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

Anonymous authors
Paper under double-blind review

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]

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 Min et al. (2019). Edge Computing (EC) extends the cloud by bringing computational resources closer to end-users, addressing latency and traffic issues Varghese & Buyya (2018). 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 Yu et al. (2020). The EC paradigm 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 Baek & Kaddoum (2023). 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 Zhu et al. (2019). Traditional optimization methods often struggle to efficiently manage these complex systems, due to the dynamic, time-varying, and complex environments of Edge Systems Xu et al. (2018). Reinforcement Learning (RL) Baek & Kaddoum (2023); Zhu et al. (2019), 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 Baek & Kaddoum (2023); Gao et al. (2022). The ability of MARL agents to iteratively learn optimal strategies through simultaneous interaction with an environment makes them particularly suited for decentralized edge systems Lin et al. (2023); Zhang et al. (2023). Due to the nature of MARL, it is common to have some form of message exchange Zhang et al. (2018); Baek & Kaddoum (2023) 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 Consul et al. (2024). 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) Qi

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

et al. (2021). 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 mitigates 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. This enables agents to benefit from each other's knowledge while minimizing communication overhead. However, conventional FL suffers when stragglers delay aggregation or drop updates, reducing training efficiency and wasting samples. This can be addressed by adopting a buffered semi-asynchronous strategy, in which faster nodes continue contributing updates without waiting, while slower nodes are still able to align with the evolving global critic. FAuNO adopts a buffered semi-asynchronous strategy, where faster nodes continue contributing updates without waiting, while slower nodes are still able to align with the evolving global critic. In this way, we extend Federated Buffering Nguyen et al. (2021) to reinforcement learning, enabling continuous local training without being bottlenecked by stragglers. We summarize the motivations and principal contributions of this work below.

Motivations & Contributions

- We address the **TO problem in edge systems** by framing it within a **Partially Observable Markov Game (POMG)**, enabling decentralized decision-making under partial observability.
- We introduce **FAuNO**, the first framework to integrate **buffered semi-asynchronous aggregation** with **actor–critic MARL (PPO)** in a federated setting for edge offloading. Our adaptation of FedBuff to reinforcement learning enables faster agents to contribute multiple updates without waiting for stragglers, improving sample efficiency under heterogeneous conditions. By federating only the **critic** while keeping **actors local**, FAuNO mitigates heterogeneity, respects partial observability, and supports fully decentralized execution. Through empirical evaluation, we show that FAuNO **outperforms or matches FRL and heuristic baselines** in terms of task completion time and task completion.
- We **extend the PeersimGym** environment to support **federated update exchanges** over the simulated network (details in annex 7). This extension simulates the communication of the updates affecting how and when updates are propagated and aggregated. As a result, the evaluation reflects the conditions of realistic edge systems.

Background & Related Work

TO involves transferring computations from constrained devices to more capable ones, addressing the *what*, *where*, *how*, and *when* of offloading Fahimullah et al. (2022). TO methods include vertical offloading to higher-tier systems Qiu et al. (2019), horizontal offloading among peers Baek et al. (2019), and hybrid approaches Baek & Kaddoum (2023). Offloading target selection may prioritize proximity Van Le & Tham (2018); Yu et al. (2020) or queue length Baek et al. (2019), or consider unrestricted selection, accounting for consequences of offload failures. Failures are affected by factors like latency Dai et al. (2022), resource capacity Van Le & Tham (2018), energy shortages, or others Peng & et al. (2022). This study focuses on *Binary TO* Hamdi et al. (2022) 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 case, TO is commonly modeled as an MDP and solved with Q-learning in Fog networks Baek et al. (2019), DQN in ad-hoc mobile clouds Van Le & Tham (2018), DDPG for task dependencies Liu et al. (2023), SARSA variants for real-time MEC Alfakih et al. (2020), and DQN extensions for delay-sensitive tasks Liu et al. (2022). Bandit formulations have also been used to simplify binary offloading while optimizing latency and energy Zhu et al. (2019). In the multi-agent case, MARL methods address resource allocation and collaboration in heterogeneous, partially observable environments. For example, in Baek & Kaddoum (2020) TO in Multi-Fog systems is modeled as a Stochastic Game, and a Deep Recurrent Q-Network (DRQN) with Gated Recurrent Units is employed to handle partial state observations.

FRL has been explored 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 Li et al. (2023), 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 using

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) Baek & Kaddoum (2020)	✓		DRQN	✓			
Baek et al. (2022) Baek & Kaddoum (2023)	✓	✓	✓	✓	✓		
Zang et al. (2022) Zang et al. (2022)	✓	✓	DQN	✓	✓		
Li et al. (2023) Li et al. (2023)	✓	✓	DDQN				
Peng et al. (2024) Peng et al. (2024)		✓	Dueling DQN	✓			
FAuNO (ours)	✓	✓	✓	✓	✓	✓	✓

a weighted federated averaging algorithm. A unary outlier detection technique is used to manage stragglers.

In Consul et al. (2024), a hierarchical FRL model is proposed for frame aggregation and offloading of Internet of Medical Things data, optimizing energy and latency by aggregating learned parameters from body-area devices to edge and central servers. In Chen & Liu (2022) 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 McMahan et al. (2017) based algorithm for the global solution. In Tang & Wong (2022), 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 Zang et al. (2022), 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 Baek & Kaddoum (2023), 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, Peng et al. (2024) 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 Metelo et al. (2024). The comparison with the related work is summarized in Table 1.

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 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 $Q_{\max}^n$. Whenever the capacity is reached, no new tasks are accepted until there is available space. 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 can be both a worker and a client.

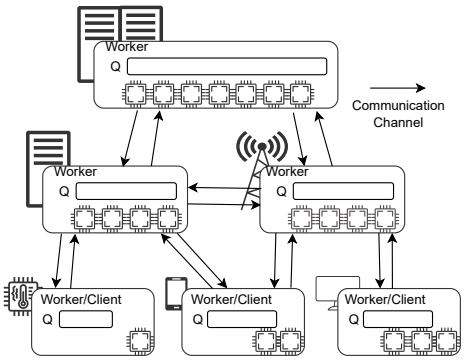


Figure 1: Edge System Architecture of our system model. The workers are capable of independently offloading tasks, exchanging information, and FL model updates through the communication channels.

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 is 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 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 Anttalainen (2003). According to this theorem, the 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. See annex 10.1 for details on communication protocols.

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 Baek et al. (2019); Kumari et al. (2022). The delay function incorporates hyperparameters χ_D^{wait}, χ_D^{comm}, and $\chi_D^{\text{exc}} \geq 0$. The delay terms for a given action a_t^n are:

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

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, Q_n^t is the queue size of the same at time t, 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 $T_{i,a_t}^{\text{comm}}(\alpha_k^{\text{out}})$ denotes the communication cost of TO, defined as a delay (eq. 1), where a_t^n indicates the neighboring node i. If the task is executed locally, this term becomes zero. The term:

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

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:

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

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

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

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 eq. 7, and no tasks should breach their deadlines. Additionally, no node should exceed its computational resource limit, as outlined in eq. 8.

Partially-Observable Markov Game. To solve the TO problem with distributed and decentralized agents, we define it as a Partially-Observable Markov Game (POMG) Hu et al. (2024), 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 its local computational and communication resources, the information shared by its neighbors on the same, and information about the next offloadable task. The details on the observation space are provided in 10.2. $\mathcal{A} = \{A_i\}_{i \in \mathcal{N}}$ 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 $\chi_O O(s_t, a_t^i)$ is the distance to overload the workers involved in an offloading, we define $O(s_t, a_t) = -\log(p_t^{Oa_t})/3$. 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 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 workers. A detailed breakdown of FAuNO node components is provided in annex 8.

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 (PPO) Schulman et al. (2017). 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 for training the critic network builds on FedBuff Nguyen et al. (2021), a buffered asynchronous aggregation method that we adapt to RL by allowing agents to keep training and sending updates to the global critic without stopping after the first round. This prevents stragglers from blocking progress while still incorporating shared updates into the global critic. The crux of the proposed algorithm is that by federating the global network, we mitigate selfish behavior among agents and improve sample efficiency through continuous, non-blocking training.

Local Policy Optimization Our local optimization uses an adaptation of PPO to FL, combined with Generalized Advantage Estimation (GAE) Schulman et al. (2018) for computing advantages. In our version, agents independently interact with the environment and, after a configurable number of training steps, share their latest critic network with the global manager. Upon receiving an updated global model, each agent incorporates it as the next critic for training. The local optimization procedure is summarized in algo. 1, with full details provided in annex 11.1.

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 non-blocking semi-asynchronous method to tackle the following optimization problem:

$$\min_w f(w), \text{ s.t. } f(w) := \frac{1}{m}\sum_{k=1}^{m} p_k l_k(w, \theta_k). \tag{10}$$

Here, m represents the number of participants, and θ_k are 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 the symmetric of eq. 23.

FAuNO's semi-asynchronous design addresses heterogeneity and stragglers. Faster agents contribute updates more frequently, while slower ones do not block progress. The gradients are buffered at the global manager (GM), with newer updates from the same agent replacing older ones and increasing the weight of that agent's last update in the aggregation. The weights are computed following algo. 3. The global critic is updated once updates from K distinct agents are received. This allows for agents to continue training without waiting for the global training round to complete, allowing for continuous training even under straggling devices. To mitigate policy divergence and allow for specialization on each node as well, we federate only the critic network, while keeping the actors local. Furthermore, each agent's observation space is standardized and includes its own queue size, neighbors' queue and capacity states, aggregate task instruction counts, and features of the next task to be processed, more details in annex 10.2. Upon aggregation, the GM updates the global critic via a weighted average:

$$\hat{w} = w + \sum_{k \in \bar{K}} p_k \nabla w_k, \tag{11}$$

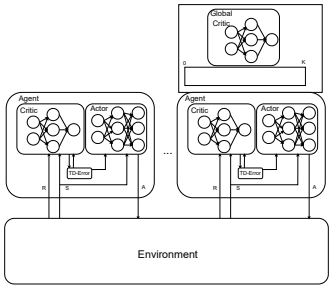
where $\bar{K} \geq K$ is the set of buffered updates and $\sum_{k \in \bar{K}} p_k = 1$. Fig. 2 illustrates the global training flow. Upon aggregation, the GM updates the global critic via a weighted average:

$$\hat{w} = w + \sum_{k \in \bar{K}} p_k \nabla w_k, \tag{12}$$

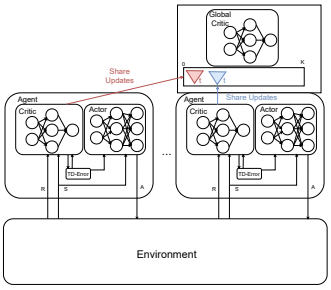
where $\bar{K} \geq K$ is the set of buffered updates. The coefficient p_k is calculated based on the number of update steps each agent performed, ensuring that $\sum_{k \in \bar{K}} p_k = 1$. Fig. 2 illustrates the global training flow. The global algorithm can be observed in algo. 2 in annex 9.

4 Performance Evaluation

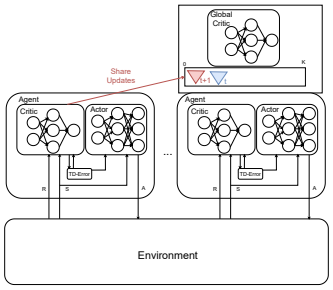
In this section, we evaluate FAuNO's performance using two standard TO metrics: average task completion time and percentage of completed tasks—the proportion of tasks that were created and whose results were successfully returned to the originating client. We compare FAuNO against



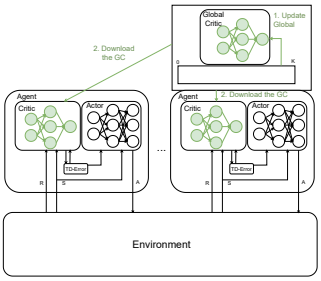
(a) Agents are training their local models.



(b) Updates of agents arrive at the GM 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.

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 Peng et al. (2024). Since no public implementation of SCOF was available, we reimplemented the algorithm and released the code in FAuNO's repository. For fairness, we adapted SCOF to our observation and reward spaces and disabled its DP component, as privacy was not the focus of this work, and DP typically reduces accuracy. These adaptations ensure comparability without diminishing SCOF's core capabilities. Details on the baselines are provided in annex 11.2. We benchmark our solution using PeersimGym Metelo et al. (2024), with realistic topologies generated by the Ether tool Rausch et al. (2020). 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 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 Tian et al. (2019), which we have rescaled to better 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. Finally, we present an ablation study on the impact of agent heterogeneity on convergence and the impact of the K parameter on the performance of the algorithm; due to space constraints, the ablation on the K parameter is in annex 6. Additional details on the testing setup, topologies, workloads, and hyperparameters used are provided in annexes 11 and 12.

4.1 Ether based topologies

Tab. 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 in response time. Although SCOF achieves lower response times, it does so at the cost of significantly reduced

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

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.

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

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

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 outperforms FAuNO in task completion. This can be explained by the larger accessibility to more powerful nodes in the random topology, leading to more offloading and concurrent tasks being processed. A deeper analysis of the impact of topology is available in 6.3. However, LQ's 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 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. Moreover, the higher task completion rate of LQ in random topologies stems from dense connectivity, which favors aggressive offloading. Our reward shaping, tailored to mitigate congestion in star-like Ether topologies, biases FAuNO toward local processing—explaining the observed trade-off in throughput versus delay.

Table 6: Global disagreement score (↓ better)

Variant ↓ / Packet-drop D →	0.3	0.5	0.8
FAuNO vs FAuNO	74.5	232.7	289.8
FAuNO vs Oracle critic	63.6	240.1	307.2
Pure MARL $(D = 1)$	147.54	262.22	313.81

Table 7: Disagreement scores. MARL (packet-drop rate 1.0) vs. centralized; FAuNO not shown

Variant	Global disagreement δ
Pure MARL vs Pure MARL	319.35
Fully centralized oracle vs MARL	325.55

4.3 Learning under Heterogenity

To evaluate FAuNO's stability under asynchronous, non-IID conditions, we designed a heterogeneous workload experiment using the 15-node random topology. The network was partitioned into three regions, each configured to process a specific workload class with different task sizes and arrival rates. Clients in each region generated tasks only from their corresponding class distribution; the details on the experiment and result analysis are available in the annex 6.1. We compared three training setups: FAuNO, pure MARL PPO, where agents learn without any shared critic, and a centralized oracle where all nodes share a single critic model. To assess consistency between the different critics, we introduced a critic-agreement protocol and measured critic consistency using a global disagreement score (eq. 14), δ, based on pairwise RMSE across sampled states (eq. 13). During each evaluation episode, we sampled 500 global states and collected observations from each agent. To correct for the fact that agents processing faster task streams naturally accumulate higher rewards, all values were normalized by the corresponding task arrival rate before computing disagreement. Results are summarized in Tab. 6 and 7. As expected, disagreement between critics decreases when the packet-drop rate is reduced, indicating more consistent models as communication becomes more reliable. FAuNO's critics approach the predictions of the centralized oracle at low drop rates, confirming that aggregation yields stable shared learning. By contrast, the pure MARL variant showed substantially higher disagreement, highlighting its divergence under heterogeneous workloads. These results confirm that FAuNO is robust to non-IID conditions and mitigates policy inconsistency even when agents face systematically different task distributions.

5 Conclusion, Limitations & Future Work

We addressed the decentralized TO problem in edge systems by modeling it as a cooperative objective over a federation of agents, formalized within a POMG. To this end, we proposed **FAuNO**, a novel 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 superiority over heuristic and FRL baselines in terms of task loss and latency, highlighting its adaptability to dynamic and heterogeneous edge settings.

Limitations. From a security perspective, the current formulation assumes that all nodes are honest and cooperative. Adversarial and Byzantine behavior, although likely to occur in real-world edge environments, is not considered, and privacy preservation is also outside the present scope. At the system level, we model a stable network with reliable nodes and communication links, excluding failures, congestion, and bandwidth constraints. Furthermore, we do not consider energy costs that could trade off with latency. These assumptions simplify the evaluation but omit factors critical to practical edge deployments. Algorithmically, the leveraging of FedBuff introduces a bias toward faster clients, potentially underrepresenting slower nodes. Moreover, reliance on a GM creates a single point of failure and a potential bottleneck in very large networks.

Future Work. Future work includes supporting dynamic topologies and node mobility, handling intermittent connectivity, and addressing adversarial participation. At the system level, we plan to extend our objective to consider data locality, fault tolerance, and energy-consumption. Algorithmically, we will address the single-manager bottleneck by exploring hierarchical or decentralized critics to improve scalability and robustness. We also intend to incorporate energy-aware objectives to capture trade-offs between latency and resource use, and to design defenses against malicious agents to enhance security.

Reproducibility Statement

We provide an anonymous repository at https://anonymous.4open.science/r/FAuNO-C976, which contains the complete codebase developed for this paper. This includes implementations of all proposed methods, test configurations, hyperparameters, and supporting scripts required to re-run the experiments and reproduce the reported results. Furthermore, the detailed implementation choices, hyperparameters, and training configurations are also partially documented in annex 11. The annex further includes descriptions of the experimental setup and evaluation protocol. Together, these materials are intended to enable full reproducibility of our results.

References

Taha Alfakih, Mohammad Mehedi Hassan, Abdu Gumaei, Claudio Savaglio, and Giancarlo Fortino. Task offloading and resource allocation for mobile edge computing by deep reinforcement learning based on sarsa. *IEEE Access*, 8:54074–54084, 2020.

Marcin Andrychowicz, Anton Raichuk, Piotr Stanczyk, Manu Orsini, Sertan Girgin, Raphaël Marinier, Léonard Hussenot, Matthieu Geist, Olivier Pietquin, Marcin Michalski, Sylvain Gelly, and Olivier Bachem. What matters in on-policy reinforcement learning? A large-scale empirical study. *CoRR*, abs/2006.05990, 2020. URL `https://arxiv.org/abs/2006.05990`.

Tarmo Anttalainen. *Introduction to telecommunications network engineering*. Artech House telecommunications library. 2nd edition, 2003. ISBN 9781580535007.

Jung-yeon Baek, Georges Kaddoum, Sahil Garg, Kuljeet Kaur, and Vivianne Gravel. Managing fog networks using reinforcement learning based load balancing algorithm. In *2019 IEEE Wireless Communications and Networking Conference (WCNC)*, pp. 1–7, 2019. doi: 10.1109/WCNC.2019.8885745.

Jungyeon Baek and Georges Kaddoum. Heterogeneous task offloading and resource allocations via deep recurrent reinforcement learning in partial observable multifog networks. *IEEE Internet of Things Journal*, 8(2):1041–1056, 2020.

Jungyeon Baek and Georges Kaddoum. Floadnet: Load balancing in fog networks with cooperative multiagent using actor-critic method. *IEEE Trans. Netw. Serv. Manage.*, 2023.

Nikhil Barhate. nikhilbarhate99/ppo-pytorch, July 2024. URL `https://github.com/nikhilbarhate99/PPO-PyTorch`.

X. Chen and G. Liu. Federated deep reinforcement learning-based task offloading and resource allocation for smart cities in a mobile edge network. *Sensors*, 22(13):4738, 2022. doi: 10.3390/s22134738.

Prakhar Consul, Ishan Budhiraja, Ruchika Arora, Sahil Garg, Bong Jun Choi, and M. Shamim Hossain. Federated reinforcement learning based task offloading approach for mec-assisted wban-enabled iomt. *Alexandria Engineering Journal*, 86:56–66, 2024. ISSN 1110-0168. doi: https://doi.org/10.1016/j.aej.2023.11.041.

Fei Dai, Guozhi Liu, Qi Mo, WeiHeng Xu, and Bi Huang. Task offloading for vehicular edge computing with edge-cloud cooperation. *World Wide Web*, 25(5):1999–2017, 2022.

Muhammad Fahimullah, Shohreh Ahvar, and Maria Trocan. A review of resource management in fog computing: Machine learning perspective. 2022. doi: 10.48550/arXiv.2209.03066. arXiv:2209.03066 [cs].

Zhen Gao, Lei Yang, and Yu Dai. Large-scale computation offloading using a multi-agent reinforcement learning in heterogeneous multi-access edge computing. *IEEE Transactions on Mobile Computing*, 2022.

Aisha Muhammad A. Hamdi, Farookh Khadeer Hussain, and Omar K. Hussain. Task offloading in vehicular fog computing: State-of-the-art and open issues. *Future Generation Computer Systems*, 133:201–212, 2022. ISSN 0167-739X. doi: 10.1016/j.future.2022.03.019.

Tianyi Hu, Zhiqiang Pu, Xiaolin Ai, Tenghai Qiu, and Jianqiang Yi. Measuring policy distance for multi-agent reinforcement learning. In *Proceedings of the 23rd International Conference on Autonomous Agents and Multiagent Systems*, AAMAS '24, pp. 834–842, 2024.

Nidhi Kumari, Anirudh Yadav, and Prasanta K. Jana. Task offloading in fog computing: A survey of algorithms and optimization techniques. *Computer Networks*, 214:109137, 2022. ISSN 1389-1286.

Jie Li, Zhiping Yang, Xingwei Wang, Yichao Xia, and Shijian Ni. Task offloading mechanism based on federated reinforcement learning in mobile edge computing. *Digital Communications and Networks*, 9(2):492–504, 2023. doi: 10.1016/j.dcan.2022.04.006.

Lixia Lin, Wen Zhou, Zhicheng Yang, and Jianlong Liu. Deep reinforcement learning-based task scheduling and resource allocation for noma-mec in industrial internet of things. *Peer-to-Peer Networking and Applications*, 16(1):170–188, 2023.

Shumei Liu, Yao Yu, Xiao Lian, Yuze Feng, Changyang She, Phee Lep Yeoh, Lei Guo, Branka Vucetic, and Yonghui Li. Dependent task scheduling and offloading for minimizing deadline violation ratio in mobile edge computing networks. *IEEE Journal on Selected Areas in Communications*, 41(2): 538–554, 2023. doi: 10.1109/JSAC.2022.3233532.

Xiaowei Liu, Shuwen Jiang, and Yi Wu. A novel deep reinforcement learning approach for task offloading in mec systems. *Applied Sciences*, 12(21), 2022. ISSN 2076-3417. doi: 10.3390/app122111260.

Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54 of *Proceedings of Machine Learning Research*, pp. 1273–1282, 2017.

Frederico Metelo, Cláudia Soares, Stevo Racković, and Pedro Ákos 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*, pp. 38–54, Cham, 2024. Springer Nature Switzerland. ISBN 978-3-031-70378-2.

Minghui Min, Liang Xiao, Ye Chen, Peng Cheng, Di Wu, and Weihua Zhuang. Learning-based computation offloading for iot devices with energy harvesting. *IEEE Transactions on Vehicular Technology*, 68(2):1930–1941, 2019.

John Nguyen, Kshitiz Malik, Hongyuan Zhan, Ashkan Yousefpour, Michael G. Rabbat, Mani Malek, and Dzmitry Huba. Federated learning with buffered asynchronous aggregation. *CoRR*, abs/2106.06639, 2021.

Kai Peng, Peiyun Xiao, Shangguang Wang, and Victor C.M. Leung. Scof: Security-aware computation offloading using federated reinforcement learning in industrial internet of things with edge computing. *IEEE Transactions on Services Computing*, 17(4):1780–1792, 2024. doi: 10.1109/TSC.2024.3377899.

Xin Peng and et al. Deep reinforcement learning for shared offloading strategy in vehicle edge computing. *IEEE Systems Journal*, 2022.

Jiaju Qi, Qihao Zhou, Lei Lei, and Kan Zheng. Federated reinforcement learning: Techniques, applications, and open challenges. *Intelligence & Robotics*, 2021. doi: 10.20517/ir.2021.02. arXiv:2108.11887 [cs].

Xiaoyu Qiu, Luobin Liu, Wuhui Chen, Zicong Hong, and Zibin Zheng. Online deep reinforcement learning for computation offloading in blockchain-empowered mobile edge computing. *IEEE Transactions on Vehicular Technology*, 68(8):8050–8062, 2019.

Thomas Rausch, Clemens Lachner, Pantelis A Frangoudis, Philipp Raith, and Schahram Dustdar. Synthesizing plausible infrastructure configurations for evaluating edge computing systems. In *3rd USENIX Workshop HotEdge 20*, 2020.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. 2017. doi: 10.48550/arXiv.1707.06347. arXiv:1707.06347 [cs].

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv*, 2018. URL http://arxiv.org/abs/1506.02438.

Richard S. Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Proceedings of the 12th International Conference on Neural Information Processing Systems (NIPS)*, pp. 1057–1063, 1999.

Ming Tang and Vincent W.S. Wong. Deep reinforcement learning for task offloading in mobile edge computing systems. *IEEE Transactions on Mobile Computing*, 21(6):1985–1997, 2022. doi: 10.1109/TMC.2020.3036871.

Huangshi Tian, Yunchuan Zheng, and Wei Wang. Characterizing and synthesizing task dependencies of data-parallel jobs in alibaba cloud. In *Proc. ACM Symp. Cloud Comput.*, 2019.

Duc Van Le and Chen-Khong Tham. A deep reinforcement learning based offloading scheme in ad-hoc mobile clouds. In *IEEE INFOCOM WKSHPS*, pp. 760–765, 2018.

Blesson Varghese and Rajkumar Buyya. Next generation cloud computing: New trends and research directions. *Future Generation Computer Systems*, 79:849–861, 2018. ISSN 0167-739X.

Zhiyuan Xu, Jian Tang, Jingsong Meng, Weiyi Zhang, Yanzhi Wang, Chi Harold Liu, and Dejun Yang. Experience-driven networking: A deep reinforcement learning based approach. In *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, pp. 1871–1879, 2018. doi: 10.1109/INFOCOM.2018.8485853.

Shuai Yu, Xu Chen, Zhi Zhou, Xiaowen Gong, and Di 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*, 8(4):2238–2251, 2020.

Lianqi Zang, Xin Zhang, and Boren Guo. Federated deep reinforcement learning for online task offloading and resource allocation in wpc-mec networks. *IEEE Access*, 10:9856–9867, 2022. doi: 10.1109/ACCESS.2022.3144415.

Fan Zhang, Guangjie Han, Li Liu, Yu Zhang, Yan Peng, and Chao Li. Cooperative partial task offloading and resource allocation for iiot based on decentralized multi-agent deep reinforcement learning. *IEEE Internet of Things Journal*, 2023.

Kaiqing Zhang, Zhuoran Yang, Han Liu, Tong Zhang, and Tamer Basar. Fully decentralized multi-agent reinforcement learning with networked agents. *CoRR*, abs/1802.08757, 2018.

Zhaowei Zhu, Ting Liu, Yang Yang, and Xiliang Luo. Blot: Bandit learning-based offloading of tasks in fog-enabled networks. *IEEE Trans. Parallel Distrib. Syst.*, 2019.

6 ABLATIONS

6.1 HETEROGENEOUS SETTING EXPERIMENT

We designed a heterogeneous workload experiment to evaluate whether FAuNO's federated critic converges stably under asynchronous, non-IID conditions, the precise setting where policy inconsistency would arise. We set specific regions of the network to handle different task types and have different cadences of arriving applications, λ.

We focus on the 15-node random topology detailed in 12, partitioned into 3 regions. Each of these regions is designed to process a specific workload class with different task types varying in number of instructions, ρ in MBytes, data size, α^{in} and task arrival rate λ. We define each of the workload classes as:

- **W1** (λ=0.5; ρ=38,α^{in}=32e7) this class is composed of nuc:2, rpi5 8G:2, rpi5 6G:1
- **W2** (λ=1; ρ=16; α^{in}=64e7) this class is composed of nuc:2, rpi5 8G:2, rpi4:1
- **W3** (λ=2.0; ρ=64; α^{in}=16e7) this class is composed of nuc:1, rpi5 6G:2, rpi4:2

A client attached to node i draws tasks only from that node's class distribution.

Training variants to compare

- FAuNO (federated critic)
- Pure MARL PPO, no shared critic network considered (Obtained by always dropping updates, so the agents never train anything in a centralized manner)
- Centralized oracle, where a single critic network is shared among all the participants.

Critic-agreement protocol

1. **Evaluation set:** For M states, $\{s_m\}$, collect the observations,$o_i(s_m)$, of each agent of that given state. During an evaluation episode, we sample around $M = 500$ distinct global states.
2. **Value Prediction matrix:** We then build matrix $V \in \mathbb{R}^{N\times M}$, where line i represents the evaluation of critic i and column m is the evaluation of point m. Thus, entry $V_{i,m}$ is defined as
$$V_{i,m} = V_i(o_i(s_m))$$
And, we also built the $V_\star$ matrix with the evaluation of the centralized critic model for the same observations.
3. **Pair-wise RMSE:** We then compute our metrics. A matrix where for each agent, we have the Root Mean Squared Error (RMSE) between the different state evaluations. This metric is meant to capture the differences on evaluating the observations by each of the agents. Each entry of this matrix is given by:
$$\text{RMSE}_{ij} = \sqrt{\tfrac{1}{M}\sum_m \big(V_{i,m} - V_{j,m}\big)^2}. \tag{13}$$
Where the $V_{i,m}$ is the evaluation of agent i, to provide a global metric of divergence of the matrix, we consider the global disagreement score computed as:
$$\delta = \tfrac{2}{N(N-1)}\sum_{i<j}\text{RMSE}_{ij}. \tag{14}$$
These same metrics are then re-computed for $V_\star$.

Comparison of the value function between nodes and against the V_* The group with a higher rate of task arrival λ will have better chances of increasing the received reward; thus, the estimate of the value will not be comparable. To have a fair measure of the quality between the different agent groups, we use the following normalization, for every node i:

$$\tilde{V}_{i,m} = V_{i,m}/\lambda_i$$

We study the global disagreement score, δ, across packet drop rates D. As expected, disagreement decreases as D is reduced (tab.6), reflecting more consistent agents. The federated critic increasingly aligns across the network as communication becomes more reliable. tab.6 shows that this alignment approaches the centralized oracle's predictions at low D, confirming that FAuNO benefits from improved communication and maintains stable shared learning even in highly heterogeneous settings. In contrast, tab.7 demonstrates that the MARL variant suffers from substantially higher disagreement under the same conditions. We must disclose that, although the MARL agent had begun converging, training was not completed but we expect the divergence to increase with training.

This analysis confirms that FAuNO's federated critic is robust to non-IID workloads and can mitigate policy inconsistency even when agents face systematically different distributions.

6.2 K Exploration

Table 8: Straggler and aggregation-threshold ablation results (mean ± s.d.)

Setting	Finished-task ratio	Avg. response time (ticks)
K = 0.3, s = 0.3	0.77 ± 0.02	258.85 ± 5.95
K = 0.3, s = 0.5	0.76 ± 0.02	259.74 ± 6.91
K = 0.3, s = 0.8	0.76 ± 0.02	260.07 ± 7.02
K = 0.5, s = 0.3	0.76 ± 0.02	259.09 ± 6.35
K = 0.5, s = 0.5	0.76 ± 0.02	258.83 ± 7.71
K = 0.5, s = 0.8	0.76 ± 0.02	260.86 ± 6.97

We evaluated FAuNO's robustness under straggler conditions and varying aggregation thresholds. Let s denote the fraction of gradients dropped before reaching the global node. In the straggler test, the objective is not to improve performance metrics but to avoid collapse. FAuNO achieves this goal: even when 80% of gradients are dropped ($s = 0.8$), both the finished-task ratio and average response time remain well within one standard deviation of their baseline values. This demonstrates that FAuNO gracefully degrades to local MARL when global connectivity is severely reduced, maintaining essentially the same performance. We also examined buffer sensitivity by doubling the aggregation threshold from $K = 0.3$ to 0.5. This change affects performance by less than 1% in either metric, indicating that FAuNO is robust to variations in buffer size under this workload.

6.3 Impact of topology on performance on synthetic networks

Table 9: Connection counts by topology and node type

Connection Type	**Random (10 nodes)**	**Random (15 nodes)**	**2 Clusters (23 nodes)**	**4 Clusters (45 nodes)**
nuc–nuc	2.4	2.4	0	0
nuc–rpi	3.2	8.4	8	8
rpi–nuc	3.2	4.2	1	1
rpi–rpi	3.2	7.6	0	0
srv–nuc	0	0	2	4
srv–rpi	0	0	16	32

The stronger task completion rate of LQ in random topologies reflects important topology-specific dynamics. As shown in Table 9, the random topologies considered in our experiments exhibit high connectivity between weaker RPIs and stronger NUCs (e.g., approximately 8.4 connections per RPI in the random topology with 15 nodes). This connectivity enables offloading with relatively low delay and cost, helping avoid congestion at any single NUC.

In contrast, our reward shaping was designed to discourage excessive offloading to mitigate latency and congestion in the star-like topologies generated by Ether. This global reward structure thus introduces a bias toward local processing. LQ, being reactive and unconstrained by this reward design, offloads more aggressively and achieves higher task throughput, albeit with higher delay. This illustrates a throughput–latency trade-off shaped jointly by topology and reward structure.

7 EXTENDING PEERSIMGYM

The PeersimGym Metelo et al. (2024) 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 (https://anonymous.4open.science/r/FAuNO-C976; anonymized).

8 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 GM, managing the global model; this role can be assumed by any node in the network. In our experiments, the data processing and collection layers are built into the simulation.

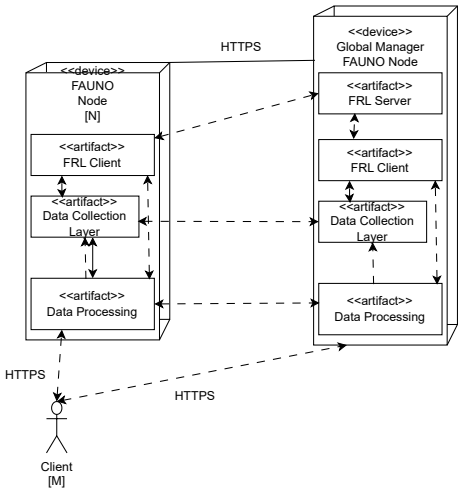


Figure 3: Deployment diagram representing the different components in the basic and GM FAuNO Nodes, with the arrow representing that some messages are exchanged between the nodes.

9 ALGORITHMS

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

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_t^F 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. 23, $-L_t^F$, is equivalent to maximizing the original objective.

Then we look at the global algorithm executed by one of the nodes in the federation in algo. 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, see 3, and *sendLatestModelToClients()* is a method that sends the latest global critic network to all the clients.

Algorithm 3 computeCoefficient

Require: Buffer with updates Buffer, target agent k
Ensure: Coefficient of agent k's update

$total\_k \leftarrow 0$
no_steps $\leftarrow 0$
for each entry $i \in$ Buffer **do**
  agent$_i \leftarrow$ agent that sent entry i
  steps$_i \leftarrow$ steps performed in i's update
  $total\_k \leftarrow total\_k +$ steps$_i$
  if $k ==$ agent$_i$ **then**
    no_steps $\leftarrow$ steps$_i$
  end if
end for
return no_steps$/total_k$

10 PeersimGym Environment and the POMG

10.1 Communication Protocols

There are three types of messages shared between the nodes. These are

- **Exchange of information to neighbors** - Our framework assumes that each node can share its local state only with directly connected neighbors through low-overhead broadcast or multicast mechanisms. This realistically mirrors real-world edge deployments, where full global state observability is impractical due to network size, reliability, and cost constraints.

- **Exchange of tasks** - Our framework assumes that tasks can be offloaded between directly connected nodes within their neighborhood, allowing localized workload distribution without reliance on centralized coordination.
- **Exchange of federated updates** - Model updates are propagated to the global node through multihop communication when direct connectivity is not available.

10.2 Observation Space

The observation space for agent p in node W^p at time step t consists of it's own queue size Q_t^p, the latest queue size known for each of it's neighbors $\{Q_t^j | W_j \in \tilde{W}^p\}$, where $\tilde{W}_p$ is the node W^p's neighborhood, and the percentage of space free for itself, F_t^p, and each of the neighbors, F_t^j, computed as:

$$F^n = Q_t^n / Q_{max}^n \tag{15}$$

Then on the task dimension they observe information about the tasks to be processed in particular the total number of instructions in the queue given by eq. 16, the total number of instructions of tasks assigned to be processed locally given by eq. 17 where $\mathbb{I}_{\text{local}_p}(\tau^i)$ is the identifier whether task τ^i was assigned to be processed locally in node p. Lastly, we have information on the next task to be processed, namely, its id i, the current progress of the task at time-step t, ρ_t^i, the total instructions, and the data input, α_i^{in}, and output size, α_i^{out}.

$$Q_{\rho,t}^p = \sum_{\tau_i \in Q_p} \rho^i \tag{16}$$

$$Q_{\rho,t}^{p,\text{local}} = \sum_{\tau_i \in Q_p} \mathbb{I}_{\text{local}}(\tau^i) \tag{17}$$

$$\mathbb{I}_{\text{local}_p}(\tau^i) = \begin{cases} \rho^i, & \text{if } \tau^i \text{ is local} \\ 0, & \text{otherwise} \end{cases} \tag{18}$$

Moreover, we convert the observations of all the agents to be structurally similar by normalizing and padding the observation spaces, ensuring consistent input dimensionality and robustness to network topological changes or node failures. Specifically, missing neighbors are represented using normalized placeholder values (-1), maintaining stable critic evaluation despite node heterogeneity.

11 Implementation Details

11.1 PPO Formulation

PPO Schulman et al. (2017) is a policy gradient method grounded in the Policy Gradient Theorem Sutton et al. (1999), which enables training a policy approximator by estimating the policy gradient and applying stochastic gradient ascent:

$$\hat{g} = \mathbb{E}_t \left[\nabla_\theta \log \pi_\theta(a_t \mid s_t) \hat{A}_t \right] \tag{19}$$

Here, π_θ is the policy being optimized, and $\hat{A}_t$ is an estimator for the Advantage Function computed with Generalized Advantage Estimation Schulman et al. (2018).

$$\hat{A}_t = \sum_{i=0}^{k-1} \gamma^i r_{t+i} + \gamma^k V(s_{t+k}) - V(s_t), \tag{20}$$

Here, k can vary from state to state and is upper-bounded by a parametrized value, N, while $V(\cdot)$ would be an estimator for the value function.

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:

$$L^{\text{CLIP}}(\theta) = \mathbb{E}_t \Big[\min \Big(r_t(\theta) \hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon) \hat{A}_t \Big) \Big] \tag{21}$$

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

$$r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}. \tag{22}$$

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 Schulman et al. (2017). 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. Schulman et al. (2017) recommend maximizing the following objective:

$$L_t^F(\theta, w) = \mathbb{E}_t\Big[L_t^{\text{CLIP}}(\theta) - c_1 L_t^{\text{VF}}(w) + c_2 S_{\pi_\theta}(s_t)\Big], \tag{23}$$

where we have the objective of the Actor, L^{CLIP}, as shown in eq. 21. The critic's loss function, $L^{\text{VF}}(w)$, where w is the parameters of the critic network, given by,

$$L_t^{\text{VF}}(w) = \left(r + \gamma V(s_{t+1}) - V(s_t)\right)^2. \tag{24}$$

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 following algo. 1 in annex 9.

11.2 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 performance, offering a familiar comparison point that helps contextualize the results.

We then consider the State-of-the-art synchronous FRL algorithm, SCOF Peng et al. (2024), 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 Peng et al. (2024) in FAuNO's repository. Since no public implementation of SCOF was available, we reimplemented the algorithm and released the code in FAuNO's repository. To ensure a fair comparison with FAuNO, which does not use DP, we disabled SCOF's DP component, as DP often reduces accuracy and was not the focus of this study. We further adapted SCOF to our Markov game formulation and edge setting, modifying the observation and reward structures for compatibility. These adaptations were applied consistently and do not disadvantage SCOF beyond removing features absent in FAuNO.

11.3 Hyperparameters used for FAuNO

We based our choice of hyperparameters on Andrychowicz et al. (2020). The parameters used for FAuNO:

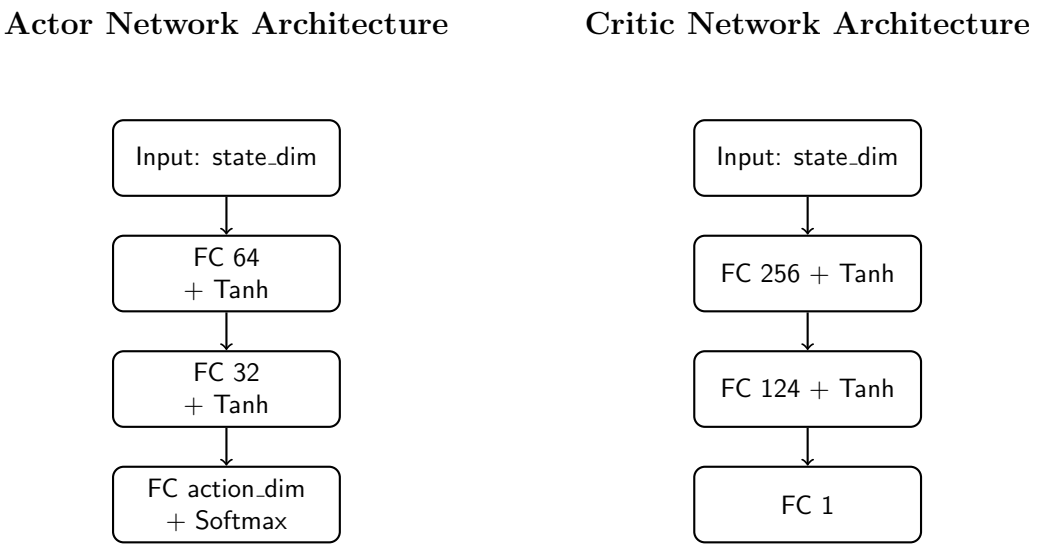
Table 10: Hyperparameters Used in FAuNO Experiments

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

For SCOF, we adopted the hyperparameter settings reported in the original paper Peng et al. (2024). For parameters not specified, we selected values empirically. A complete list of settings is provided in the repository under `configs/algo_configs`.

11.4 Network Architecture

The architectures for the PPO are based on the ones in Barhate (2024)



(a) Actor Network

(b) Critic Network

Figure 4: Actor-Critic Neural Network Representations

12 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[2]

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

12.1 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 SBCs, 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. 11, and the number of nodes taking up a given function is available in tab. 12.

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

Table 11: Cluster Composition Table

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

Table 12: 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. 13.

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. 14. We note that all time-dependent functions are scaled as well.

Table 13: Device Capacities

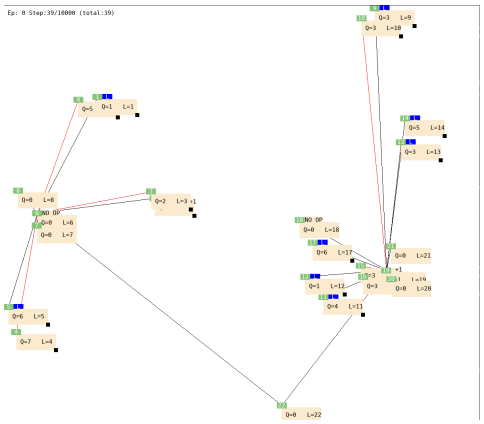
Device	CPU (Millis)	Memory (Bytes)
Raspberry Pi 4	7200	6442450944
Raspberry Pi 5 6GB	9600	6442450944
Raspberry Pi 6 8GB	9600	8589934592
Intel NUC	14800	68719476736
Cloudlet	290400	188000000000

Table 14: Parameter values in the experimental setup.

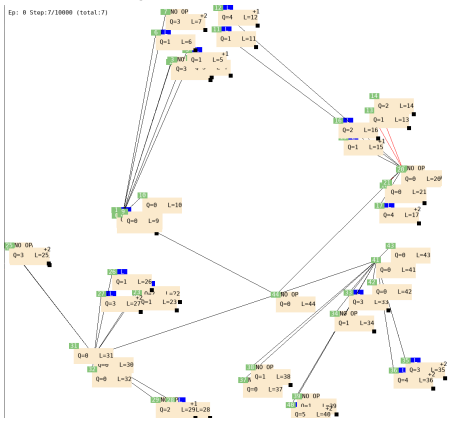
Simulation time, T	1000 s		
Task input size, α_i^{in}	150 Mbytes	**Task output size,** α_i^{out}	150 Mbytes
Task instructions, ρ_i	8×10^7	**Task utility,** r_u	100
CPI, ξ_i	1	**Weight waiting,** χ_D^{wait}	1
Deadline, δ_i	100	**Weight execution,** χ_D^{exc}	0.5
Bandwidth, $B_{i,j}$	4 MHz	**Weight comm,**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

12.2 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 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.13. The concrete number of nodes for each is given



(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. 11.



(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. 11.

Figure 5: Visualization of the different simulations used.

in tab. 15: We consider the same hyperparameters explained in 10. And consider similar training

No. Nodes	SBCs	NUCs
10	5	5
15	10	5

Table 15: Cluster Composition Table

conditions to the ether-based topologies.

12.3 Alibaba Cluster Trace-based Workload

The workload considered for the experiments in the paper was based on the integration of PeerSimGym Metelo et al. (2024) with an Alibaba Cluster trace-based workload generation tool Tian et al. (2019). 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% reduction, which maintained a meaningful level of computational demand without causing excessive task loss.

12.4 Computational Requirements

The tests were all executed in a private High-Performance Computer, orchestrated by Slurm. Each Slurm job utilized 16 GB of RAM memory, 4 cores, and a MiG partition with one compute partition and 10 GB 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.

13 Use of Large Language Models (LLMs)

Large language models were used solely as assistive tools to improve the clarity and readability of the manuscript. Their role was limited to editing for grammar, style, and wording. All research ideas, methodology, analysis, results, and conclusions were conceived and written by the authors. The authors carefully reviewed and verified all text to ensure accuracy and that the original meaning of the content was not violated, and we take full responsibility for the final content.