FAuNO: Semi-Asynchronous Federated Reinforcement Learning Framework for Task Offloading in Edge Systems

Frederico Metelo[†,*], Alexandre Oliveira[†], Stevo Racković[‡], Pedro Ákos Costa[†], Cláudia Soares[†]

[†]NOVA School of Science and Technology, Lisbon, Portugal

[‡]Instituto Superior Técnico, Lisbon, Portugal

June 9, 2025

Abstract

Edge computing addresses the growing data demands of connected-device networks by placing computational resources closer to end users through decentralized infrastructures. This decentralization challenges traditional, fully centralized orchestration, which suffers from latency and resource bottlenecks. We present **FAuNO**—*Federated Asynchronous Network Orchestrator*—a buffered, asynchronous *federated reinforcement-learning* (FRL) framework for decentralized task offloading in edge systems. FAuNO adopts an actor–critic architecture in which local actors learn node-specific dynamics and peer interactions, while a federated critic aggregates experience across agents to encourage efficient cooperation and improve overall system performance. Experiments in the *PeersimGym* environment show that FAuNO consistently matches or exceeds heuristic and federated multi-agent RL baselines in reducing task loss and latency, underscoring its adaptability to dynamic edge-computing scenarios.

1 Introduction

The growth of connected device networks, such as the Internet of Things (IoT), has led to a surge in data generation. Traditionally, Cloud Computing handled these computational demands, but increased network traffic and latency became apparent as these networks expanded [1]. Edge Computing (EC) extends the cloud by bringing computational resources closer to end-users, addressing latency and traffic issues [2, 3]. Despite distinguishing between Mobile Edge Computing (MEC) and Fog computing, this paper treats them interchangeably, focusing on their goal of minimizing device-to-cloud distances [4]. The EC paradigm

[*]fc.metelo@campus.fct.unl.pt

distributes computational resources, making centralized network orchestration inefficient. Centralization would require aggregating data at a single node, straining the network, and creating a single point of failure [5]. This highlights the value of decentralized orchestration, particularly through Task Offloading (TO). Optimal TO in such distributed environments involves managing multiple factors, including task latency, energy consumption, and task completion reliability [6]. Traditional optimization methods often struggle to efficiently manage these complex systems, due to the dynamic, time-varying, and complex environments of Edge Systems [7]. Reinforcement Learning (RL) [5, 6, 8], is a powerful candidate and dominant approach to solving the TO problem. Specifically, Multi-Agent Reinforcement Learning (MARL) has been explored as a promising solution for decentralized orchestration in Edge Systems [5, 8]. The ability of MARL agents to iteratively learn optimal strategies through simultaneous interaction with an environment makes them particularly suited for decentralized edge systems [9, 10]. Due to the nature of MARL, it is common to have some form of message exchange [11, 5] between participants, as this reduces the uncertainty generated by having multiple agents interacting simultaneously, making it particularly suitable for Federated Learning (FL), which has recently gained academic interest as an efficient and distributed approach to agent cooperation in learning [12, 13, 14, 15]. When FL is applied to MARL and the agents only have partial observability of the state, as is commonly the case in decentralized systems where obtaining information about all nodes comes at a premium, it creates a paradigm known as Vertical Federated Reinforcement Learning (VFRL) [16]. The MARL problem is transformed from one in which agents focus solely on their own objectives into a global optimization problem that accounts for the collective objectives of the participants in the federation. FL also minimizes the strain on the network by avoiding the exchange of large amounts of information, since agents only need to periodically share their learned updates that are aggregated into a global unified model solving the global objective. Effectively enabling agents to benefit from each other's knowledge while minimizing communication overhead.

We propose a buffered asynchronous approach to the VFRL problem based on Federated Buffering [17], so that the training of the model does not need to wait for the stragglers.

Motivations & Contributions

- We define the TO problem in edge systems, establishing a **structured framework** that captures its complexities.
- We model the TO problem as a **Partially-Observable Markov Game**, providing a deeper understanding of the interactions and uncertainties in decentralized edge environments.
- We propose a federated TO algorithm combining a **Semi-Asynchronous Federated Algorithm** (based on the FedBuff algorithm [17]) with Proximal Policy Optimization (PPO) [18]. This solution is key for **workload balancing through TO** in Federated AI Network Orchestrator (**FAuNO**), a conceptual

system for task processing in edge systems.

- We **extend the PeersimGym** environment to **support the exchange of FL updates** through the simulated network, details on the extension are in the annexA.
- We benchmark the FAuNO algorithm against baseline algorithms, showing superior or matching performance in task completion time and the number of tasks completed. Simulations in various network environments demonstrate FAuNO's effectiveness in optimizing TO, with significant improvements in orchestration efficiency.

The code for the algorithm and the baselines developed is available in the FAuNO repository.

Background & Related Work TO involves transferring computations from constrained devices to more capable ones, addressing the *what*, *where*, *how*, and *when* of offloading [19]. TO methods include vertical offloading to higher-tier systems [20], horizontal offloading among peers [6, 21], and hybrid approaches [5]. Offloading target selection may prioritize proximity [22, 4] or queue length [21], or consider unrestricted selection, accounting for consequences of offload failures. Failures are affected by factors like latency [23], resource capacity [22], energy shortages, or others [24]. This study focuses on *Binary TO* [25] for indivisible tasks with horizontal and vertical offloading.

RL has been applied to TO in both single-agent and multi-agent settings. In the single-agent setting, various approaches focus on modeling the TO problem as a Markov Decision Process (MDP) and leveraging RL techniques to find optimal solutions. In [21] a centralized decision-maker is used in a Fog Network using Software-Defined Networking (SDN) infrastructure. They formulate the problem as an MDP with global state visibility and solve it using Q-learning. Similarly, [22] tackles TO in ad-hoc mobile clouds by formulating an MDP and solving it with a Deep Q-Network (DQN). In [26], task-dependency requirements in MEC are considered, and a migration-enabled multi-priority sequencing algorithm is proposed to minimize deadline violations and enhance reliability by applying the Deep Deterministic Policy Gradient (DDPG) algorithm to find the optimal TO policy. In [27] address real-time TO in MEC networks with cloud support. Using SARSA-based OD-SARSA, the problem is modeled as an MDP, and an on-policy, one-step bootstrapping approach to enable online learning is adopted. In [28], a task-offloading scheme for MEC systems is proposed in which mobile devices act as independent agents responsible for delay-sensitive tasks. The exploration-exploitation trade-off is managed by integrating a parameterized index function into a DQN framework. Finally, in [6], the TO problem is simplified by using Multi-Armed Bandits for binary offloading in Fog Networks, optimizing latency and energy through bandit feedback and sequential decision-making. Their proposed BLOT algorithm reduces complexity while maintaining efficiency.

MARL methods tackle resource allocation and collaboration in heterogeneous, partially observable environments. In [29] TO in Multi-Fog systems is modeled

as a Stochastic Game, and a Deep Recurrent Q-Network (DRQN) with Gated Recurrent Units is used to handle the partial state observations.

FRL has been explored as a mechanism for TO in Edge systems, emphasizing agent cooperation. However, most RL research in TO focuses on parallel RL, where agents act on independent environment replicas, not considering the uncertainty introduced by shared environments. In [15], a multi-TO algorithm is developed that uses a Double Deep Q-Network (DDQN) and K-Nearest neighbors to obtain local offloading schemes. The agents then participate in training a global algorithm utilizing a weighted federated averaging algorithm. A unary outlier detection technique is used to manage stragglers. In [13], an FRL-based model for frame aggregation and offloading Internet of Medical Things (IoMT) data to Edge Servers is proposed, that optimizes for energy consumption and latency A Hierarchical FRL algorithm I used to train models in Wireless Body Area Networks to minimize local energy consumption, with learned parameters offloaded to an edge server and aggregated at a central server. In [12] an FRL-based joint TO and resource allocation algorithm to minimize energy consumption on the IoT devices in the Network, considering a delay threshold and limited resources is proposed. The considered approach uses DDPG locally and a FedAvg [30] based algorithm for the global solution. In [31], a binary TO algorithm for MEC systems is proposed, employing dueling and double DQN with LSTM to improve long-term cost estimation for delay-sensitive tasks. In [14], a scenario with multiple agents in the same environment is considered, and FEDOR – a Federated DRL framework for TO and resource allocation to maximize task processing is proposed. In FEDOR, Edge users collaborate with base stations for decisions, and a global model is aggregated using FedAvg, with an adaptive learning rate improving convergence. Although FEDOR considers multiple agents in the same scenario, the decision-making depends on base stations for smoothing the offloading decisions of the multiple agents. In [5], FLoadNet is proposed as a framework that combines local actor networks with a centralized critic, trained synchronously in a federated manner, to enable collaborative task offloading in Edge-Fog-Cloud systems. Their solution learns what information to share between nodes to enhance cooperation and their offloading scheme learns the optimal paths for tasks to take through a Software-defined Network. In the Industrial IoT(IIoT) setting with dependency-based tasks, [32] propose SCOF that considers a Federated Duelling DQN, that is aggregated with a FedAvg-based approach and utilizes differential privacy (DP) to improve the security of the update exchanges. Focusing on selecting the best offloading targets from a pool of Edge Servers.

Lastly, none of the studied solutions uses an environment that facilitates the comparison of the proposed algorithms, which we do by training and benchmarking our solution in the PeersimGym environment [33]. The comparison with the related work is summarized in Table 1.

Table 1: Comparison of RL-based Task Offloading approaches.

Work	Multi-Agent	Federated	Actor-Critic	Partially Obs.	Shared Env.	Buffered Async.	OSS Env.
Baek et al. (2020) [29]	✓		DRQN	✓			
Baek et al. (2022) [5]	✓	✓	✓	✓	✓		
Zang et al. (2022) [14]	✓	✓	DQN	✓	✓		
Li et al. (2023) [15]	✓	✓	DDQN				
Peng et al. (2024) [32]		✓	Dueling DQN	✓			
FAuNO (ours)	✓	✓	✓	✓	✓	✓	✓

2 Federated Task Offloading Problem

In this section, we elaborate on the system modeling of our Edge System and formulate the TO problem as a global optimization problem that will be solved by all the participants in the network orchestration. Lastly, we formulate the local learning problem of the participants as a Partially Observable Markov Game (POMG).

2.1 System Model

We consider a set of nodes $\mathcal{W} = W_1, \ldots, W_k$ comprising the network entities (e.g., edge servers, mobile users). These nodes offer computational resources to a set of clients, such as IoT sensors that require processing for collected data. Time is discretized into equidistant intervals $t \in \mathbb{N}_0$.

The system includes two types of entities, as illustrated in Fig. 1. **Clients** generate computational workloads in the form of tasks for accessible nodes, following a Poisson process with rate λ; the set of all clients is denoted by $\dot{C}$. **Workers**, denoted by $\dot{W}$, provide computational resources and are represented as nodes with specific properties. Each worker W^n maintains a task queue Q_t^n at time t, with a maximum capacity $Q_{\max}^n$. The Workers are characterized by the number of CPU cores N_ϕ^n, the per-core frequency ϕ^n (in instructions per time step), and a transmission power budget $\mathcal{P}_n$. Workers periodically share their state with neighbors. A single machine may simultaneously act as both a worker and a client.

Task Model Computational requirements are modeled as tasks in our system. Let $T = \{\tau_i\}, i \in \mathbb{N}$ be the set of all tasks, where a task with ID i is represented as $\tau_i = \langle i, \rho_i, \alpha_i^{\text{in}}, \alpha_i^{\text{out}}, \xi_i, \delta_i \rangle$, with the following attributes: i as a unique task identifier, ρ_i as the number of instructions to be processed, α_i^{in} as the total input data size, α_i^{out} as the output data size, ξ_i as the CPU cycles per instruction, and δ_i as the task deadline, or maximum allowed latency for the return of results. A task may be dropped if it arrives at a node with a full queue or if its deadline expires.

Communication Model The communication model defines the latency of message transmission between nodes in the same neighborhood, where a node can only communicate directly with its neighbors. Each entity within a node

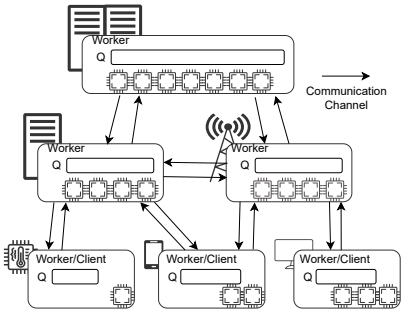


Figure 1: Edge System Architecture following our system model. The workers are capable of independently offloading tasks, exchanging information and FL model updates through the communication channels.

can send and receive messages, modeled as the tuple $\langle \omega_i, \omega_j, \alpha \rangle$, where ω_i is the origin node ID, ω_j is the destination node ID, and α represents the message size. To measure transmission delay, we consider the *Shannon-Hartley* theorem [34]. According to this theorem, the communication latency for transmitting α bits between nodes W_i and W_j is given by:

$$T_{i,j}^{\text{comm}}(\alpha) = \frac{\alpha}{B_{i,j} \log(1 + 10^{\frac{\mathcal{P}_i + G_{i,j} - \omega_0}{10}})}, \tag{1}$$

where $T_{i,j}^{\text{comm}}(\alpha)$ is the transmission time, $B_{i,j}$ is the bandwidth between nodes, $\mathcal{P}_i$ is the source node's transmission power, $G_{i,j}$ is the channel gain, and ω_0 is the noise power. This formulation evaluates the communication overhead associated with TO in Edge networks.

2.2 Problem Formulation

We aim to optimize workload orchestration based on task processing latency and avoid the loss of tasks due to resource exhaustion. At time-step t, we define the system as a tuple $\langle W, \dot{W}, \dot{C}, \mathcal{C}, T_t \rangle$. Each node can decide to process a task locally or offload it to a neighbor, represented by the action variable a_t^i for worker i. The delay incurred by the decisions of all agents is given by:

$$D_n(T_t, \dot{W}) = \sum_{a_t^n} d(a_t^n) \tag{2}$$

The function $d(a_t^n)$ represents the local extra delay of the decision made by agent n, defined as:

$$d(a_t^n) = \chi_D^{\text{wait}} T_{i,a_t^n}^{\text{wait}}(\tau_k) + \chi_D^{\text{comm}} T_{i,a_t^n}^{\text{comm}}(\alpha_k^{\text{out}}) + \chi_D^{\text{exc}} T_{i,a_t^n}^{\text{exc}}(\tau_k). \tag{3}$$

This function is a weighted sum of three time-related terms associated with offloading decisions, based in [21, 35]. The delay function incorporates hyper-parameters χ_D^{wait}, χ_D^{comm}, and $\chi_D^{\text{exc}} \geq 0$. The delay terms for a given action a_t^n are:

$$T^{\text{wait}}_{\dot{w}^n, a_t^n}(\tau_k) = \frac{Q_n^t}{N_\phi^n \phi_n} + \sum_{j \neq n} \frac{Q_j}{N_\phi^j \phi_j} I_j(a_t^n), \tag{4}$$

which represents the waiting time for task τ_k in the queue of node W_i (and W_j in case it is offloaded). Here, ϕ_i is the computing service rate of node W_i, and N_ϕ^i is its number of processors. The indicator function $I_j(a_t^n)$ equals 1 if the task is processed locally on node w_n(i.e., $I_n(a_t^n) = 1$) or offloaded to a neighboring node W_j (i.e., $I_j(a_t^n) = 1$) with $j \neq n$. The term $T^{\text{comm}}_{i,a_t}(\alpha_k^{\text{out}})$ denotes the communication cost of TO, defined as a delay according to Equation ((1)), where a_t^n indicates the neighboring node i. If the task is executed locally, this term becomes zero. Lastly, the term:

$$T^{\text{exc}}_{i,a_t^n}(\tau_k) = \frac{t\rho_k \xi_k}{N_\phi^{a_t^n} \phi_{a_t^n}} - \frac{t\rho_k \xi_k}{N_\phi^n \phi_n} \tag{5}$$

represents the difference in execution costs for tasks processed locally versus those processed at the target node. Here, ρ_k denotes the number of instructions per task, and ξ_k represents the number of CPU cycles per instruction.

Hence, to minimize the delay in processing the tasks at each time-step, we wish to find the solution to the constrained optimization problem:

$$\min_{\{a_t^n\}_{\dot{w}_n \in \dot{W}}} \quad D(T_t, \dot{W}) \tag{6}$$

$$\text{subject to} \quad C_1 : \delta_i \leq t_C \tag{7}$$

$$C_2 : Q^n \leq Q^n_{max} \tag{8}$$

The solution must also respect a set of constraints to minimize task drops: no tasks may be offloaded to overloaded nodes, as indicated by constraint (7), and no tasks should breach their deadlines. Additionally, no node should exceed its computational resource limit, as outlined in constraint (8).

2.2.1 Partially-Observable Markov Game

To solve the TO problem with distributed and decentralized agents, we define it as a Partially-Observable Markov Game (POMG) [36], represented as a tuple $\langle \mathcal{N}, \mathcal{S}, \mathcal{O}, \Omega, \mathcal{A}, P, R \rangle$. Here, $\mathcal{N} = 1, \ldots, n$ denotes a finite set of agents; $\mathcal{S}$ is the global state space that includes the information about all the nodes and tasks in the network; $\Omega = \{o_i\}_{i \in \mathcal{N}}$ is the set of Observation Spaces, where o_i is the observation space of agent i, that has information about the workers in its neighborhood, $\dot{N}_n$; $\mathcal{O} = \{O_i\}_{i \in \mathcal{N}}$ s.t.$O_i : S \to o_i$ is the set of Observation Functions for each agent, where O_i is the observation function of agent i. The observation function maps the state to the observations for each agent, each agent's observations includes information about it's local computational and

communication resources, the information shared by it's neighbors on the same, and information about the next offloadable task. $\mathcal{A} = \{A_i\}_{i \in \mathcal{N}}$ is the set of action spaces, where A_i is the action space of agent i. Each agent is able to select whether to send a task to one of its neighbors or process it locally; $P : S \times \mathcal{A} \to S$ is the unknown global state transition function; Lastly, $R = \{R_i\}_{i \in \mathcal{N}}$ s.t.$R_i \in S \times \mathcal{A} \times S \to \mathbb{R}$ - is the reward function for agent i. Each agent will consider the local reward given by:

$$R_i(s_t, a_t^i) = d(a_t^i) + \chi_O O(s_t, a_t^i) \tag{9}$$

Where $\chi_O \geq 0$ is a weighting parameter; And the term $\chi_O O(s_t, a_t^i)$ captures the distance to overload the workers involved in an offloading, we define the function as, $O(s_t, a_t) = -\log(p_t^{Oa_t})/3$. And, $p_t^{Oa_t} = max(0, \frac{Q_{a_t}^{\max} - Q_{a_t}}{Q_{a_t}^{\max}})$, represents the distance to overloading node W_i, and $Q'_{a_t} = min(max(0, Q_{a_t} - \phi_{a_t}) + 1, Q_{a_t}^{\max})$ is the expected state of the queue at node W_{a_t}, after taking action a_t.

3 FAuNO

We now present FAuNO - Federated Asynchronous (u) Network Orchestration - a framework designed to provide remote computing power to a group of clients, while load balancing in a decentralized fashion with an FRL-based algorithm. The FAuNO nodes also act as the workers from the modelling. A detailed breakdown of FAuNO node components is provided in the annex B.

3.1 Federated Reinforcement Learning Task Offloading Solution

We consider two components to our solution a local component and a global component. The local component utilizes Proximal Policy Optimization [18], this algorithm belongs to the Actor-Critic family of algorithms, meaning that there is an actor component that learns to interact directly with the environment and a critic component that learns to evaluate the actor and guides the training. We federated the critic network in our solution so that the experience of all the agents is used to guide the local learning of the agents. Our global solution used to train the Critic network is based on FedBuff [17], a buffered asynchronous aggregation method. The crux of the proposed algorithm is that by federating the global network, we mitigate the selfish behavior of the agents because the global objective incorporates the local objectives of all participants.

Local Policy Optimization PPO is a Policy gradient method, therefore it is based on the Policy Gradient Theorem proposed by Sutton et al. in [37]. This theorem shows that we can train a policy approximator function, by computing an estimator of the policy gradient and then doing stochastic gradient ascent. The formulation of the policy gradient we consider is given by [18]:

$$\hat{g} = \mathbb{E}_t \left[\nabla_\theta \log \pi_\theta(a_t \mid s_t) \hat{A}_t \right] \tag{10}$$

Here, π_θ is the policy being optimized, and $\hat{A}_t$ is an estimator for the Advantage Function computed with Generalized Advantage Estimation [38].

The PPO algorithms work by running a policy for a parametrizable number of steps and storing information not only about the state, action, and reward, but also, about the probability assigned to the chosen action. This information is then utilized in the next training step for computing the objective function, which in the case of PPO-Clip is given by:

$$L^{\text{CLIP}}(\theta) = \mathbb{E}_t\Big[\min\Big(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\Big)\Big] \tag{11}$$

Here, $r_t(\theta)$ denotes the probability ratio:

$$r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}. \tag{12}$$

The rationale behind using the probability ratio is that when an action with a higher advantage is selected and the new policy assigns a higher probability to that action, then the ratio will be bigger than zero, obtaining an overall higher objective. Conversely, if the probability increases for a negative advantage, then the objective function decreases faster. The clipping and the minimum are set in place so that the final objective is a lower bound (i.e., a pessimistic bound) on the unclipped objective [18]. This prevents excessive deviations from the original policy in a single update, avoiding large, harmful updates caused by outliers.

Due to the Actor-Critic nature of the PPO algorithm, two components must be trained: the critic and the actor. Consequently, when utilizing automatic differentiation frameworks, like PyTorch, Schulman et al. [18] recommend maximizing the following objective:

$$L_t^{\text{CLIP+VF+S}}(\theta, w) = \mathbb{E}_t\Big[L_t^{\text{CLIP}}(\theta) - c_1 L_t^{\text{VF}}(w) + c_2 S_{\pi_\theta}(s_t)\Big], \tag{13}$$

where we have the objective of the Actor, L^{CLIP}, as shown in eq. (11). The critic's loss function, $L^{\text{VF}}(w)$, where w is the parameters of the critic network, given by,

$$L_t^{\text{VF}}(w) = \left(r + \gamma V(s_{t+1}) - V(s_t)\right)^2. \tag{14}$$

And an entropy term, $S[\pi_\theta](s_t)$ to promote exploration. The c_1 and c_2 are coefficients weighing the different components of the objective.

And, because we are exploring an FL approach, the agents will share the gradients they obtained while training the local critic networks. The algorithm used for local training in the FAuNO nodes is shown in algo. 1 in annex C.

Global Algorithm The Global Algorithm is responsible for managing the federation and aligning the local solutions from each participant to derive

the global solution. We employ a semi-asynchronous method, inspired by the algorithm proposed in FedBuff [17], to tackle the following optimization problem:

$$\min_w f(w), \text{ s.t. } f(w) := \frac{1}{m}\sum_{k=1}^{m} p_k l_k(w, \theta_k). \tag{15}$$

Here, m represents the number of participants, and θ_k denotes the parameters of the actor-network for the agent identified by k. The variable w corresponds to the global critic parameters, and p_k is the weight assigned to agent k's loss function. In our algorithm, l_k is equivalent to $-L_t^{\text{CLIP+VF+S}}$.

To solve this problem, we propose a federated algorithm with a single global manager node that all agents can communicate with via multi-hop. There must always be a path between a participant in the federation and the global node. The algorithm initializes all agents with identical Critic network weights. After completing a set number of training steps, each agent sends its Critic gradients to the global manager node. The global manager stores these gradients in a buffer. If multiple updates are received from the same agent, the newer updates replace the older ones. Upon receiving at least K updates, the global manager node aggregates them using a weighted average, where agents completing more training steps are weighed more. The updated Critic model is then broadcast to all agents, which replace their local Critic weights before resuming training. Fig. 2 outlines the global training process. Subsequent sections detail the local PPO algorithm and the FedBuff-based global solution.

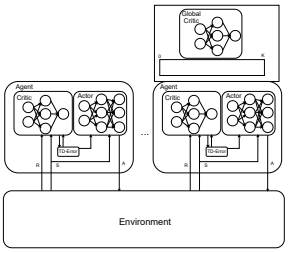
Once K updates have arrived, the global manager performs a FedAvg-style aligning:

$$\hat{w} = w + \sum_{k \in \bar{K}} p_k \nabla w_k \tag{16}$$

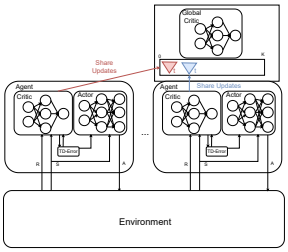
Here, $\hat{w}$ represents the new global model, and $\bar{K} \geq K$ represents the set of received updates, that may be bigger than the K updates required. The coefficient p_k is calculated based on the number of update steps each agent performed, ensuring that $\sum_{k \in \bar{K}} p_k = 1$. The global algorithm can be observed in algo. 2 in annex C.

4 Performance Evaluation

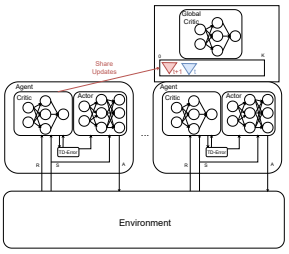
In this section, we evaluate FAuNO's performance using two standard TO metrics: average task completion time and percentage of completed tasks—the proportion of tasks that were created and whose results were successfully returned to the originating client. We compare FAuNO against two baseline algorithms: Least Queues(LQ), which offloads tasks to the worker with the shortest queue, and an adaptation of the synchronous FRL solution, SCOF [32], tailored to our setting. We benchmark our solution using PeersimGym [33], with realistic topologies generated by the Ether tool [39]. These are structured as hierarchical star topologies, where a stronger server provides resources to a small set of client nodes, and a more powerful central server supports the intermediate servers. We



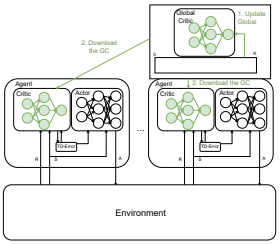
(a) Agents are training their local models.



(b) Updates of agents arrive at the Global Manager and are stored in a buffer.



(c) A second update from a known agent arrives and the old update is swapped.



(d) Global model updated with aggregated agent updates, and the latest global critic model is sent to the agents.

Figure 2: Flow of the FAuNO global algorithm.

also evaluate on synthetic topologies composed of 10 and 15 nodes randomly distributed in a 100×100 area. In these settings, the number of high-capacity nodes remains fixed, while the number of client nodes increases. All tests use realistic task distributions enabled by PeersimGym's integration with the Alibaba Cluster Trace workload generator [40], that we have rescalled to batter suit the considered edge devices. Each algorithm is trained for 40 episodes, for a total of 400,000 steps, and evaluated during training all presented results are the average result for the metric in question across the 40 episodes. Additional details on the testing setup, details on the topologies and workloads used are provided in Annex E.

4.1 Ether based topologies

Tables 2 and 3 report the average percentage of completed tasks and the average task response time for each algorithm, across varying topologies and task arrival rates (λ). A general trend is that performance degrades as the number of nodes and λ increase, primarily due to faster exhaustion of computational and shared resources (e.g., cloudlets). Despite this, FAuNO consistently achieves the highest task completion rates in most scenarios and outperforms the heuristic baselines

Table 2: Finished Tasks (as a ratio of total tasks created)

Algorithm	$\lambda = 0.5$		$\lambda = 1$		$\lambda = 2$	
	2	4	2	4	2	4
FAuNO	**0.967±0.014**	**0.957±0.008**	**0.956±0.015**	**0.957±0.010**	0.893±0.015	0.896±0.012
LQ	0.943±0.004	0.948±0.003	0.943±0.004	0.948±0.003	**0.910±0.005**	**0.915±0.006**
SCOF	0.939±0.032	0.926±0.032	0.939±0.053	0.926±0.037	0.740±0.052	0.680±0.039

Table 3: Response Time (in simulation ticks)

Algorithm	$\lambda = 0.5$		$\lambda = 1$		$\lambda = 2$	
	2	4	2	4	2	4
FAuNO	148.22±12.36	148.82±6.43	175.69±12.09	175.86±6.42	215.82±8.70	209.29±6.28
LQ	258.41±9.07	239.59±6.64	278.21±8.92	256.99±7.66	296.60±7.79	269.76±5.07
SCOF	**127.14±24.34**	**75.46±41.04**	**66.43±25.70**	**45.70±15.10**	**102.19±24.07**	**69.61±17.89**

in response time. Although SCOF achieves lower response times, it does so at the cost of significantly reduced task completion. We attribute this to the heterogeneity of the nodes, making it so that using a single global network without the local specialization sets an orchestration strategy that is too general which leads to offloading from high-capacity nodes when they fill up, leading to task expiration and exclusion from the response time calculation.

4.2 Random topology

Table 4: Response Time (in simulation ticks)

Algorithm	$\lambda = 0.5$		$\lambda = 1$		$\lambda = 2$	
	10	15	10	15	10	15
FAuNO	301.32±12.34	404.53±7.55	337.66±10.46	411.09±4.79	353.16±9.27	385.42±4.90
LQ	377.32±11.34	439.73±8.26	401.64±17.04	433.19±5.82	408.26±10.38	388.61±4.51
SCOF	308.27±14.55	384.71±19.17	337.99±13.87	394.92±22.43	349.90±17.52	363.63±8.93

As in the Ether networks, increasing the network size significantly degrades performance in both task completion rate(tab. 4 and response time (tab. 5). This effect is exacerbated by the considered topology maintaining a fixed number of cloudlets while increasing the number of client nodes. In contrast to the more structured topology with a single cloudlet, the LQ algorithm slightly outperforms FAuNO in task completion. However, its disregard for local processing capabilities results in substantially higher response times. SCOF exhibits the opposite behavior: due to the presence of more powerful nodes distributed across the network compared to the Ether scenario, its centralized, non-personalized orchestration favors local processing. This reduces communication overhead and improves response time, but at the expense of lower task completion. FAuNO achieves a balanced trade-off between the two metrics. As the number of nodes

Table 5: Finished Tasks (as a ratio of total tasks created)

Algorithm	$\lambda = 0.5$		$\lambda = 1$		$\lambda = 2$	
	10	15	10	15	10	15
FAuNO	0.886±0.036	0.769±0.035	0.785±0.050	0.596±0.032	0.631±0.035	0.388±0.027
LQ	0.912±0.0010	0.781±0.014	0.858±0.019	0.654±0.023	0.768±0.037	0.441±0.032
SCOF	0.8720±0.0360	0.7036±0.0542	0.7707±0.0548	0.5703±0.0413	0.6129±0.0608	0.3473±0.0249

grows and the proportion of weaker nodes increases, in some tests, FAuNO even surpasses SCOF in response time while maintaining a competitive task completion rate relative to LQ.

Conclusion

We addressed the decentralized Task Offloading (TO) problem in edge systems by modeling it as a cooperative objective over a federation of agents, formalized within a Partially Observable Markov Game (POMG). To this end, we proposed **FAuNO**, a novel Federated Reinforcement Learning (FRL) framework that integrates buffered semi-asynchronous aggregation with local PPO-based training. FAuNO enables decentralized agents to learn task assignment and resource usage strategies under partial observability and limited communication, while maintaining global coordination through a federated critic. Empirical evaluation in the PeersimGym environment confirms FAuNO's consistent advantage over heuristic and FRL baselines in reducing task loss and latency, highlighting its adaptability to dynamic and heterogeneous edge settings.

References

[1] M. Min, L. Xiao, Y. Chen, P. Cheng, D. Wu, and W. Zhuang, "Learning-based computation offloading for iot devices with energy harvesting," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 2, pp. 1930–1941, 2019.

[2] M. Muniswamaiah, T. Agerwala, and C. C. Tappert, "A survey on cloudlets, mobile edge, and fog computing," in *8th IEEE CSCloud/7th IEEE EdgeCom*, 2021.

[3] B. Varghese and R. Buyya, "Next generation cloud computing: New trends and research directions," *Future Generation Computer Systems*, vol. 79, pp. 849–861, 2018.

[4] S. Yu, X. Chen, Z. Zhou, X. Gong, and D. Wu, "When deep reinforcement learning meets federated learning: Intelligent multitimescale resource management for multiaccess edge computing in 5g ultradense network," *IEEE Internet of Things Journal*, vol. 8, no. 4, pp. 2238–2251, 2020.

[5] J. Baek and G. Kaddoum, "Floadnet: Load balancing in fog networks with cooperative multiagent using actor-critic method," *IEEE Trans. Netw. Serv. Manage.*, 2023.

[6] Z. Zhu, T. Liu, Y. Yang, and X. Luo, "Blot: Bandit learning-based offloading of tasks in fog-enabled networks," *IEEE Trans. Parallel Distrib. Syst.*, 2019.

[7] Z. Xu, J. Tang, J. Meng, W. Zhang, Y. Wang, C. H. Liu, and D. Yang, "Experience-driven networking: A deep reinforcement learning based approach," in *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, 2018, pp. 1871–1879.

[8] Z. Gao, L. Yang, and Y. Dai, "Large-scale computation offloading using a multi-agent reinforcement learning in heterogeneous multi-access edge computing," *IEEE Transactions on Mobile Computing*, 2022.

[9] L. Lin, W. Zhou, Z. Yang, and J. Liu, "Deep reinforcement learning-based task scheduling and resource allocation for noma-mec in industrial internet of things," *Peer-to-Peer Networking and Applications*, vol. 16, no. 1, pp. 170–188, 2023.

[10] F. Zhang, G. Han, L. Liu, Y. Zhang, Y. Peng, and C. Li, "Cooperative partial task offloading and resource allocation for iiot based on decentralized multi-agent deep reinforcement learning," *IEEE Internet of Things Journal*, 2023.

[11] K. Zhang, Z. Yang, H. Liu, T. Zhang, and T. Basar, "Fully decentralized multi-agent reinforcement learning with networked agents," *CoRR*, vol. abs/1802.08757, 2018.

[12] X. Chen and G. Liu, "Federated deep reinforcement learning-based task offloading and resource allocation for smart cities in a mobile edge network," *Sensors*, vol. 22, no. 13, p. 4738, 2022.

[13] P. Consul, I. Budhiraja, R. Arora, S. Garg, B. J. Choi, and M. S. Hossain, "Federated reinforcement learning based task offloading approach for mec-assisted wban-enabled iomt," *Alexandria Engineering Journal*, vol. 86, pp. 56–66, 2024.

[14] L. Zang, X. Zhang, and B. Guo, "Federated deep reinforcement learning for online task offloading and resource allocation in wpc-mec networks," *IEEE Access*, vol. 10, pp. 9856–9867, 2022.

[15] J. Li, Z. Yang, X. Wang, Y. Xia, and S. Ni, "Task offloading mechanism based on federated reinforcement learning in mobile edge computing," *Digital Communications and Networks*, vol. 9, no. 2, pp. 492–504, 2023.

[16] J. Qi, Q. Zhou, L. Lei, and K. Zheng, "Federated reinforcement learning: Techniques, applications, and open challenges," *Intelligence & Robotics*, 2021, arXiv:2108.11887 [cs].

[17] J. Nguyen, K. Malik, H. Zhan, A. Yousefpour, M. G. Rabbat, M. Malek, and D. Huba, "Federated learning with buffered asynchronous aggregation," *CoRR*, vol. abs/2106.06639, 2021.

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

[19] M. Fahimullah, S. Ahvar, and M. Trocan, "A review of resource management in fog computing: Machine learning perspective," 2022, arXiv:2209.03066 [cs].

[20] X. Qiu, L. Liu, W. Chen, Z. Hong, and Z. Zheng, "Online deep reinforcement learning for computation offloading in blockchain-empowered mobile edge computing," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 8, pp. 8050–8062, 2019.

[21] J.-y. Baek, G. Kaddoum, S. Garg, K. Kaur, and V. Gravel, "Managing fog networks using reinforcement learning based load balancing algorithm," in *2019 IEEE Wireless Communications and Networking Conference (WCNC)*, 2019, pp. 1–7.

[22] D. Van Le and C.-K. Tham, "A deep reinforcement learning based offloading scheme in ad-hoc mobile clouds," in *IEEE INFOCOM WKSHPS*, 2018, pp. 760–765.

[23] F. Dai, G. Liu, Q. Mo, W. Xu, and B. Huang, "Task offloading for vehicular edge computing with edge-cloud cooperation," *World Wide Web*, vol. 25, no. 5, pp. 1999–2017, 2022.

[24] X. Peng and et al., "Deep reinforcement learning for shared offloading strategy in vehicle edge computing," *IEEE Systems Journal*, 2022.

[25] A. M. A. Hamdi, F. K. Hussain, and O. K. Hussain, "Task offloading in vehicular fog computing: State-of-the-art and open issues," *Future Generation Computer Systems*, vol. 133, p. 201–212, 2022.

[26] S. Liu, Y. Yu, X. Lian, Y. Feng, C. She, P. L. Yeoh, L. Guo, B. Vucetic, and Y. Li, "Dependent task scheduling and offloading for minimizing deadline violation ratio in mobile edge computing networks," *IEEE Journal on Selected Areas in Communications*, vol. 41, no. 2, pp. 538–554, 2023.

[27] T. Alfakih, M. M. Hassan, A. Gumaei, C. Savaglio, and G. Fortino, "Task offloading and resource allocation for mobile edge computing by deep reinforcement learning based on sarsa," *IEEE Access*, vol. 8, pp. 54 074–54 084, 2020.

[28] X. Liu, S. Jiang, and Y. Wu, "A novel deep reinforcement learning approach for task offloading in mec systems," *Applied Sciences*, vol. 12, no. 21, 2022.

[29] J. Baek and G. Kaddoum, "Heterogeneous task offloading and resource allocations via deep recurrent reinforcement learning in partial observable multifog networks," *IEEE Internet of Things Journal*, vol. 8, no. 2, pp. 1041–1056, 2020.

[30] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y. Arcas, "Communication-Efficient Learning of Deep Networks from Decentralized Data," in *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, vol. 54, 2017, pp. 1273–1282.

[31] M. Tang and V. W. Wong, "Deep reinforcement learning for task offloading in mobile edge computing systems," *IEEE Transactions on Mobile Computing*, vol. 21, no. 6, pp. 1985–1997, 2022.

[32] K. Peng, P. Xiao, S. Wang, and V. C. Leung, "Scof: Security-aware computation offloading using federated reinforcement learning in industrial internet of things with edge computing," *IEEE Transactions on Services Computing*, vol. 17, no. 4, pp. 1780–1792, 2024.

[33] F. Metelo, C. Soares, S. Racković, and P. Á. Costa, "Peersimgym: An environment for solving the task offloading problem with reinforcement learning," in *Machine Learning and Knowledge Discovery in Databases. Applied Data Science Track*. Cham: Springer Nature Switzerland, 2024, pp. 38–54.

[34] T. Anttalainen, *Introduction to telecommunications network engineering*, 2nd ed., ser. Artech House telecommunications library, 2003.

[35] N. Kumari, A. Yadav, and P. K. Jana, "Task offloading in fog computing: A survey of algorithms and optimization techniques," *Computer Networks*, vol. 214, p. 109137, 2022.

[36] T. Hu, Z. Pu, X. Ai, T. Qiu, and J. Yi, "Measuring policy distance for multi-agent reinforcement learning," in *Proceedings of the 23rd International Conference on Autonomous Agents and Multiagent Systems*, ser. AAMAS '24, 2024, p. 834–842.

[37] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Proceedings of the 12th International Conference on Neural Information Processing Systems (NIPS)*, 1999, p. 1057–1063.

[38] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," *arXiv*, 2018. [Online]. Available: http://arxiv.org/abs/1506.02438

[39] T. Rausch, C. Lachner, P. A. Frangoudis, P. Raith, and S. Dustdar, "Synthesizing plausible infrastructure configurations for evaluating edge computing systems," in *3rd USENIX Workshop HotEdge 20*, 2020.

[40] H. Tian, Y. Zheng, and W. Wang, "Characterizing and synthesizing task dependencies of data-parallel jobs in alibaba cloud," in *Proc. ACM Symp. Cloud Comput.*, 2019.

[41] M. Andrychowicz, A. Raichuk, P. Stanczyk, M. Orsini, S. Girgin, R. Marinier, L. Hussenot, M. Geist, O. Pietquin, M. Michalski, S. Gelly, and O. Bachem, "What matters in on-policy reinforcement learning? A large-scale empirical study," *CoRR*, vol. abs/2006.05990, 2020. [Online]. Available: https://arxiv.org/abs/2006.05990

A Extending PeersimGym

The PeersimGym [33] environment for TO with multi-agent reinforcement learning was not originally designed for federated learning. To address this, we extended it with the FL Updates Manager (FLManager) to enable the exchange of FL updates across the simulated network. The FL process begins with the FL algorithm determining which updates to share. These updates are sent to the environment, where the FLManager generates an ID for each update, calculates its size, and stores the relevant information. This data is then transmitted to the simulation, which sends a dummy message with the size of the update from the node hosting the source agent to the node hosting the destination agent through the network. FL agents can then query the FLManager for completed updates, prompting it to retrieve any updates that have traversed the simulated network. To ensure compatibility with other environments, we decoupled the FLManager from PeersimGym and introduced a customizable mechanism for computing the number of steps an update takes to arrive. The code for the FLManager is available in the FAuNO repository.

B FAuNO nodes

Each FAuNO node consists of three key components: the orchestration agent or manager, the information exchange module, and the resource provisioning component, as our focus is on developing an algorithm for the decentralized orchestration of clients' computational requirements, we keep the other components generic for adaptability across various scenarios. As illustrated in Fig. 3, one of the participants assumes the role of FAuNO Global manager, managing the global model, this role can be assumed by any node in the network.

C Algorithms

We provide the pseudocode for the two phases of FAuNO learning. We also provide the actual code developed in our git repository for https://anonymous.4open.science/r/FAuNO-C976 (anonimized). The first component we mention is the Local algorithm, as seen in 1 ran by the participants in the Federation:

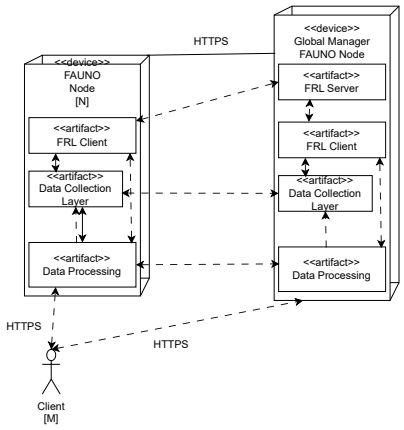


Figure 3: Deployment diagram representing the different components in the basic and global manager FAuNO Nodes, with the arrow representing that some messages are exchanged between the nodes.

Algorithm 1 FAuNOLocalPPO

Require: Initial critic weights w_0, learning rate local critic η_{critic}, learning rate local actor η_{actor}, initial actor weights θ_0, minibatch size M, number of steps between trainings N, number of steps before sharing weights with global T

$\theta_{\text{old}} \leftarrow \theta_0$
$w \leftarrow w_0$
steps $\leftarrow 0$
version $\leftarrow 0$
for iteration $= 1, 2, \ldots$ **do**
  w, version $\leftarrow$ `checkIfNewerGlobalArrived()` {Resets number of steps since last update}
  for step $= 1, 2, \ldots, N$ **do**
    Run policy $\pi_{\theta_{\text{old}}}$ in environment for timesteps
  end for
  Compute advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$ using $V(\cdot; w)$
  Optimize surrogate $L^{\text{CLIP+VF+S}}$ w.r.t. θ and w, with K epochs and minibatch size M
  $\theta_{\text{old}} \leftarrow \theta$
  steps $\leftarrow$ steps $+ 1$
  if iteration $\mod T = 0$ **then**
    `shareUpdatesWithGlobal`(∇w, steps, version) {Asynchronous operation}
  end if
end for

In this algorithm, the *checkIfNewerGlobalArrived()* function checks whether a newer version of the global critic model has been sent. If so, it returns the updated model; otherwise, it returns the current model, w, that the agent has trained. Similarly, the *shareUpdatesWithGlobal(u, steps, version)* function asynchronously shares the latest updates, u, with the global node.

We note that minimizing the negative of eq.(13), $-L_t^{\text{CLIP+VF+S}}$, is equivalent to maximizing the original objective.

Then we look at the global algorithm executed by one of the nodes in the federation in 2.

Algorithm 2 FAuNOGlobalManager

Require: Global critic learning rate η_{critic}, local actor learning rate η_{actor}, client training steps Q, buffer size K, all participating agents m, minibatch size M, number of steps between trainings N, number of steps before sharing weights with global T

Ensure: FL-trained global critic model w_g

$w_g \leftarrow w_0$
Initialize Buffer $\leftarrow \{\}$ {Start with an empty buffer}
$k \leftarrow 0$
while not converged **do**
  Run `FAuNOLocalPPO`(w_0, η_{critic}, η_{actor}, θ_0, M, N, T) on m {Asynchronous operation}
  if client update received and used latest k **then**
    Receive Δ_i, steps$_i$, version$_i$ from client i
    if $\Delta_i \notin$ Buffer **then**
      Add Δ_i, steps$_i$, version$_i$ to Buffer
      $k \leftarrow k + 1$
    else if steps$_i$ > steps stored in Buffer **then**
      Replace Δ_i in Buffer with the newer one
    end if
    if $k \geq K$ **then**
      $w_g \leftarrow w_g + \sum_{k \in \text{Buffer}} \text{computeCoefficient}(\text{Buffer}, k)\Delta_k$
      Clear Buffer
      $k \leftarrow 0$
      `sendLatestModelToClients()` {Asynchronous operation}
    end if
  end if
end while

In this algorithm, *computeCoefficient()* calculates the weight of each update based on the number of updates each agent sent, and *sendLatestModelToClients()* is a method that sends the latest global critic network to all the clients.

D Implementation Details

D.1 Hyperparameters used for FAuNO

We based our choice of hyperparameters on [41]. The parameters used for FAuNO:

Table 6: Hyperparameters Used in FAuNO Experiments

Parameter	Value	Explanation
γ	0.90	Discount factor for the long-term reward computation
ϵ	0.5	PPO clipping parameter
η	0.00001	Learning rate for the global model (affects critic)
μ	0.005	Scales the the proximal term in PPO
Actor Learning rate	0.001	Learning rate for the actor network
Critic Learning rate	0.0003	Learning rate for the critic network
Critic Loss coefficient	0.5	Coefficient for the critic loss term
Entropy Loss coefficient	0.5	Coefficient for the entropy loss term
Save interval	1500 steps	Frequency at which models are saved
Steps per exchange	150 steps	Number of steps before exchanging data
Steps per episode	150 steps	Number of steps per training episode
Batch size	30	Size of batches for gradient updates

D.2 Network Architecture

The architectures for the PPO are based on the ones in https://github.com/nikhilbarhate99/PPO-PyTorch

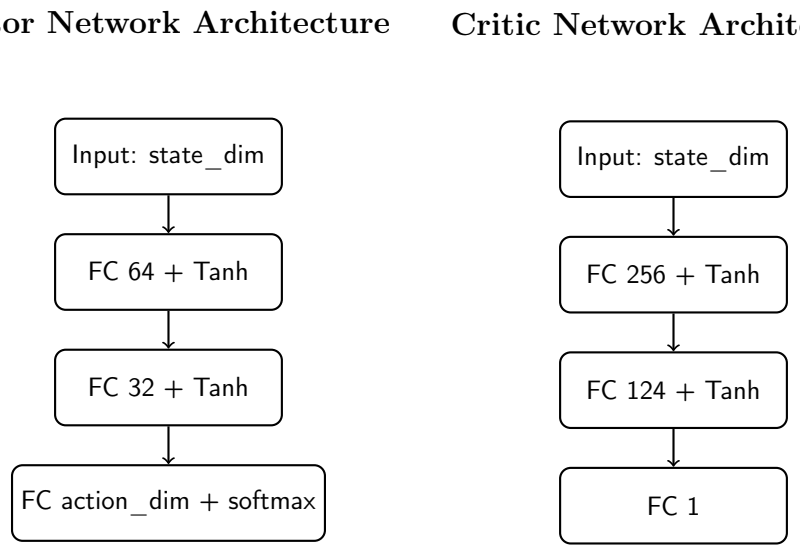
E Test Setup

Here we give the concrete simulation setup configurations and elaborate on the baseline algorithms used. More details are available in the repository FAuNO repository[1]

E.1 Baselines

To compare FAuNO, we implement a set of baseline policies. We classify these baseline policies into two different categories, the first is the heuristic baselines **Least Queue** which selects the observable worker with the smallest queue size relative to its maximum queue size and offloads the next eligible task to that worker. The purpose of the heuristic baseline is to provide a reference point that is widely understood and accessible serving as a benchmark for expected

[1] https://anonymous.4open.science/r/FAuNO-C976/README.md



(a) Actor Network (b) Critic Network

Figure 4: Actor-Critic Neural Network Representations

performance, offering a familiar comparison point that helps contextualize the results. We then consider the State-of-the-art synchronous FRL algorithm, SCOF [32] in the spirit of looking at the benefits of considering the improvements of an asynchronous training mechanism that keeps training even considering heterogeneous devices and communication delays. SCOF is an algorithm designed for TO in the IIoT setting with a focus on vertical offloading from Edge devices to a set of Edge Nodes from the SBCs, not considering the offloading mechanics of the Edge Nodes themselves. The algorithm itself considers a Federated Duelling DQN, that is aggregated with a FedAvg-based approach and utilizes differential privacy (DP) to improve the security of the update exchanges. We could not find any implementation of SCOF, so we provided our implementation of the algorithm based on SCOF's paper [32] in FAuNO's repository. We introduce some modifications to make SCOF usable in our setting. Namely, we do not use DP in our updates so we also do not use it in SCOF's, furthermore we adapt SCOF to use our MG definition and Edge Setting formulation.

E.2 Ether Experiment Parameters

The experiments are based on two distinct network topologies generated using Ether, with 2 and 4 AoT clusters. Each simulated AoT cluster consists primarily of SBC, modeled as Raspberry Pi 3s, along with a base station equipped with an Intel NUC and two GPU units, and a remote, more powerful server. The topologies vary in cluster numbers, ranging from one to four clusters, and correspondingly in node counts, from 12 to 31. The number of agents making

task-offloading decisions scales with the number of nodes, with all SBCs, NUCs, and the remote server hosting an agent. This results in 10 to 23 agents across different topologies. We configure the simulation so that only the nodes at the edge of the network, the SBCs, will directly receive tasks. The specific number of each node type is available in tab. 7, and the number of nodes taking up a given function is available in tab. 8.

No. Clusters	**SBCs**	**NUCs**	**GPU units**	**Servers**
2	16	2	4	1
4	32	4	8	1

Table 7: Cluster Composition Table

No. Clusters	**No Agents**	**Nodes getting tasks from clients**	**Total nodes**
2	19	16	23
4	37	32	40

Table 8: Cluster Configuration Table

The visualization produced for each of the scenarios can be observed in fig.5a

Regarding the capabilities of the different components involved in the simulation, we relied on the hardware specifications generated by the Ether tool. We supplemented this information with data we found for each machine. This information is available in tab. 9.

Task generation at each SBC node follows a $Poisson(\lambda)$ distribution over a simulation episode of 1000 time steps, with each time step scaled by a factor of 10, making each tick equivalent to 1/10th of a second, for a total of 10,000 ticks per episode. Each agent makes an offloading decision at every time step, performing 30 episodes, with the ability to take action at each tick. A full list of the parameters used in our simulation can be found in tab. 10. We note that all time-dependent functions are scaled as well.

Table 9: Device Capacities

Device	**CPU (Millis)**	**Memory (Bytes)**
Raspberry Pi 4	7200	6442450944
Raspberry Pi 5 6GB	9600	6442450944
Raspberry Pi 6 8GB	9600	8589934592
Intel NUC	14800	68719476736
Cloudlet	290400	188000000000

E.3 Artificial Network Experiment Parameters

We consider two topologies with 10 and 15 nodes randomly distributed across a 100x100 square. These topologies have an increasing number of SBCs and a fixed

Table 10: Parameter values in the experimental setup.

Simulation time, T	1000 s		
Task input size, α_i^{in}	150 Mbytes	**Task output size,** α_i^{out}	150 Mbytes
Task instructions, ρ_i	$8{\times}10^7$	**Task utility,** r_u	100
CPI, ξ_i	1	**Weight waiting,** χ_D^{wait}	1
Deadline, δ_i	100	**Weight execution,** χ_D^{exc}	0.5
Bandwidth, $B_{i,j}$	4 MHz	**Weight comm,** χ_D^{comm}	3
Transmission power, P_i	40 dbm	**Weight overload,** χ_O	30
Scale	10		

number of NUCs. All SBCs receive tasks and all the nodes in the topology have agents controlling them. The SBCs are picked randomly in equal proportions from the options in tab.9. The concrete number of nodes for each is given in tab. 11: We consider the same hyperparameters explained in 6. And consider

No. Nodes	SBCs	NUCs
10	5	5
15	10	5

Table 11: Cluster Composition Table

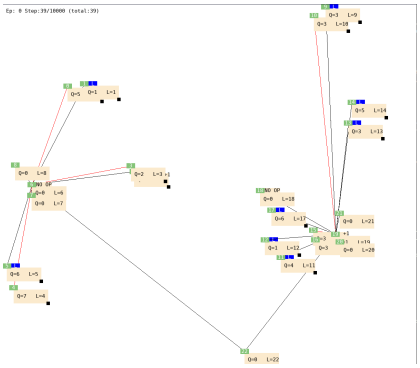
similar training conditions to the ether-based topologies.

E.4 Alibaba Cluster Trace based Workload

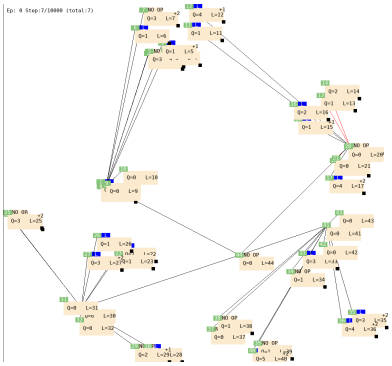
The workload considered for the experiments in the paper was based on the integration of PeersimGym [33] with an Alibaba Cluster trace based workload generation tool [40]. However, the original task sizes were unsuitable for the edge environment under study, particularly for client nodes, which became overwhelmed and dropped nearly 90% of tasks. To address this, we implemented a rescaling mechanism that adjusted the number of instructions per task while keeping all other characteristics the same. After evaluating several scaling factors, we selected a 10

E.5 Computational Requirements

The tests were all executed in a private High-Performance Computer, orchestrated by slurm and utilized 16gb of ram memory, 4 cores and a MiG partition with 10gb of memory of an Nvidia A100 GPU. Each of the tests that utilized a GPU took about 10 to 18 hours, depending on the number of agents, to complete the 400 000 steps.



(a) Topology with 2 AoT clusters, consisting of 12 SBC, 2 NUCs, 4 GPU units, and 1 server. The total number of nodes is 19, and 15 agents manage task-offloading decisions, as outlined in tab. 7.



(b) Topology with 4 AoT clusters, consisting of 18 SBC, 4 NUCs, 8 GPU units, and 1 server. The total number of nodes is 31, with 23 agents managing task-offloading decisions, as shown in tab. 7.

Figure 5: Visualization of the different simulations used.