Faster Latency Constrained Service Placement in Edge Computing with Deep Reinforcement Learning

Orso Forghieri*, Yannick Carlinet†, Emmanuel Hyon‡, Erwan Le Pennec*, Nancy Perrot†
*CMAP, École Polytechnique, Institut Polytechnique de Paris, France
Email: {orso.forghieri, erwan.le-pennec}@polytechnique.edu
†Orange Gardens, France
Email: {yannick.carlinet, nancy.perrot}@orange.com
‡Sorbonne Université, CNRS, LIP6, F-75005 Paris, France §Université Paris Nanterre, Nanterre, France
Email: emmanuel.hyon@lip6.fr

***Abstract*—To enhance the user experience on mobile devices, Mobile Edge Computing (MEC) is a paradigm which integrates computing capabilities directly within access networks. However, designing efficient computation offloading policies in MEC systems remains a challenge. In particular, the decision on whether to process an incoming computation task locally on the mobile device or offload it to the cloud must intelligently adapt to dynamic environmental conditions. This article presents a novel approach that aims to address an edge computing optimization problem, issued from industrial cases, by modeling it as a combinatorial optimization problem combining multi-commodity flow and linear latencies constraints. We develop an equivalent linear formulation of the Service Placement Problem, allowing us to use traditional Integer Linear Programming (ILP) methods that turns out to be inefficient in practice. We therefore develop a use-case-based heuristic and a Reinforcement Learning (RL) methodology to model the network configuration under orchestration actions. The latest allows us to transfer learning across pre-training of the agent and shows proof of its efficiency on a dynamic real-world instance, aiming for practical deployment conditions. This comparison reveals that RL is a robust approach that can solve large realistic instances, reaching an optimality gap smaller than 25% on average below a second of runtime for dynamic service placement.**

***Index Terms*—edge computing, reinforcement learning, linear programming, service placement, modeling.**

I. Introduction

Following the increasing connectivity of portable devices and applications such as augmented reality, the need for computation and low-latency solutions has risen significantly. However, these services induce heavy network loading, which contradicts low-latency requirements. In this context, Mobile Edge Computing (MEC) is a method that allows the offloading of remote servers and links by placing services closer to the user [1]. Through computations and caching closer to the user, network loading and latency can be drastically reduced, resulting in a global improvement in the Quality of Experience (QoE) for users.

Several models of the resource allocation problem in the MEC context have been proposed. Given the necessity of fast network orchestration decisions, a compromise is required between the computation time of the placement and the expected QoE. In particular, most of the combinatorial optimization problems studied in this context have been proven to be NP-hard [2], and exact methods doesn't scale on realistic instances. Thus, approximation and heuristic algorithms have been proposed in the literature to provide feasible sub optimal solutions for resource allocation.

Given the diversity of graph instances, resource allocation methods must be scalable and adaptable. Reinforcement Learning (RL), a framework that learns behavior through environment interaction and feedback [3], is considered for this purpose. However, RL struggles with high dimensionality, especially in large decision spaces. Deep RL partially addresses this by using neural networks to extract relevant information from large systems [3] and transferring knowledge across network configurations [4]. However, deep RL's effectiveness is impacted by the size of the action space, and its performance guarantees are difficult to assess. Estimating its time efficiency and performance remains a challenging but necessary task for operational validation, which has been infrequently explored.

This paper deals with MEC use cases inspired by several network operators in [5]. It present a practical scalable approach to service placement decision making by realistic network modeling and Reinforcement Learning solving. We formulate the problem linearly for exact resolution. Due to its NP-hardness, we propose a heuristic based on real network structures and a RL framework for efficient solutions. Benchmarks show that exact methods don't scale, and the heuristic struggles with complexity, while Deep RL offers a robust, adaptable solution for online settings.

In Section III, we model Service Placement in Edge Computing as a combinatorial optimization problem that we then linearize. Section IV-B develops a practical heuristic

ISBN 978-3-903176-72-0

based on real-world instances, and Section V presents the RL framework. Finally, Section VI benchmarks the static and dynamic capabilities of the RL approach.

II. Related Work

Mobile Edge Computing has been widely investigated as the next immediate improvement of the 5G network through adapted resources allocation for Quality of Experience purpose [5]. In this context, authors proposed multiple modelings of the service placement problem to deal with tasks requiring low-latency [6]. Salaht *et al.* proposed a rich overview of these placement approaches [7]. We note that several models include a Multi-Commodity Flow (MCF) to take network orchestration decisions [8]. However, those problems design often induces NP-complete decision problems [9]. Deep RL has recently been applied to address various problems in combinatorial optimization (see [10] for a comprehensive review). Hence [11], [12] consider RL for solving Multi-Commodity Flow, mainly by transforming network configurations into an environment state, optimizing it step by step.

In communication and networking, several studies have leveraged Deep Q-Learning-based methods. For service placement problem [13] or to tackle resource allocation and offloading issues in cache-aided MEC networks [4], [14]–[18]. Various criteria such as induced delay or Quality of Experience are optimized, however, an in-depth comparison of runtime, optimality gap and performance of the proposed RL or heuristic approaches is rarely provided. Moreover, RL modeling is not always precisely formalized. As explainability is often seen as a challenge for Deep RL, it is nevertheless essential to detail how the method learns and operates, and what advantages can be extracted to improve existing proven approaches. Some studies have proposed problem-specific policy decompositions to enhance explainability [19].

Compared to existing works, we observed that real-world-based combinatorial optimization models for the MEC challenge are rare, and there is limited runtime comparison between the RL approach and exact ILP solving. Moreover, most of the studies do not estimate the adaptive capacities of RL in the online context. To balance this, our approach details a linear programming model of the service placement for latency optimization, as well as the RL problem design. We finally assess both in static and dynamic situations.

III. Problem Setup

This section defines the Service Placement Problem as a joint placement and routing task under a maximum latency constraint. Services are assigned to resource-limited nodes, with associated demands contributing to edge saturation and increasing network latency.

A. Context

Inspired by use cases like drone control and smart warehouses [5], we model a network with many mobile users offering limited caching and computing capacity. These users connect to 5G antennas that help offload tasks from cloud servers. Each user requires a single type of resource (e.g., cached data or image processing), provided by cloud or selected local servers. To ensure performance, we impose a maximum latency per demand. The problem is thus framed as a service placement task with linear latency, aiming to maximize user Quality of Experience (QoE), assumed to be proportional to the total data flow. Our problem addresses three key decisions. First, selecting the server for each service, modeled with variable $\mathbf{y}$ to enable offloading closer to users. Second, determining the service quality or data flow, represented by $\mathbf{q}$, which we aim to maximize as a proxy for Quality of Service. Third, routing decisions are captured by $\mathbf{x}$. Overall, the goal is to maximize flow through joint service placement and routing, under latency constraints.

B. Problem Design

The network is modeled as a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where nodes represent users, access points, and servers, and edges represent network connections. Following the network orchestration context [7], we introduce three decision variables, namely $\mathbf{x}_d^e \in \{0,1\}$ for routing, $\mathbf{y}_S^v \in \{0,1\}$ for placement and $\mathbf{q}_d \in \{q_{d,\min}, \dots, q_{d,\max}\}$ for the flow. In our model, each node provides resources $\mathrm{res}^v \in \mathbb{R}^+$ such as storage, CPU or GPU resoures which are available for any service. The edges support a flow/bandwidth between 0 and capa^e, representing the bandwidth usage. For each edge, the induced latency is modeled to depend linearly on the flow that the edge carries [20]:

$$\mathrm{lat}^e = \alpha^e \cdot \mathbf{q}_d + \beta^e \,.$$

The notations and typical numerical values used in the graph and instances design are summarized in Table I following [21].

TABLE I
Main graph parameters and variables

Symbol	Parameter	Typical Value
$\mathrm{capa}^e \in \mathbb{N}^+$	Maximum flow	1-100 Mbps
$\mathrm{res}^e \in \mathbb{N}$	Resources	10 TB-10 PB
$\beta^e \in \mathbb{R}^+$	Base latency	1 ms
$\alpha^e \in \mathbb{R}^+$	Induced latency	1 ms/Mbps
$\mathrm{lat}^d \in \mathbb{R}^+$	Required letency	10 ms

The main decision is where to place a service within the graph, allowing data to pass through the user, an Edge Server, or the cloud. In 5G, services typically run in the cloud unless users have significant onboard computing power. The second decision concerns the discrete Quality of Service $\mathbf{q}_d$, selected from a range $q_{d,\min}, \dots, q_{d,\max}$ with $q_{d,\min} \geq 1$. Each demand must respect a maximum latency $\mathrm{lat}_{d,\max}$, computed as the sum of latencies along its path. An instance includes multiple services, each $S \in \mathcal{S}$ defined by a set of demands with fixed origins d_o (e.g., users or the cloud). Each service must be placed on a node offering $r_S \in \mathbb{R}^+$ resources. Demands are routed from their origins to the placement node. The goal is to

maximize total Quality of Service under various constraints, defining the combinatorial Service Placement Problem (SPP).

$$
\begin{aligned}
&\max \sum_{d\in\mathcal{D}} \mathbf{q}_d && \text{(SPP)}\\
&\text{s. t.} \sum_{u\in\delta(v)} \left(\mathbf{x}_d^{(v,u)} - \mathbf{x}_d^{(u,v)}\right) = \mathbf{1}_{v=d_o} - \mathbf{y}_S^v && \forall v, \forall d\\
&\sum_{S\in\mathcal{S}} \mathbf{y}_S^v r_S \leq \mathrm{res}^v && \forall v \in \mathcal{V}\\
&\sum_{d\in\mathcal{D}} \mathbf{q}_d \mathbf{x}_d^e \leq \mathrm{capa}^e && \forall e \in \mathcal{E}\\
&\sum_{e\in\mathcal{E}} \mathbf{x}_d^e \left(\alpha^e \sum_{d'\in\mathcal{D}} \mathbf{q}_{d'} \mathbf{x}_{d'}^e + \beta^e\right) \leq \mathrm{lat}^d && \forall d \in \mathcal{D}\\
&\sum_{v\in\mathcal{V}} \mathbf{y}_S^v \geq 1 && \forall S \in \mathcal{S}\\
&\mathbf{q}_d \in \{q_{d,\min}, \ldots, q_{d,\max}\} && \forall d \in \mathcal{D}\\
&\mathbf{x}_d^e, \mathbf{y}_S^v \in \{0,1\} && \forall e, \forall d
\end{aligned}
$$

In (SPP), we maximize the QoS *i.e.* $\sum_{d\in\mathcal{D}} \mathbf{q}_d$. Any service $S \in \mathcal{S}$ is placed following the constraint $\sum_{v\in\mathcal{V}} \mathbf{y}_S^v \geq 1$ on the node v such that $\mathbf{y}_S^v = 1$. For each node v, its available resources res^v should not be exceeded by usage given the inequality $\sum_{S\in\mathcal{S}} \mathbf{y}_S^v r_S \leq \mathrm{res}^v$. The routing and flow conservation is then ensured by

$$\sum_{f\in\delta(v)} \left(\mathbf{x}_d^{(v,f)} - \mathbf{x}_d^{(f,v)}\right) = \mathbf{1}_{v=d_o} - \mathbf{y}_S^v \,.$$

We then constrain the edge capacity using the quadratic inequality $\sum_{d\in\mathcal{D}} \mathbf{q}_d \mathbf{x}_d^e \leq \mathrm{capa}^e$. Given the linear latency model, we ensure each demand latency to be at most lat^d by enforcing

$$\sum_{e\in\mathcal{E}} \mathbf{x}_d^e \left(\alpha^e \sum_{d'\in\mathcal{D}} \mathbf{q}_{d'} \mathbf{x}_{d'}^e + \beta^e\right) \leq \mathrm{lat}^d \,,$$

computing the total flow crossing each edge being visited by the current demand flow. Following the use cases, deploying any service on the cloud while ensuring the minimum Quality of Service (QoS) should constitute a feasible solution to the initial problem.

IV. Linearization and Heuristic

A. Linearization

The non-linearity of Service Placement Problem can be tackled introducing new variables and constraints that allow to eliminate the product of variables $\mathbf{q}_d$ and $\mathbf{x}_d^e$ introducing the variables $\mathbf{Q}_d^e$ that models the flow on each edge and $\mathbf{L}_d^e$ that models the latency. To ensure that $\mathbf{Q}_d^e = \mathbf{q}_d \cdot \mathbf{x}_d^e$, we add the following constraints to the problem:

$$
\begin{aligned}
&\mathbf{Q}_d^e \leq M \cdot \mathbf{x}_d^e && \forall d, \forall e\\
&\mathbf{Q}_d^e \geq \mathbf{q}_d - M \cdot (1 - \mathbf{x}_d^e) && \forall d, \forall e\\
&\mathbf{Q}_d^e \leq \mathbf{q}_d && \forall d, \forall e
\end{aligned}
$$

The second linearization is done to reduce the product $\mathbf{x}_e^d \left(\alpha^e \sum_{d'\in\mathcal{D}} \mathbf{Q}_{d'}^e + \beta^e\right)$ into a latency variable $\mathbf{L}_d^e$ setting $N := \alpha^e \cdot D \cdot \max_{d\in\mathcal{D}} q_{d,\max} + \beta^e$ by adding:

$$
\begin{aligned}
&\mathbf{L}_d^e \leq N \cdot \mathbf{x}_d^e && \forall d, \forall e\\
&\mathbf{L}_d^e \geq \alpha^e \sum_{d'\in\mathcal{D}} \mathbf{Q}_{d'}^e + \beta^e - N \cdot (1 - \mathbf{x}_d^e) && \forall d, \forall e\\
&\mathbf{L}_d^e \leq \alpha^e \sum_{d'\in\mathcal{D}} \mathbf{Q}_{d'}^e + \beta^e && \forall d, \forall e
\end{aligned}
$$

We therefore obtain a linearized problem, replacing the products by the variables $\mathbf{Q}_d^e$ and $\mathbf{L}_d^e$, and adding the previous constraints. This new problem being linear, it can be solved by a classical ILP solver like Gurobi or CPLEX. However, the number of variables and constraints is large, making the resolution of medium instances already difficult given its complexity. In the following, we propose practical solving methods to approach the optimal value of this problem.

B. Heuristic Approach

To solve the Service Placement Problem, we propose a heuristic method that relies on the basic cloud placement and solves the resulting routing problem greedily. This approach primarily relies on the special structure of the network. Assuming that the largest servers are positioned at the top of the hierarchy, placement is considered as the primary variable to determine. Once the placement is completed, the problem can be formulated as a Multi-Commodity Flow (MCF) problem with additional linear latency constraints. The MCF problem is typically addressed using a greedy approach [22].

A solution is the tuple $(\mathbf{y}, \mathbf{q}, \mathbf{x})$ in which each term is a vector composed of the decision variables described in section III-B and the heuristic then develops as follows:

(i) Place all services on the largest available server using the shortest path: $\mathbf{y}_S^{\text{cloud}} = 1$ and $\forall d \in d,\ \mu_d = 1$;
(ii) For each demand with the greatest margin m_d to maximum latency and arc capacity, increase the associated flow q_d until the path is saturated;
(iii) Displace one random service to a smaller server and reroute it using greedily increased flows;
(iv) Return the best visited solution.

A direct improvement of this method is to refine the best solution obtained from this heuristic. Since this placement is likely to be promising, we set it in the Service Placement Problem and solve the remaining linearized subproblem of routing under linear latency constraint.

V. Reinforcement Learning method

RL offers a flexible framework for modeling problems as step-by-step improvement processes. By exploring solutions and adapting to feedback, an agent learns to converge to effective strategies. In this section, we present the RL model used for the Service Placement Problem, detailing the environment design. An RL problem is defined as $(\mathcal{S}, \mathcal{A}, T, R)$, where $s \in \mathcal{S}$ is the state, $a \in \mathcal{A}$ an action, $R(s, a, s')$

the reward, and $T(s, a, s')$ the transition probability. The agent updates its policy from trajectories $(s_t, a_t, r_t)_{t\geq 0}$ to maximize expected cumulative rewards.

Here, the state $s = (\mathbf{x}, \mathbf{y}, \mathbf{q})$ captures the current environment with $\mathbf{x} = (\mathbf{x}_d^e)$, $\mathbf{q} = (\mathbf{q}_d)$ and $\mathbf{y} = (\mathbf{y}_S^v)$. The state space $\mathcal{S}$ encompasses all possible configurations. While larger state spaces offer more information, they reduce efficiency. Unlike previous DRL works that use redundant state details, we adopt a compact form. Instead, the neural network receives an *observation* which is a refined vector combining the state and useful derived information $o = (\mathbf{x}, \mathbf{y}, \mathbf{q}, \mathcal{G}_{\text{des}}, \mathcal{S}_{\text{des}})$.

Here, $\mathcal{G}_{\text{des}}$ contains a complete description of the current state of the graph, including exhaustively the capacity, congestion and latency of the edges, the resources and usage of the nodes. Concerning the services, $\mathcal{S}_{\text{des}}$ includes the problem constraints (maximum latencies lat^d, resources needed r_S, minimum and maximum flows $q_{d,\text{min}}$ and $q_{d,\text{max}}$) but also information about the current chosen solution (latency for each demand, proportion to maximum latency and length of each path). Some information of $\mathcal{S}_{\text{des}}$ are based on the problem instance and are only integrated within the dynamic setting to improve the RL adaptiveness. We initialize each episode with the known feasible solution associated with the cloud placement.

The action space is made of vectors of the following shape:

$$a = (a_{\text{type}}, a_{\text{node}}, a_{\text{route}}, a_q, S, d)\,.$$

The decision type a_{type} takes values in:

$$\{\text{replacement}, \text{reroute}, \text{quantities}\}\,.$$

Choosing *replacement* action results in displacing the service S to the node $a_{node} \in \mathcal{V}$. Choosing the *reroute* action makes the demand d being rerouted using the index $a_{\text{route}} \in \{0, \ldots, n\}$. Finally, selecting the action $a_{\text{type}} = \textit{quantities}$ adds a value $a_q \in \{0, \pm 1\}$ to the quantity q_d. We note that for each rerouting or service displacement, we reset the quantities of concerned demands to $q_{d,\text{min}}$. We choose to restrict the exploration to feasible solutions only. That is, an action is applied only if the next resulting solution remains feasible, thereby reducing the exploration phase. As a consequence, there should exist a path of states starting from the initial state to a state with a high objective value.

A common strategy is to define the reward function as the optimization objective, but this can lead to local maxima. To avoid this, the reward is designed to capture local improvements, encouraging better flows while penalizing inefficient routing and excessive service displacement. The reward function r_t used is:

$$\begin{cases} \max(0, q_{\text{total, next}} - q_{\text{best}}) & \text{if changing flow} \\ -10 & \text{if failed service displacement} \\ -2 & \text{if failed demand rerouting} \\ -1 & \text{otherwise} \end{cases}$$

Here, $q_{\text{total, next}}$ is the objective value of the next state, and q_{best} is the best value encountered during the episode. This RL setup leads to the experimental results discussed in the next section.

VI. Experimentation Design

This section presents numerical results to evaluate the proposed algorithm. First, we compare RL, the ILP approach, and the heuristic on a single instance, which is the most challenging for RL due to the lack of pre-training. Despite this, RL performs reasonably well by extracting valuable information from the problem. In the second part, we consider a dynamic case with changing placement requests. The RL approach, unlike ILP, adapts to new configurations quickly using pre-trained data, providing efficient placement and routing solutions crucial for real-time orchestration.

A. Instance Design

To match the warehouse instance of [5], the study examines a *star-shaped* structure, designed to reflect real-world network architectures with realistic parameters for edge computing use-cases [23]. Key network aspects include:

- Connectivity: User-to-user connections (e.g., Bluetooth) are proximity-based, with strong, low-latency connections to the Edge (5 ms base latency with low variation), though susceptible to saturation (at most 10 Gbps). Edge-to-Cloud connections offer higher capacity (1 Tbps), but latency increases with network load.
- Node Resources: A single resource type (e.g., CPU or memory) is considered, with the highest resource availability in the Cloud (10 PB), decreasing towards the lower network layers (10 TB on the edge).

We design an initial tree-shaped network, where users are connected to an Edge Server (e.g., a WAP in a warehouse). Services can be deployed locally, at the edge, or in the cloud. As the network expands, we arrange multiple parallel trees in a star configuration centered around the cloud server (Figure 1), which increases the complexity of the ILP due to potential network saturation. The RL approach addresses this by penalizing distant edge server placements with negative rewards or by producing infeasible solutions. We scale the instances up to 500 nodes, representing a medium-sized warehouse, accounting for all connected devices [24].

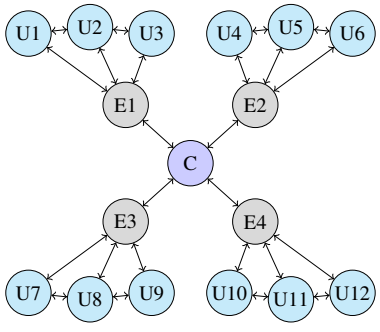

Fig. 1. Star-shaped network model with 4 edge servers.

B. Implementation Details

We compared three methods: Integer Linear Programming (ILP), the heuristic approach (Section IV-B), and RL (Section V). The benchmark was implemented in Python using the Gurobi solver [25] for ILP and the Gymnasium library [26] for RL. We converted network parameters to Gurobi variables for faster ILP solving and built an RL simulator using the `StableBaselines3` implementation. All solvers ran on CPUs with a memory cap of 16GB and 6 threads on an Intel Xeon 3.2GHz processor for a fair comparison. For Deep RL, we tested Deep Q-Learning, Advantage Actor-Critic, and Proximal Policy Optimization (PPO) [27]. Based on fast adaptation during the learning phase, we selected PPO for its stable learning and controlled policy adjustments.

The principle of PPO is to improve an agent's policy by optimizing a clipped objective function, which ensures stable and reliable updates. During the learning phase, the policy is approximated and updated using a neural network (NN). A second neural network is used to approximate the advantage function. Given the inputs in our problem, we use fully connected layers to extract information from the states. More specialized NN architectures have not yielded conclusive results in training our model. This algorithm is not guaranteed to converge, so the user must manually provide a stopping criterion. The most commonly used criterion is the average reward per episode. Typically, the average reward stabilizes after a certain number of episodes and ceases to improve. We choose to stop the algorithm when there is no further progress in the average episode reward.

VII. Results

A. Specific instance Solving

We focus now on the solving of a single instance. We start from the main feasible solution and we generated episodes using the agent policy. We then update the current policy along those trajectories. Given a network configuration with a fixed service set, the agent performs placement attempts and learns from next states. The final returned result is the best placement and routing discovered in the total training process. As we observe in Figure 2, the heuristic approach provides a consistent objective but is relatively slow. Nevertheless, it offers a good initial solution. The ILP resolution, on the other hand, requires a significant amount of time and lacks scalability: computation time quickly reaches 10 minutes, after which the solver fails due to insufficient RAM.

Notably, the Gurobi solver exhibits high memory consumption during problem construction. Once the memory usage exceeds 16GB, the solver is not even able to fully construct the problem and therefore cannot start the solving process, which explains the instance size limit reached by the ILP curve. Consequently, an intermediate method in terms of computational time is desirable. The PPO resolution serves as an excellent trade-off: its computational time remains nearly constant, and the obtained objective is correct, achieving approximately 75% of the optimal value on average. We point out that the RL runtime presented here includes the whole training process of the RL agent.

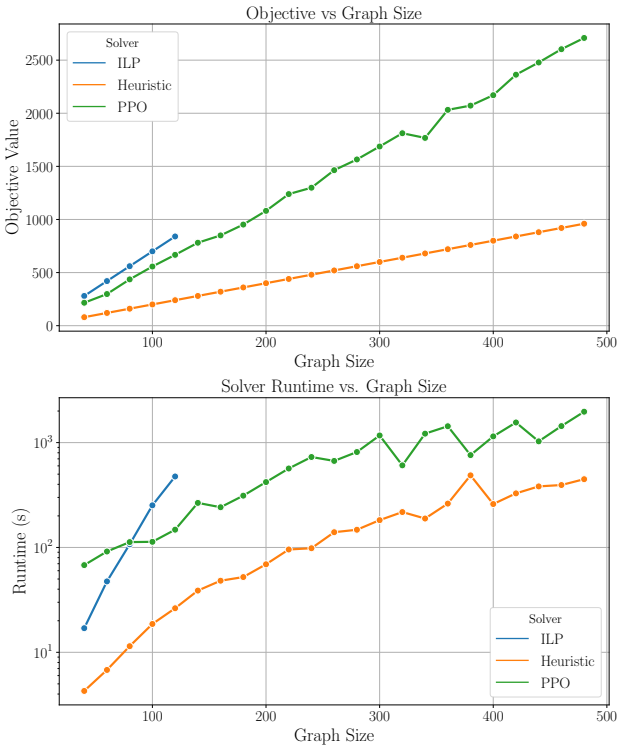

Fig. 2. Plot of objective reached vs. runtime depending on the star-shaped instance size.

B. Dynamic Problem Solving

The RL framework is not only expected to explore the environment, but also to adapt to unseen situations. In our context, we aim to evaluate the adaptation capabilities of a pretrained RL model. To this end, it is necessary to select a set of training instances that can reasonably reflect potential future scenarios. The design of the training process is therefore critical for ensuring meaningful experience: training and testing instances should present a realistic level of difficulty, and the gap between the two phases should remain reasonable.

For the training set, we generate instances with randomized origins and variable latency and quantity constraints. The test set is designed to emulate a dynamically changing network: across 10 consecutive situations, a single service is replaced with a new one at each step. For each evaluation instance, we compute the average runtime and performance for service flow placement, routing, and configuration.

While the RL model is able to adapt incrementally from one instance to the next one, the ILP solver must reset and resolve each instance from scratch. Table II presents the results, where the tree and star instances have 50 and 100 users, respectively. As expected, ILP had a long solving runtime, with RL being, on average, 50 times faster. ILP also

struggled with larger instances due to memory limitations. This highlights the challenge of linear programming in adapting to problem variations and leveraging problem structure for faster solving. The RL model achieved, on average, 75% of the optimal objective value of the ILP. The short adapting time of RL allows a deployment for orchestration purpose.

TABLE II
AVERAGE OBJECTIVES AND RUNTIMES IN A DYNAMIC CONTEXT.

	Instance	Tree-50	Tree-100	Star-50	Star-100
Runtime (s)	**ILP**	80.2	266.0	63.7	252.3
	Heuristic	4.7	17.7	6.5	19.5
	RL	1.7	4.6	1.2	1.9
Objective	**ILP**	370.0	700.0	480.0	700.0
	Heuristic	100.9	198.5	111.5	197.5
	RL	278.5	582.7	265.4	500.8
Opt. Gap (%)	**Heuristic**	72.7	71.6	76.8	71.8
	RL	24.7	16.8	44.7	28.5

VIII. CONCLUSION

We tackled the service placement problem constrained by latency and edge capacity, that arises in a real Edge Computing context. To speed up the solving process, we proposed a linear version of the problem, sacrificing some solution quality, and developed a heuristic based on network structure knowledge. Additionally, we introduced a RL framework, which showed strong performance in both static and dynamic cases. RL demonstrated scalability for large instances and adaptability by using pre-training to transfer network knowledge and handle new scenarios.

We aim to refine the RL learning process further to improve solution quality. One potential enhancement is integrating the three-part problem structure from the heuristic into a Hierarchical RL framework, which could better target the placement, routing, and flow tasks. Additionally, we plan to extend our approach to other industrial use cases with more precise latency models, such as those based on edge congestion, which may not be solvable by ILP.

REFERENCES

[1] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[2] C. Li, L. Toni, J. Zou, H. Xiong, and P. Frossard, "QoE-driven mobile edge caching placement for adaptive video streaming," *IEEE Transactions on Multimedia*, vol. 20, no. 4, pp. 965–984, 2017.

[3] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[4] B. Soykan and G. Rabadi, "Optimizing multi commodity flow problem under uncertainty: A Deep Reinforcement Learning approach," in *ICMLA 2023*. IEEE, 2023, pp. 1267–1272.

[5] V. Stavroulaki, E. C. Strinati, F. Carrez, Y. Carlinet, M. Maman, D. Draskovic, D. Ribar, A. Lallet, K. Mößner, M. Tosic *et al.*, "DEDICAT 6G-Dynamic coverage extension and distributed intelligence for human centric applications with assured security, privacy and trust: From 5G to 6G," in *2021 Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit)*. IEEE, 2021, pp. 556–561.

[6] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, "Edge intelligence: Paving the last mile of artificial intelligence with Edge Computing," *Proceedings of the IEEE*, vol. 107, pp. 1738–1762, 2019.

[7] F. A. Salaht, F. Desprez, and A. Lebre, "An overview of service placement problem in Fog and Edge Computing," *ACM Computing Surveys (CSUR)*, vol. 53, no. 3, pp. 1–35, 2020.

[8] M. Charikar, Y. Naamad, J. Rexford, and X. K. Zou, "Multi-Commodity flow with in-network processing," in *ALGOCLOUD 2018*. Springer, 2019, pp. 73–101.

[9] P. Bonami, D. Mazauric, and Y. Vaxès, "Maximum flow under proportional delay constraint," *Theoretical Computer Science*, vol. 689, pp. 58–66, 2017.

[10] N. Mazyavkina, S. Sviridov, S. Ivanov, and E. Burnaev, "Reinforcement learning for combinatorial optimization: A survey," *Computers & Operations Research*, vol. 134, p. 105400, 2021.

[11] K.-C. Tsai, L. Fan, L.-C. Wang, R. Lent, and Z. Han, "Multi-commodity flow routing for large-scale leo satellite networks using Deep Reinforcement Learning," in *2022 IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE, 2022, pp. 626–631.

[12] Y. Wang, Y. Li, T. Wang, and G. Liu, "Towards an energy-efficient data center network based on Deep Reinforcement Learning," *Computer Networks*, vol. 210, 2022.

[13] P. Frohlich, E. Gelenbe, J. Fiolka, J. Checinski, M. Nowak, and Z. Filus, "Smart SDN management of fog services to optimize QoS and energy," *Sensors*, vol. 21, no. 9, p. 3105, 2021.

[14] J. Li, H. Gao, T. Lv, and Y. Lu, "Deep Reinforcement Learning based computation offloading and resource allocation for MEC," in *WCNC*. IEEE, 2018, pp. 1–6.

[15] Z. Yang, Y. Liu, Y. Chen, and G. Tyson, "Deep Reinforcement Learning in cache-aided MEC networks," in *ICC*. IEEE, 2019, pp. 1–6.

[16] B. Qu, Y. Bai, Y. Chu, L. Wang, F. Yu, and X. Li, "Resource allocation for MEC system with multi-users resource competition based on Deep Reinforcement Learning approach," *Computer networks*, vol. 215, 2022.

[17] X. Jiao, H. Ou, S. Chen, S. Guo, Y. Qu, C. Xiang, and J. Shang, "Deep Reinforcement Learning for time-energy tradeoff online offloading in MEC-enabled industrial internet of things," *Transactions on Network Science and Engineering*, vol. 10, no. 6, pp. 3465–3479, 2023.

[18] J. Wang, L. Zhao, J. Liu, and N. Kato, "Smart resource allocation for mobile Edge Computing: A Deep Reinforcement Learning approach," *IEEE Transactions on Emerging Topics in Computing*, vol. 9, no. 3, pp. 1529–1541, 2019.

[19] N. Zhao, Y.-C. Liang, D. Niyato, Y. Pei, M. Wu, and Y. Jiang, "Deep Reinforcement Learning for user association and resource allocation in heterogeneous cellular networks," *IEEE Transactions on Wireless Communications*, vol. 18, no. 11, pp. 5141–5152, 2019.

[20] M. Klinkowski, J. Perelló, and D. Careglio, "Application of linear regression in latency estimation in packet-switched 5g xhaul networks," in *2023 23rd International Conference on Transparent Optical Networks (ICTON)*. IEEE, 2023, pp. 1–4.

[21] F. Liu, G. Tang, Y. Li, Z. Cai, X. Zhang, and T. Zhou, "A survey on edge computing systems and tools," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1537–1562, 2019.

[22] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network flows: theory, algorithms and applications*. Prentice hall, 1995.

[23] M. Lorenzo Hernández, "Edge computing for 5g networks - white paper," 5G PPP, Tech. Rep., 06 2022.

[24] K. Buntak, M. Kovačić, and M. Mutavdžija, "Internet of things and smart warehouses as the future of logistics," *Tehnički glasnik*, vol. 13, no. 3, pp. 248–253, 2019.

[25] Gurobi Optimization, LLC, "Gurobi Optimizer Reference Manual," 2024. [Online]. Available: https://www.gurobi.com

[26] M. Towers, A. Kwiatkowski, J. Terry, J. U. Balis, G. De Cola, T. Deleu, M. Goulao, A. Kallinteris, M. Krimmel, A. KG *et al.*, "Gymnasium: A standard interface for reinforcement learning environments," *arXiv preprint arXiv:2407.17032*, 2024.

[27] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.