A Deep Reinforcement Learning Agent for Distributed Task Offloading Based on Graph Neural Networks

Manuel Dileo, Matteo Zignani, Sabrina Gaito, Christian Quadri
Computer Science Department – Università degli Studi di Milano, Milan, Italy
manuel.dileo@unimi.it, matteo.zignani@unimi.it, sabrina.gaito@unimi.it, christian.quadri@unimi.it

***Abstract*—We propose a fully distributed task-offloading framework for mobile edge computing based on deep reinforcement learning (DRL) and graph neural networks (GNNs). Each edge node runs an agent that decides whether to execute an incoming latency-sensitive task locally or forward it to a neighbor using only local and two-hop information. The decision process is modeled as an MDP, represented through a GATv2 encoder, and trained with PPO using a reward that promotes deadline-compliant execution and low CPU usage. Simulations on realistic MEC topologies show that the agent outperforms shortest-path greedy baselines in the hardest scenarios and remains comparable elsewhere.**

***Index Terms*—Task offloading, deep reinforcement learning, graph neural networks, mobile edge computing.**

I. Introduction

IoT devices and connected vehicles increasingly generate low-latency workloads for smart-city, industrial, traffic, and safety applications. Since such devices have limited computation and energy budgets, tasks are often offloaded to more capable infrastructure. Cloud offloading, however, may not satisfy stringent latency requirements, while mobile edge computing (MEC) exposes heterogeneous, energy-limited, and geographically distributed resources close to the data sources [1]. This makes online task-placement decisions difficult, particularly when agents have only partial knowledge of the network.

Task offloading has been addressed with exact and heuristic methods, but their inference cost, limited adaptability, and weak generalization can be problematic in dynamic MEC deployments [2]. DRL offers an online decision-making framework [3], and GNNs provide a natural mechanism to encode graph-structured network states [4]. Existing GNN-based offloading approaches typically assume centralized control or global topology visibility [5]–[7]. In contrast, this paper proposes a fully distributed agent with two-hop visibility. The contributions are: (i) an MDP formulation for distributed task offloading under partial observability; (ii) a GATv2-based state encoder coupled with a PPO policy; and (iii) an evaluation against distributed greedy baselines on realistic MEC topologies.

II. System Model and Agent Design

We model the edge infrastructure as a directed graph $G = (N, E)$. Node $n \in N$ may have computational capacity μ_n CPU cycles/s, while each link $(n, m) \in E$ has bandwidth $\beta_{n,m}$. A task j is generated at time σ_j and is described by CPU demand η_j, input size ρ_j, and deadline τ_j. A task executed on node n at time t succeeds only if $t - \sigma_j \leq \tau_j$ and the node has enough free capacity, i.e., $\bar{\mu}_n(t) \geq \lceil \eta_j/(t - \sigma_j) \rceil$. When a task is forwarded, it waits in the outgoing link FIFO queue and is discarded if its transmission would end after the deadline.

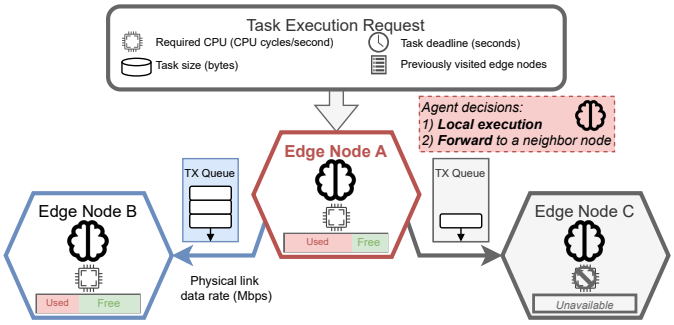

Fig. 1. Local agent decision process.

Each decision node chooses either local execution or forwarding to one of its neighbors, as depicted in Fig. 1. The observation is a two-hop subgraph centered on the decision node. Node features include normalized free CPU, a binary feasibility flag, and a normalized counter of previous visits by the same task. Edge features encode transmission queueing time. Task features encode normalized CPU demand, size, deadline, and elapsed-deadline ratio. A two-layer GATv2 encoder [8] produces node embeddings with attention weights that depend on neighboring node and edge features. The decision embedding concatenates the current node features, the task features, the decision-node embedding, and the neighbor embeddings, and is passed to an MLP policy.

The action space is discrete: a_0 executes locally and a_i forwards to the i-th neighbor, with neighbors sorted by available CPU to ensure consistent action semantics. Invalid actions are not masked; instead, the reward discourages them. The reward strongly penalizes forwarding to a missing neighbor, executing on a node without CPU, or discarding a task, while successful execution receives a positive reward reduced by CPU usage. Forwarding receives a small penalty based on elapsed deadline

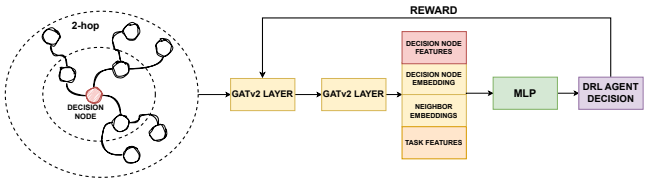

Fig. 2. Pipeline of the proposed approach. We combine a GNN based on an attention mechanism to extract node embeddings and an MLP for policy learning.

time and repeated visits. The policy is trained with Proximal Policy Optimization (PPO) [9], which is suitable for stochastic policies when several actions may be viable under similar network states. The internal representation of the state is shown in Fig. 2.

III. Simulation Setup

The environment is implemented with Gymnasium and SimPy, while PPO training uses Tianshou [11]. We use two MEC topologies from the dataset in [10]: `25N50E`, a mesh-like graph with 25 nodes and 50 edges, and `50N50E`, a tree-like graph with 50 nodes and 50 edges. The fraction of nodes equipped with CPU resources varies from 10% to 50%. Active nodes receive capacities from $\{50, 100, 200\}$ CPU cycles/s with small random perturbations; links have 100 Mbps bandwidth. Tasks arrive according to 0.5s plus an exponential inter-arrival component and are sampled from CPU demands $\{100, 200, 300\}$, deadlines $\{5, 6, 7\}$s, and sizes $\{50, 100, 200\}$ Mb, again with perturbations where appropriate.

We compare the DRL agent with three baselines. *Greedy Global* has complete topology, computing, and queue-state visibility and forwards toward the node minimizing estimated processing plus shortest-path transmission cost. *Greedy Local* applies the same logic but only inside the two-hop neighborhood and forwards randomly when no suitable node is found. *Random* uniformly selects local processing or a feasible forwarding action.

IV. Results

Table I reports mean task-completion rates and standard deviations over the evaluation scenarios. The DRL agent is strongest in the most constrained cases. With only 10% processing coverage, it reaches 0.53 on `25N50E`, compared with 0.50 and 0.51 for the global and local greedy policies, and 0.48 on `50N50E`, compared with 0.47 and 0.42. On `50N50E`, DRL also improves over the local greedy baseline at all coverage levels and matches or exceeds Greedy Global in four out of five settings. On `25N50E`, where ingress nodes are closer to computational nodes and the topology offers more forwarding alternatives, all informed policies converge to similar performance, with DRL within about one percentage point of the best greedy result for coverage levels above 10%.

The results indicate that the GNN encoder captures useful local capacity patterns despite partial observability. Greedy Global benefits from full visibility, but can concentrate many tasks on the same apparently optimal nodes because it does not reserve future capacity. Greedy Local avoids some concentration but must rely on random exploration outside its two-hop view, causing longer paths and deadline misses. Failure analysis shows that most DRL discards are due to insufficient time for further transmission, whereas missing-neighbor actions and execution on non-computing nodes are rare, confirming that the reward successfully encodes topological and computational constraints.

TABLE I
Comparison of agent performance. Each cell reports the mean and standard deviation of the percentage of completed tasks.

Network	Nodes CPU(%)	DRL	Greedy Global	Greedy Local	Random
`25N50E`	10%	**0.53 ± 0.12**	0.50 ± 0.13	0.51 ± 0.13	0.05 ± 0.05
	20%	0.76 ± 0.09	0.77 ± 0.10	**0.77 ± 0.09**	0.11 ± 0.06
	30%	0.86 ± 0.06	0.87 ± 0.08	**0.87 ± 0.06**	0.17 ± 0.07
	40%	0.89 ± 0.05	0.90 ± 0.08	**0.90 ± 0.05**	0.21 ± 0.07
	50%	0.91 ± 0.05	0.92 ± 0.07	**0.92 ± 0.05**	0.25 ± 0.08
`50N50E`	10%	**0.48 ± 0.11**	0.47 ± 0.10	0.42 ± 0.10	0.06 ± 0.02
	20%	**0.62 ± 0.08**	0.62 ± 0.07	0.58 ± 0.08	0.12 ± 0.03
	30%	**0.75 ± 0.06**	0.74 ± 0.05	0.71 ± 0.07	0.17 ± 0.03
	40%	**0.80 ± 0.06**	0.79 ± 0.05	0.78 ± 0.06	0.22 ± 0.03
	50%	0.83 ± 0.05	**0.83 ± 0.04**	0.82 ± 0.05	0.27 ± 0.03

The best result for each network and agent is in bold, the second-best is underlined

V. Conclusion

This work presents a fully distributed DRL solution for MEC task offloading. By combining a two-hop GATv2 state representation with PPO, the agent learns forwarding and execution policies that are competitive with shortest-path greedy approaches while requiring only local neighborhood information. The approach is particularly effective when processing nodes are sparse and decisions are most difficult. Future work will address more dynamic settings, such as UAV-assisted edge networks, and training on representative subgraphs to improve scalability and generalization.

References

[1] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Commun. Surveys Tuts.*, 2017.

[2] S. Dong *et al.*, "Task offloading strategies for mobile edge computing: A survey," *Computer Networks*, 2024.

[3] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, 2015.

[4] J. Gilmer *et al.*, "Neural message passing for quantum chemistry," in *ICML*, 2017.

[5] Z. Sun, Y. Mo, and C. Yu, "Graph-reinforcement-learning-based task offloading for multiaccess edge computing," *IEEE IoT J.*, 2023.

[6] S. Wang *et al.*, "Energy-efficient collaborative task offloading in multi-access edge computing based on deep reinforcement learning," *Ad Hoc Networks*, 2025.

[7] Y. Bo *et al.*, "Multi-agent reinforcement learning with graph representation for green edge-cloud computation offloading," *Computer Communications*, 2025.

[8] S. Brody, U. Alon, and E. Yahav, "How attentive are graph attention networks?" in *ICLR*, 2022.

[9] J. Schulman *et al.*, "Proximal policy optimization algorithms," *CoRR*, 2017.

[10] B. Xiang *et al.*, "A dataset for mobile edge computing network topologies," *Data in Brief*, 2021.

[11] J. Weng *et al.*, "Tianshou: A highly modularized deep reinforcement learning library," *JMLR*, 2022.