending on resource availability; Always Edge and Always Local represents the extremes of the offloading spectrum, offloading all tasks to edge servers or processing them entirely on the robot, respectively. In contrast, SEM-O-RAN dynamically computes optimal compression factors and allocates edge resources based on a primal gradient function, balancing resource consumption against availability.

A. Experimental Parameters and Setup:

The following parameters were defined to reflect a realistic operational scenario for a robotic application [13]:

- **Number of functions**: varied from 1 to 15 to evaluate scalability and performance under increasing workloads.
- **QoS requirements**: latency and quality targets were set to representative values, specifically latency $L_\pi = 250ms$ and quality $Q_\pi = 80\%$.
- **Function rewards** (o_{f_c}): randomly selected integers from the range $[1, 50]$ to simulate varying priority levels with different robotic task.
- **Energy costs and energy budget**: generated uniformly in $[0.05, 0.1]$ units for edge-offloaded tasks, and $[0.1, 0.3]$ units for locally processed tasks, with a Robot Energy budget (E_{tot}) randomly selected between $[0.7, 1]$. These intervals reflect the relative energy efficiency differences between edge and local computation.
- **Edge resources vector** (S_{edge}): defined as [RBs: 25, CPU cores: 16, RAM: 18 GB, GPUs: 2]. These parameters were chosen to reflect an edge-server capacity encountered in industrial deployments. The corresponding resource-cost vector was defined as $r_{edge} = 1/S_{edge}$, normalizing resource usage cost according to availability.

All experiments were implemented in a *MATLAB* simulation environment adapted from the public SEM-O-RAN implementation[1]. Quality and latency functions where modeled via regression-based approximations derived from empirical data, capturing the relationship between compression, quality and latency.

Each experimental scenario (varying function load from 1 to 15) was repeated across 100 independent trials. Results are averaged to provide statistically insights into each strategy's performance.

Figure 3 shows the number of successfully admitted functions accross varying workloads intensities. The ideal scenario is to accept all functions (diagonal grey dashed line at 100%). However, resource constraints of the system cause most strategies to reject a function as load grows. ARDO significantly outperformed the baseline strategies, admitting about 65% of functions at peak load (15 tasks), compared to only 40% for SEM-O-RAN, 33% for Mixed Greedy, 22% for Always Edge, and 17% for Always Local. We see that ARDO's accepted functions curve almost reaches the ideal line at low loads (accepting $\sim$100% up to 8 functions), and even at 10 functions still completes $\sim$9.4 on average.

Figure 4 further ilustrates the overall system utility score. The utility score captures how much net utility each strategy obtains. For every function successfully accepted, we add its

[1] https://github.com/corrado113/Semoran/

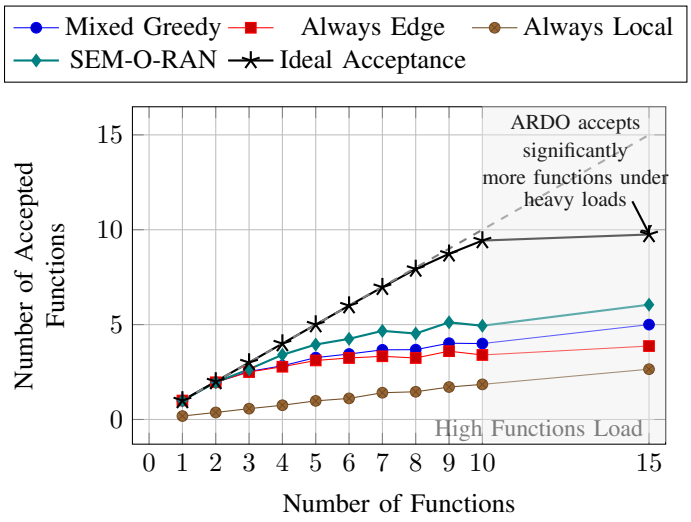

Fig. 3: Number of admitted functions for each approach over varying function loads.

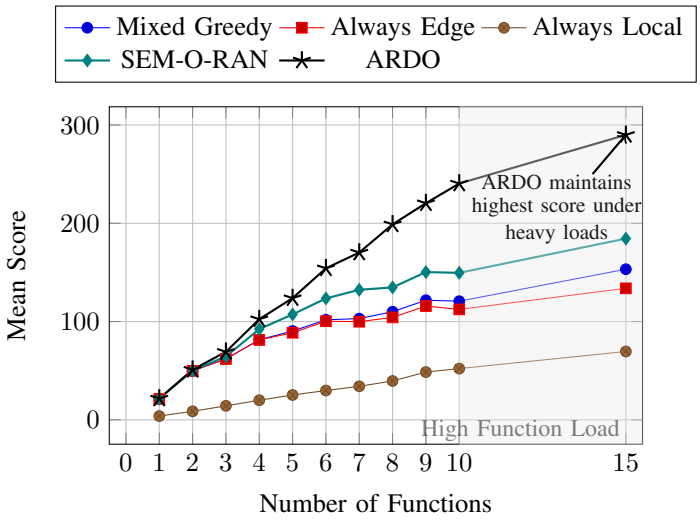

Fig. 4: Overall performance (Mean Score) as the number of functions increases.

reward o_{f_c} minus its energy cost e_{f_c}, and if the function is offloaded, we also subtract a resource-usage cost $\mathbf{r}_{\text{edge}} \cdot \mathbf{s}_{f_c}$:

$$\text{score} = \sum_{\text{accepted } t} \Big(o_{f_c} - \underbrace{\left(\mathbf{r}_{\text{edge}} \cdot \mathbf{s}_{f_c}\right)}_{\text{if offloaded}} - e_{f_C} \Big). \quad (13)$$

We see that as the function load grows, ARDO maintains the highest performance (score) at all scales. Initially, at 1–3 functions, all allocation strategies perform similarly, but as load increases, ARDO's advantage becomes evident. For example, at 10 functions, ARDO's score is about 240, roughly 2x higher than Always Edge and Mixed Greedy strategies ($\sim$120).

SEM-O-RAN improves over Mixed Greedy by using a gradient-based optimization, which selects edge allocations by maximizing a utility-to-resource ratio under latency and quality constraints, maximizing the overall utility. However, ARDO further improves upon this by considering both local and edge execution paths and using a lightweight gradient descent decision rule to compare their trade-offs for each function. This dual evaluation enables ARDO to admit more functions and achieve a better overall score.

V. Conclusions

This work presents a framework that allocates robotic services under different policies, outperforming traditional strategies in task acceptance and overall system utility, even under heavy load conditions. Future work should extend this framework to incorporate real-time adaptation to fluctuating environmental conditions while integrating edge-cloud continuum architectures to improve reliability in large-scale deployments.

Acknowledgments

This work was partially funded by the EC through the SNS JU PREDICT-6G project under grant agreement No.101095890; 6GINSPIRE PID2022-137329OB-C42, funded by MCIN/ AEI/10.13039/501100011033, and project UNICO 5G I+D 6G-EDGEDT.

References

[1] S. Dong, J. Tang, K. Abbas, R. Hou, J. Kamruzzaman, L. Rutkowski, and R. Buyya, "Task offloading strategies for mobile edge computing: A survey," *Computer Networks*, p. 110791, 2024.

[2] M. Afrin, J. Jin, A. Rahman, A. Gasparri, Y.-C. Tian, and A. Kulkarni, "Robotic edge resource allocation for agricultural cyber-physical system," *IEEE Transactions on Network Science and Engineering*, vol. 9, no. 6, pp. 3979–3990, 2022.

[3] L. Roda-Sanchez, C. Garrido-Hidalgo, F. Royo, J. L. Maté-Gómez, T. Olivares, and A. Fernández-Caballero, "Cloud–edge microservices architecture and service orchestration: An integral solution for a real-world deployment experience," *Internet of Things*, vol. 22, p. 100777, 2023.

[4] X. Luo, H.-H. Chen, and Q. Guo, "Semantic communications: Overview, open issues, and future research directions," *IEEE Wireless Communications*, vol. 29, no. 1, pp. 210–219, 2022.

[5] S. D'Oro, L. Bonati, M. Polese, and T. Melodia, "Orchestran: Orchestrating network intelligence in the open ran," *IEEE Transactions on Mobile Computing*, vol. 23, no. 7, pp. 7952–7968, 2023.

[6] G. Zheng, M. Wen, Z. Ning, and Z. Ding, "Computation-aware offloading for dnn inference tasks in semantic communication assisted mec systems," *IEEE Transactions on Wireless Communications*, 2025.

[7] F. Mungari, C. Puligheddu, A. Garcia-Saavedra, and C. F. Chiasserini, "Oreo: O-ran intelligence orchestration of xapp-based network services," in *IEEE INFOCOM 2024-IEEE Conference on Computer Communications*. IEEE, 2024, pp. 71–80.

[8] C. Puligheddu, J. Ashdown, C. F. Chiasserini, and F. Restuccia, "Sem-o-ran: Semantic and flexible o-ran slicing for nextg edge-assisted mobile systems," *arXiv preprint arXiv:2212.11853*, 2022.

[9] Y. Zhang, Y. Mao, H. Wang, Z. Yu, S. Guo, J. Zhang, L. Wang, and B. Guo, "Orchestrating joint offloading and scheduling for low-latency edge slam," *IEEE Transactions on Mobile Computing*, 2025.

[10] N. Bombieri, S. Germiniani, F. Lumpp, and G. Pravadelli, "Edge-cloud orchestration of assertion-based monitors for robotic applications," *ACM Transactions on Embedded Computing Systems*, 2025.

[11] B. Anuraj, D. Calvaresi, J.-M. Aerts, and J.-P. Calbimonte, "Dynamic swarm orchestration and semantics in iot edge devices: A systematic literature review," *Ieee Access*, 2024.

[12] Z. Ge, S. Liu, F. Wang, Z. Li, and J. Sun, "Yolox: Exceeding yolo series in 2021," *arXiv preprint arXiv:2107.08430*, 2021.

[13] D. Urbaniak, S. B. Damsgaard, W. Zhang, J. Rosell, R. Suárez, and M. Suppa, "Distributed control for collaborative robotic systems using 5g edge computing," *IEEE access*, 2024.