Dynamic Placement and Service Provisioning of AI/ML Inference Models in O-RAN Infrastructure

Abdul Basit Mir*, Samaresh Bera†, and Ankur Bansal*
*Department of Electrical Engineering, †Department of Computer Science and Engineering
Indian Institute of Technology Jammu, Jammu and Kashmir, India
{2024ree1025, samaresh.bera, ankur.bansal}@iitjammu.ac.in

***Abstract*—The open-radio access network (O-RAN) supported by softwarized and disaggregated 5G/6G RAN infrastructure enables placement of AI/ML inference models at distributed edge compute nodes near to the user. While this flexibility supports heterogeneous and time-varying services in 5G/6G, dynamic placement, re-use, and removal of the AI/ML models in near real-time are crucial for efficient service provisioning in O-RAN. In this paper, we study the dynamic placement of AI/ML inference models and service provisioning at O-RAN infrastructure while considering the service-specific requirements and constraints imposed by the O-RAN infrastructure. The problem is formulated as a per-slot constrained optimization model, shown to be NP-hard, capturing AI/ML models' functional coverage, CPU and memory capacities of O-RAN infrastructure, inference delay, and lock-based stability constraints. Consequently, we propose a low-complexity greedy dynamic placement and service provisioning algorithm that considers a trade-off between model placement and re-use to consider the associated overheads and time complexities. Simulation results under dynamic multi-service traffic show that the proposed greedy achieves performance within 4%, 18%, and 10% of the optimal in terms of average placements, cost, and removals, respectively, while reducing computation time by orders of magnitude.**

***Index Terms*—O-RAN, AI/ML model placement, mixed-integer linear programming, greedy algorithm.**

I. Introduction

The 5G/6G networks are undergoing a fundamental architectural transformation driven by the need for flexibility, programmability, and intelligence. Traditional hardware-based and monolithic Radio Access Network (RAN) architectures lack the flexibility and automation required for rapid reconfiguration and service innovation [1], [2]. To address these limitations, the open RAN (O-RAN) paradigm introduces a softwarized and disaggregated architecture based on open interfaces, functional decomposition, and cloud-edge deployment of RAN intelligence [2], [3]. This architectural openness enables the deployment of AI/ML models for a wide range of control and analytics tasks, including traffic prediction, scheduling, load balancing, and interference mitigation [4], [5]. Unlike legacy RAN designs, AI/ML-driven inference (i.e., the runtime execution of trained models to serve incoming control or analytics requests) in O-RAN can be executed on distributed compute nodes located at the CU/DU edge, allowing intelligence to be placed closer to the data plane [3].

The emerging applications supported by 5G/6G networks include enhanced mobile broadband (eMBB), ultra-reliable and low-latency communications (uRLLC), and massive machine-type communications (mMTC) [4], [5]. These applications exhibit fundamentally different characteristics in terms of latency tolerance, bandwidth demand, arrival dynamics, and computational complexity. Consequently, the demand for AI/ML inference and control decisions varies significantly over time [4]. As a result, static deployment of AI/ML inference models may lead to resource under-utilization, and violations of service-level requirements under time-varying traffic and heterogeneous service demands. Moreover, the deployment of AI/ML models and service provisioning need to be done in near real-time, considering the application-specific requirements.

This motivates the need for *dynamic placement of AI/ML inference models and service provisioning* across O-RAN compute nodes. In each time interval, the system must decide the following: a) which AI/ML models to deploy; b) which AI/ML models already deployed to reuse; and c) which AI/ML models to remove, while preserving the associated constraints. Importantly, such decisions must also preserve system stability. In particular, the trade-off between AI/ML model placement and re-use should be taken into account for system stability and associated overhead [6]. Consequently, designing such a dynamic orchestration mechanism is challenging for several reasons. First, O-RAN infrastructure enabled with compute nodes exhibit heterogeneous and limited CPU and memory capacities, restricting the number of AI/ML models that can be hosted simultaneously [3]. Second, individual service requests may require multiple AI/ML functionalities that must be jointly satisfied by one or more deployed models [2]. Third, control and inference latency depend on the coupling between computation time, system load and control-loop timing effects, which are particularly critical for uRLLC traffic [4].

Recent studies focus on the placement of AI/ML inference models in O-RAN infrastructure [2], [3], [6]–[8]. For example, D'Oro et al. [2] extend this vision with OrchestRAN, which provides mechanisms for selecting execution locations for xApps and rApps within the O-RAN ecosystem. Similarly, Almeida et al. [3] propose RIC-O, which investigates the optimal placement of disaggregated near-real-time RIC components under latency and resource constraints. Other optimization-based approaches, such as energy-aware orchestration of ML workloads by Ho et al. [7] or robust VNF reconfiguration under dynamic traffic by Amiri et al. [6],

consider temporal dynamics and reconfiguration costs. Other works explore security-aware DRL in O-RAN [8] and hybrid beamforming strategies for mmWave O-RAN systems [9].

Although AI/ML-based optimization in O-RAN has received considerable attention, existing works primarily focus on learning-based control policies [4], [5], energy-efficient orchestration [7], [10], or xApp/rApp execution frameworks [2], [3]. The problem of *dynamic AI/ML inference model placement and service provisioning*, under event-driven traffic, fine-grained delay constraints, and temporal stability requirements, has not been systematically studied. In particular, prior approaches do not consider request-driven, per-slot inference model placement with explicit placement continuity and lock-based stability constraints.

In this paper, we address these gaps by proposing a per-slot optimization framework for dynamic placement of AI/ML inference models and service provisioning within the near real-time RAN intelligent controller (near RT-RIC) of O-RAN. We consider a softwarized and disaggregated O-RAN architecture in which virtualized software entities are deployed on distributed edge infrastructure. Therefore, AI/ML inference models execute within the near RT-RIC platform (as xApps) and share a pool of distributed edge compute resources to serve heterogeneous, and time-varying service requests. The main contributions of this work are summarized as follows:

- **Dynamic AI/ML Model Placement Formulation:** We formulate the per-slot dynamic placement and assignment problem as a Mixed-Integer Quadratic Constrained Program (MIQCP).
- **MILP Reformulation and Optimal Solution:** By applying McCormick linearization [11], the original MIQCP is transformed into a Mixed-Integer Linear Program (MILP). This NP-hard is solved using a commercial solver to provide performance benchmarks.
- **Low-Complexity Greedy Algorithm:** To enable near-real-time operation, we develop a lightweight greedy dynamic placement algorithm that prioritizes model reuse, minimizes unnecessary re-configurations, and maintains feasibility with dynamic traffic rates.

The remainder of the paper is organized as follows. Section II presents the system model and problem formulation. Sections III describe the greedy algorithm. Section IV evaluates the proposed approaches through simulations, and Section V concludes the paper.

II. System Model

We consider a softwarized and disaggregated O-RAN architecture in which AI/ML inference models execute within the near-RT RIC platform (as xApps), sharing distributed edge compute resources, as illustrated in Fig. 1. Time is divided into discrete slots $t \in \{0, 1, \ldots, T_{\max}\text{-}1\}$. At each time slot, the system observes the currently active requests, the deployed AI/ML models, and available resources. Based on these, the system determines which AI/ML models must be placed, reused, or removed while satisfying functional, resource, and delay constraints.

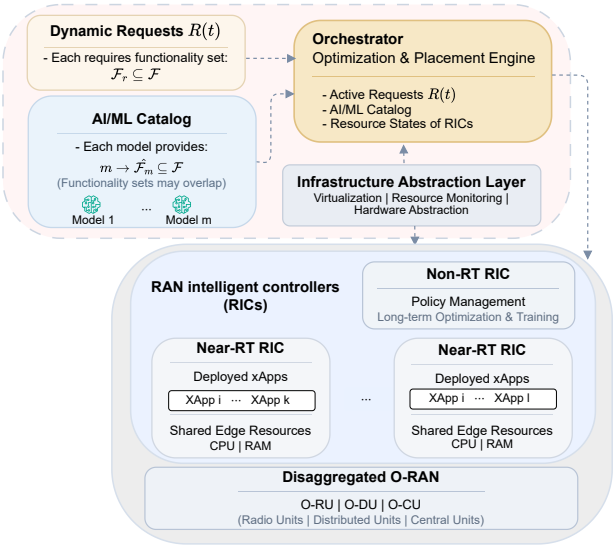


Fig. 1. System model of a disaggregated O-RAN architecture with AI/ML models running as xApps in the near-RT RIC at the edge cloud.

A. AI/ML Models with Functionalities

Let M be the set of pre-trained AI/ML inference models, whose inference quality is validated offline, available for deployment [12]. Let $\mathcal{F}$ denote the set of all possible AI/ML-driven functionalities. Each model $m \in M$ provides functionalities $\hat{\mathcal{F}}_m \subseteq \mathcal{F}$, corresponding to analytics or control tasks within the O-RAN ecosystem. Deploying a model m requires C_m^{cpu} CPU units and C_m^{ram} memory units.

B. Compute Node

We consider a set I of compute nodes corresponding to near-RT RIC. Each node $i \in I$ has CPU capacity C_i^{cpu} and memory capacity C_i^{ram}. If model m runs on node i, its effective service rate is as follows:

$$\mu_{m,i} = \frac{\gamma_{m,i}^{\text{cpu}} f_i^{\text{cpu}}}{N_m^{\text{flop}} D_m^{\text{in}}}, \tag{1}$$

where, $\gamma_{m,i}^{\text{cpu}}$ denotes the number of FLOPs per CPU cycle. f_i^{cpu} is the CPU frequency (in cycles/s). N_m^{flop} denotes the number of FLOPs per sample required by model m, and D_m^{in} denotes the number of input samples processed per inference using model m. Accordingly, the inference time of model m on node i, denoted by $t_{m,i}^{\text{inf}}$, is the reciprocal of the service rate, i.e., $t_{m,i}^{\text{inf}} = 1/\mu_{m,i}$.

C. Traffic and Request Model

At every time slot t, a set $R(t)$ of active user requests is present. The set of active requests, with heterogeneous service requirements, varies over time due to arrivals and departures. Each request $r \in R(t)$ is characterized by a required functionality set $\mathcal{F}_r \subseteq \mathcal{F}$, a maximum delay requirement Δ_r. A

request can be served only if all required functions in $\mathcal{F}_r$ are covered by functionalities of the AI/ML models serving the request.

D. Optimization Problem

We present the per-slot Mixed-Integer Quadratically Constrained Program (MIQCP) that is used to compute model placements, removals, and request model assignments at each time slot t. The formulation is designed to (i) preserve assignments for requests that persist across slots, (ii) enforce lock-based stability of newly deployed models, and (iii) minimize the cost composed of placement, removal and operational terms while satisfying capacity, functionality, and delay requirements.

1) Decision variables: For each time slot t, we define the following binary decision variables:

- $x_{m,i}(t) \in \{0,1\}$: 1 if model m is deployed at node i at time t, 0 otherwise.
- $p_{m,i}(t) \in \{0,1\}$: 1 if model m is *newly placed* at node i at time t.
- $z_{m,i}(t) \in \{0,1\}$: 1 if model m is *removed* from node i at time t.
- $y^{[r]}_{m,i}(t) \in \{0,1\}$: 1 if request $r \in R(t)$ is assigned to model m at node i at time t.

2) Objective: The per-slot objective minimizes a weighted sum of placement cost, removal benefit, and operational cost:

$$\min \underbrace{\sum_{m\in M}\sum_{i\in I} \alpha\, w_{m,i}\, p_{m,i}(t)}_{\text{placement cost}} - \underbrace{\sum_{m\in M}\sum_{i\in I} \beta\, w^{\text{rem}}_{m,i}\, z_{m,i}(t)}_{\text{removal benefit}} + \underbrace{\sum_{r\in R(t)}\sum_{m\in M}\sum_{i\in I} \gamma\, c^{\text{op}}\, y^{[r]}_{m,i}(t)}_{\text{operational cost}}. \tag{2}$$

The design of coefficients α, β, γ reflects the trade-offs between minimizing reconfiguration events, incentivizing removal of idle models, and reducing per-request operational overhead. $w_{m,i}$ is the placement cost for placing model m on node i. The symbol $w^{\text{rem}}_{m,i}$ is the removal benefit (or cost reduction) for removing model m from node i, and c^{op} is a constant per-request operational cost incurred whenever a request is served by a deployed model.

E. Constraints

The optimization is subject to the following constraints.

1) Persistency of assignments: Assignments of ongoing requests are preserved to avoid disruption of active inference pipelines:

$$y^{[r]}_{m,i}(t) = y^{[r]}_{m,i}(t-1), \forall r \in \{R(t) \cap R(t-1)\}, \forall m \in M, \forall i \in I. \tag{3}$$

2) Deployment evolution and lock: Deployment evolution is constrained as follows:

$$z_{m,i}(t) \le 1 - \sum_{r\in R(t)} y^{[r]}_{m,i}(t), \tag{4a}$$

$$z_{m,i}(t) \le \mathbf{1}_{\{t-\delta_{p_{m,i}} \ge \tau_{\text{lock}}\}}, \tag{4b}$$

$$z_{m,i}(t) \le 1 - p_{m,i}(t), \tag{4c}$$

where, $\delta_{p_{m,i}}$ denotes the most recent time slot at which model m was placed on node i (i.e., $p_{m,i}(t) = 1$), and τ_{lock} is the lock period (minimum lifetime after placement). Constraints (4a)–(4c) impose deployment stability in the near-RT RIC. Constraint (4a) prevents removal of a model that is currently serving at least one request, ensuring uninterrupted inference for ongoing services. Constraint (4b) enforces a uniform lock period τ_{lock} after placement, modeling the non-negligible cost of frequent deployment and teardown of inference models in near-RT operation. Constraint (4c) explicitly prevents placement and removal of a model within the same time slot, avoiding degenerate reconfiguration decisions. These assumptions reflect practical near-RT RIC operation, in which frequent model churn is limited by non-negligible control-plane and initialization overhead.

3) Resource capacity: For each node i, the sum of deployed model resource footprints cannot exceed the node capacities:

$$\sum_{m\in M} C^{\text{cpu}}_m\, x_{m,i}(t) \le C^{\text{cpu}}_i, \forall i \in I, \tag{5}$$

$$\sum_{m\in M} C^{\text{ram}}_m\, x_{m,i}(t) \le C^{\text{ram}}_i, \forall i \in I. \tag{6}$$

4) Functionality coverage: Each request r requires a set $\mathcal{F}_r$ of functionalities. For each function $f \in \mathcal{F}_r$, we require at least one assigned model that provides it:

$$\sum_{m\in M}\sum_{i\in I} a_{m,f}\, y^{[r]}_{m,i}(t) \;\ge\; 1, \forall r \in R(t), \forall f \in \mathcal{F}_r, \tag{7}$$

where, $a_{m,f} = 1$ if model m provides functionality f, and 0 otherwise.

5) Assignment feasibility: An assignment is feasible only if the model is deployed:

$$y^{[r]}_{m,i}(t) \;\le\; x_{m,i}(t), \forall r \in R(t), \forall m \in M, i \in I. \tag{8}$$

6) Delay constraint: The end-to-end inference delay experienced by each request $r \in R(t)$, including inference and queueing effects due to resource contention, must not exceed its delay bound Δ_r. We model this using a load-coupled end-to-end inference delay with implicit contention-induced queueing effects, following processor-sharing queueing abstractions that capture concurrent service sharing in virtualized computing platforms [13], yielding

$$\sum_{m\in M}\sum_{i\in I} \frac{L_{m,i}(t)}{\mu_{m,i}}\, y^{[r]}_{m,i}(t) \;\le\; \Delta_r, \forall r \in R(t), \tag{9}$$

where, $L_{m,i}(t) \triangleq \sum_{r\in R(t)} y^{[r]}_{m,i}(t)$ denotes the aggregate number of active requests assigned to model m on node i at time slot t.

The formulated optimization problem contains binary placement and assignment variables together with nonlinear delay expressions that involve products of decision variables, which makes the original formulation a Mixed-Integer Quadratic Constrained Program (MIQCP). Therefore, we linearize the problem as a Mixed-Integer Linear Program (MILP) using standard McCormick envelopes by introducing auxiliary variables and linear envelope constraints [11]. However, the combinatorial nature of the placement and assignment decisions, which generalize capacitated facility location and assignment problems, makes the problem NP-hard. To support operation in near real-time, we propose a lightweight greedy dynamic-placement and service provisioning approach in the subsequent section.

III. Proposed Greedy Dynamic Placement Approach

The proposed greedy approach consists of three phases: a) existing request allocation; b) new request allocation; and c) removal of outstanding AI/ML models. The steps involved in each of these phases are presented in Algorithm 1.

Phase 1: At each time slot t, the controller identifies the set of active requests $R(t)$ by removing any expired sessions and preserving all requests that continue from the previous slot. All assignments $y^{[r]}_{m,i}(t)$ associated with persistent requests are kept unchanged, while only newly arrived requests form a set that requires fresh allocation decisions, corresponding to Steps 1–3 in Algorithm 1.

Phase 2: For a new request r, it is served/rejected based on the priority score, $\phi[r] = |\mathcal{F}_r|$, and constraints, corresponding to Steps 4–19 in Algorithm 1. For each feasible pair (r, m, i) for service provisioning, a priority score is computed to prefer: (i) reuse of already deployed models; (ii) same-node assignment to reduce fragmentation; and (iii) new deployment only if necessary. The heuristic score for a feasible pair is computed as:

$$\begin{aligned}\text{score}(r,m,i) = &-\gamma\, c_{\text{op}} - \mathbb{I}_{\text{new}}\,\kappa\, w_{m,i} + \mathbb{I}_{\text{reuse}}\,\beta\, w^{\text{rem}}_{m,i} \\ &+ \rho_{r,i} + \zeta_1\,|\hat{\mathcal{F}}_m| - \zeta_2\, t^{\text{inf}}_{m,i},\end{aligned} \tag{10}$$

where $\mathbb{I}_{\text{new}}$, and $\mathbb{I}_{\text{reuse}}$ are indicator variables denoting whether the model requires a new deployment or is being reused, respectively. $\rho_{r,i}$ is the locality reward for serving request r from node i. The parameter κ penalizes new deployments and thus regulates the trade-off between responsiveness and deployment stability. The heuristic weights $(\gamma, \beta, \zeta_1, \zeta_2)$ are fixed design parameters that approximate the relative importance of operational cost, model reuse incentive, coverage gain, and model inference time, respectively.

Phase 3: After finishing allocation, the controller identifies the deployed AI/ML models that are still being assigned to active requests. Any pair (m, i) not serving an active request is considered for removal. If $t - \delta_{p_{m,i}} < \tau_{\text{lock}}$, the model remains deployed due to the lock constraint. Otherwise, it is removed and its resources are restored, corresponding to Steps 20–27 in Algorithm 1. The overall complexity of the greedy algorithm at any time slot can be expressed as: $\underbrace{O(|R|\,\log|R|)}_{\text{request ordering}} + \underbrace{O(|R|\,|\mathcal{F}_{\max}|\,|M|\,|I|}_{\text{model-node pair selection}} + \underbrace{O(|M|\,|I|)}_{\text{placement update}}$.

Algorithm 1: Proposed greedy algorithm

Input: Nodes I; models M with functionalities $\hat{\mathcal{F}}_m$; dynamic requests $R(t)$ with functionalities $\mathcal{F}_r$; lock period τ_{lock}; resource capacities.

Output: $x_{m,i}(t)$; $y^{[r]}_{m,i}(t)$, $\forall i \in \mathcal{I}$, $\forall r \in R(t)$, $\forall m \in M$.

Phase 1: Existing Request Allocation:

1 Get previously assigned requests: $\hat{R}(t) \leftarrow \{r \in R(t) : \exists (m,i)\, y^{[r]}_{m,i}(t) = 1\}$.
2 Determine new requests $\tilde{R}(t) = R(t) \setminus \hat{R}(t)$.
3 Preserve $y^{[r]}_{m,i}(t) = 1, \forall r \in \hat{R}(t)$.

Phase 2: New Request Allocation:

4 Compute available CPU/RAM on each node.
5 Sort $\tilde{R}(t)$ in decreasing order of $|\mathcal{F}_r|$.
6 **foreach** $r \in \tilde{R}(t)$ **do**
7 Remaining functionalities: $\tilde{F}_r \leftarrow \mathcal{F}_r$
8 **while** $\tilde{F}_r \neq \emptyset$ **do**
9 Identify candidate models: $\hat{M} = \{m : \hat{\mathcal{F}}_m \cap \tilde{F}_r \neq \emptyset\}$.
10 **if** $\hat{M} = \emptyset$ **then**
11 **Reject** request r and **break**
12 Select a feasible $(m^\star, i^\star)$ with constraints (5), (6), and (9), and the highest score in (10).
13 **if** *no feasible pair exists* **then**
14 **Reject** request r and **break**
15 **if** $x_{m^\star,i^\star}(t) = 0$ **then**
16 $p_{m^\star,i^\star}(t) \leftarrow 1$, $x_{m^\star,i^\star}(t) \leftarrow 1$, $\delta_{p_{m^\star,i^\star}} \leftarrow t$.
17 Update residual resources at i.
18 $y^{[r]}_{m^\star,i^\star}(t) \leftarrow 1$.
19 Update $\tilde{F}_r \leftarrow \tilde{F}_r \setminus \hat{\mathcal{F}}_{m^\star}$

Phase 3: Remove Outstanding Models:

20 $\text{KeepPlaced}(t) = \{(m,i) : \exists r \in \hat{R}(t)\, y^{[r]}_{m,i}(t) = 1\}$.
21 **foreach** (m,i) *with* $x_{m,i}(t) = 1$ **do**
22 **if** $t - \delta_{p_{m,i}} < \tau_{lock}$ **then**
23 add (m,i) to $\text{KeepPlaced}(t)$
24 **foreach** (m,i) *with* $x_{m,i}(t) = 1$ **do**
25 **if** $(m,i) \notin \text{KeepPlaced}(t)$ ***and*** $t - \delta_{p_{m,i}} \geq \tau_{lock}$ **then**
26 $x_{m,i}(t) \leftarrow 0$, $z_{m,i}(t) \leftarrow 1$.
27 **Restore** resources.
28 **return** $x_{m,i}(t)$, $y^{[r]}_{m,i}(t)$.

IV. Performance Evaluation

In this section, we evaluate the proposed greedy dynamic placement algorithm and compare it against the MILP bench-

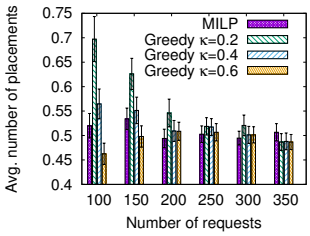


Fig. 2. Average number of placements

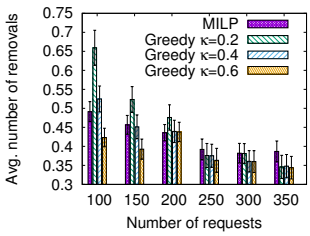


Fig. 3. Average number of removals

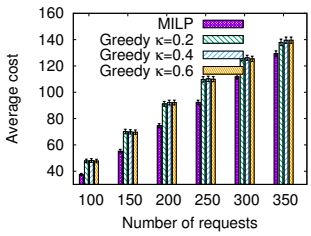


Fig. 4. Average cost

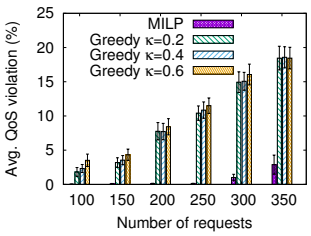
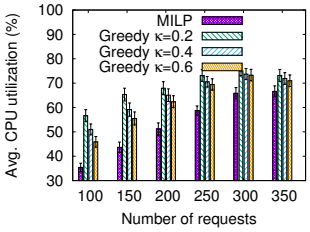


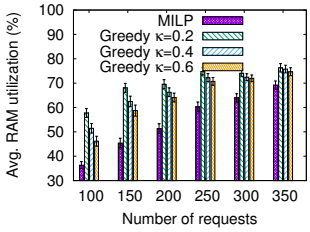
Fig. 5. Percentage of QoS violation

TABLE I
SIMULATION SETTINGS

Parameter	**Value**
Number of Nodes ($\|I\|$)	5
Number of AI/ML Models ($\|M\|$)	10
Total Functionalities ($\|\mathcal{F}\|$)	30
Maximum Number of Requests (R)	$100, 150, 200, 250, 300, 350$
Time Horizon ($T_{\max}$)	30 slots
System Arrival Rate (λ)	5–20 requests/slot (Poisson)
Delay Requirement (Δ_r)	5–10 ms
Model Input Samples (D_m^{in})	30–40 samples
CPU Capacity per Node (C_i^{cpu})	4–6
RAM Capacity per Node (C_i^{ram})	4–6 GB
CPU Frequency per Node (f_i^{cpu})	3.1–3.4 GHz
Model FLOPs per Inference (N_m^{flop})	4×10^4–6×10^4 FLOPs
CPU Efficiency Factor ($\gamma_{m,i}^{\text{cpu}}$)	1–2 FLOPs/cycle
Lock Period (τ_{lock})	2 slots
κ	$0.2, 0.4, 0.6$



(a) CPU utilization (%)



(b) RAM utilization (%)

Fig. 6. Average resource utilization

mark solution obtained using the IBM CPLEX solver [14]. Table I summarizes the simulation parameters. We consider a softwarized O-RAN compute infrastructure composed of heterogeneous compute nodes with limited CPU and memory resources and a pool of AI/ML inference models with diverse functional capabilities. The system scale reflects a localized near-RT RIC coordination domain. Traffic requests arrive dynamically according to a Poisson process and represent a mixture of eMBB, uRLLC, and mMTC services. Each request specifies a set of required functionalities and a maximum allowable inference delay. Placement and assignment decisions are made at discrete time slots over a finite simulation horizon. Each experiment is repeated over 50 independent runs, and results are reported with 95% confidence intervals [15]. All simulations were conducted on a workstation with an Intel Core i7 (14th Gen) processor and 32 GB RAM. Performance is evaluated in terms of the average number of placements and removals, objective value, CPU and RAM utilization, QoS violation rate, and computation time.

Fig. 2 illustrates the average number of model placements as the number of service requests increases. For light traffic load (100 requests), the MILP benchmark requires on average approximately 0.52 placements per time slot. In comparison, the greedy algorithm results in 0.70, 0.56, and 0.46 placements for $\kappa = 0.2$, 0.4, and 0.6, respectively. This corresponds to an overhead of about 35% for $\kappa = 0.2$, while $\kappa = 0.6$ achieves a reduction of roughly 12% relative to the MILP solution. As the request load increases beyond 250 requests, the number of placements stabilizes for all schemes, converging to approximately 0.49–0.51 placements. This behavior indicates that, under high load, both the greedy and MILP approaches effectively reuse already deployed models.

Fig. 3 shows the average number of model removals. With 100 requests, the MILP benchmark performs about 0.49 removals per slot, whereas the greedy algorithm incurs approximately 0.66, 0.52, and 0.42 removals for $\kappa = 0.2$, 0.4, and 0.6, respectively. Thus, for small κ, the greedy approach introduces up to 34% more removals due to its local decision-making. As the traffic load increases, removal rates decrease and converge across all schemes. This convergence highlights the effectiveness of the lock-period constraint in limiting excessive configuration churn, even under heavy load conditions.

The average objective value is depicted in Fig. 4. As expected, the cost increases monotonically with the number of requests due to higher resource utilization. Although the greedy scheme exhibits a noticeable gap at low traffic intensities, its performance improves with increasing load, resulting in a progressively smaller optimality gap. The lower optimality gap at higher loads indicates that the greedy heuristic increasingly captures the dominant cost trade-offs of the optimization problem.

Fig. 5 shows the percentage of requests violating their delay constraints. The violations in the MILP happen from delay infeasibility under resource saturation. Whereas in the greedy approach, violations may additionally stem from locally suboptimal or infeasible assignments for newly arriving requests. The greedy algorithm exhibits an increasing violation rate

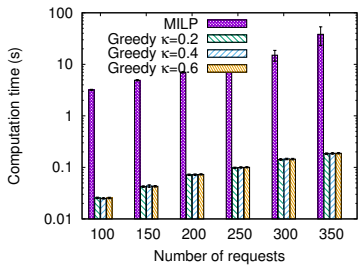


Fig. 7. Computation time

as the system becomes heavily loaded. For $\kappa = 0.6$, the violation rate grows from approximately 4% at 100 requests to about 18–20$\%$ at 350 requests. However, for moderate loads up to 250 requests, the violation rate remains below 12%, indicating that the greedy scheme maintains delay performance comparable to the MILP benchmark while operating within near-RT computational constraints.

Figs. 6(a) and 6(b) report the average CPU and RAM utilization, respectively. For light traffic load (100 requests), the MILP benchmark yields an average CPU utilization of approximately 35%, whereas the greedy scheme with $\kappa = 0.2$ reaches about 48%. As the load increases to 350 requests, CPU utilization rises to nearly 67% under MILP and about 72% under the greedy approach. A comparable trend is observed for RAM utilization. Overall, the greedy scheme consistently exhibits higher resource utilization than MILP, with the performance gap narrowing as the traffic load increases.

Fig. 7 compares the computation time of the greedy and MILP-based approaches. The MILP solver exhibits an exponential increase in runtime with increasing number of requests. In contrast, the greedy algorithm consistently completes within 20–200 milliseconds across all traffic loads. This corresponds to a reduction of up to two orders of magnitude in computation time, making the proposed greedy approach suitable for near-real-time deployment in O-RAN control environments.

V. Conclusion

This paper investigated the problem of dynamic placement of AI/ML inference models in O-RAN-based NextG networks, with the goal of serving heterogeneous and time-varying service requests under resource and delay constraints while limiting excessive model reconfigurations. The problem was formulated as a per-slot constrained optimization model incorporating functional coverage, CPU and memory capacities, inference delay, and lock-based stability requirements, and was shown to be NP-hard. To address this challenge, we adopted a twofold solution approach: a MILP formulation was used to obtain optimal benchmark solutions, and a lightweight greedy dynamic placement algorithm was proposed to enable near-real-time operation. Simulation results under dynamic multi-service traffic demonstrate that the proposed greedy algorithm approximates the MILP benchmark in terms of cost, placement efficiency, and resource utilization, while achieving orders-of-magnitude lower computation time and reduced model churn. These characteristics make the greedy approach well suited for execution within near-RT RIC control loops in practical O-RAN deployments. Future work will extend the proposed framework to multi-domain orchestration across O-RAN, edge, and cloud resources, and explore learning-based or predictive mechanisms for proactive model placement.

References

[1] L. Bonati, M. Polese, S. D'Oro, S. Basagni, and T. Melodia, "Neutran: An open ran neutral host architecture for zero-touch ran and spectrum sharing," *IEEE Trans. Mobile Comput.*, vol. 23, no. 5, pp. 5786–5798, 2023.

[2] S. D'Oro, L. Bonati, M. Polese, and T. Melodia, "Orchestran: Orchestrating network intelligence in the open ran," *IEEE Trans. Mobile Comput.*, no. 7, pp. 7952–7968, 2023.

[3] G. M. Almeida, G. Z. Bruno, A. Huff, M. Hiltunen, E. P. Duarte, C. B. Both, and K. V. Cardoso, "Ric-o: Efficient placement of a disaggregated and distributed ran intelligent controller with dynamic clustering of radio nodes," *IEEE J. Sel. Areas Commun.*, vol. 42, no. 2, pp. 446–459, 2023.

[4] M. Tsampazi, S. D'Oro, M. Polese, L. Bonati, G. Poitau, M. Healy, M. Alavirad, and T. Melodia, "Pandora: Automated design and comprehensive evaluation of deep reinforcement learning agents for open ran," *IEEE Trans. Mobile Comput.*, 2024.

[5] A. Lacava, M. Polese, R. Sivaraj, R. Soundrarajan, B. S. Bhati, T. Singh, T. Zugno, F. Cuomo, and T. Melodia, "Programmable and customized intelligence for traffic steering in 5g networks using open ran architectures," *IEEE Trans. Mobile Comput.*, vol. 23, no. 4, pp. 2882–2897, 2023.

[6] E. Amiri, N. Wang, M. Shojafar, M. Q. Hamdan, C. H. Foh, and R. Tafazolli, "Deep reinforcement learning for robust vnf reconfigurations in o-ran," *IEEE Trans. Netw. Service Manag.*, vol. 21, no. 1, pp. 1115–1128, 2023.

[7] T. M. Ho, K.-K. Nguyen, J. D. Vo, A. Larabi, and M. Cheriet, "Energy efficient orchestration for o-ran," in *Proc. IEEE GLOBECOM*, 2024, pp. 3316–3321.

[8] M. J. Shehab and A. Badawy, "Securing irs-swipt deployment in o-ran," *IEEE Trans. on Cogn. Commun. Netw.*, 2025.

[9] N. H. Tu, M. Kim, and K. Lee, "Semi-static hybrid beamforming for o-ran mmwave massive mimo systems," *IEEE Trans. Wireless Commun.*, 2025.

[10] G. Gemmi, M. Elkael, M. Polese, L. Maccari, H. Castel-Taleb, and T. Melodia, "Joint routing and energy optimization for integrated access and backhaul with open ran," in *Proc. IEEE GLOBECOM*, 2023, pp. 1962–1967.

[11] G. P. McCormick, "Computability of global solutions to factorable nonconvex programs: Part I—Convex underestimating problems," *Mathematical programming*, vol. 10, no. 1, pp. 147–175, 1976.

[12] O.-R. Alliance, "O-RAN AI/ML Workflow Description and Requirements 1.03," O-RAN Alliance, Tech. Rep. O-RAN.WG2.AIML-v01.03, 2021.

[13] L. Kleinrock, *Queueing Systems, Volume II: Computer Applications*. Wiley, 1976.

[14] IBM, "IBM ILOG CPLEX Optimization Studio," 2022. [Online]. Available: https://www.ibm.com/analytics/cplex-optimizer

[15] A. Hackshaw, "Statistical formulae for calculating some 95% confidence intervals," in *A Concise Guide to Clinical Trials*. John Wiley & Sons, 2009, pp. 205–207.