FEDORA: Federated Ensemble Reinforcement Learning for DAG-Based Task Offloading and Resource Allocation in MEC

Sangrez Khan, Amir Ali-Pour, Marios Avgeris, Julien Gascon-Samson, and Aris Leivadeas

***Abstract*—The increasing demand for compute intensive Internet of Thing (IoT) applications has accelerated the adoption of multi-access edge Computing (MEC) to offload tasks from resource constrained devices to edge servers. However, making optimal offloading decisions in multi-user MEC environments is challenging due to the dependencies between tasks, resource constraints, and the need to preserve user privacy. In this work, we propose FEDORA, a federated ensemble reinforcement learning framework for directed acyclic graph (DAG)-based task Offloading and resource allocation in MEC environments, that integrates twin delayed deep deterministic policy gradient (TD3) for continuous resource allocation and multi-head deep Q-networks (DQN) for discrete offloading decisions. To handle task dependencies, we model applications as DAGs and generate feature embeddings for offloading decisions. Our federated learning (FL) approach uses local training at MEC level and periodic model aggregation at a global server to preserve data privacy. Finally, extensive simulations across different DAG topologies demonstrate that FEDORA reduces system costs and improves task completion rates compared to state-of-the-art baselines including FL-DQN, FL-DDPG, FedAvg, FedNova, and SCAFFOLD, highlighting its scalability and robustness in large scale MEC deployments.**

***Index Terms*—DRL, Energy Efficiency, Federated learning, Graph Attention Networks, IoT, MEC, Task Offloading.**

I. Introduction

The rapid expansion of the Internet of Things (IoT) has led to a significant increase in the amount and complexity of data generated by resource-constrained, heterogeneous user devices [1]. Modern IoT applications, including real-time video analytics, autonomous vehicles, and industrial automation, require considerable computational resources and low-latency processing, surpassing the capabilities of individual IoT devices. Multi-access edge computing (MEC) addresses these challenges by bringing computational resources closer to the user [2]. This proximity reduces data transmission delays, eases pressure on centralized cloud infrastructures, and enhances energy efficiency through localized processing [3]. By enabling task offloading to edge servers, MEC facilitates a balanced distribution of computational workloads, preventing individual devices from becoming overburdened and optimizing resource utilization across the network [4].

Despite these advancements, a critical yet frequently underexplored aspect of task offloading in MEC is the interdependency among tasks within contemporary applications. Many real-world IoT applications, such as smart healthcare systems, collaborative robotics, and augmented reality, involve workflows where tasks are not independent but form complex dependency structures, often represented as directed acyclic graphs (DAGs) [5]. In these scenarios, the execution of one task may depend on the completion of others, introducing significant challenges in determining optimal offloading strategies [6].

This problem, referred to as *dependent task offloading*, involves making energy-efficient decisions regarding whether tasks should be executed locally or offloaded to edge servers [7]. These decisions significantly impact the energy consumption of IoT devices and the overall network performance, especially due to the intricate task inter-dependencies and varied computational resource requirements (e.g., data size and CPU processing resources), and the current state of the MEC infrastructure. Dependent task offloading has been shown to be an NP-hard problem, making exact solutions computationally infeasible for large-scale systems with dynamic conditions [8].

The traditional task offloading approaches in MEC often rely on centralized optimization or heuristic-based techniques, which require precise, real-time global system state information. These methods struggle with scalability, adaptability to fluctuating network conditions, and computational overhead, while also raising privacy concerns due to centralized data aggregation [9]. Moreover, heuristic solutions tailored to dependent tasks typically depend on static analytical models, limiting their ability to accommodate the evolving dynamics of IoT environments and stringent quality of service (QoS) requirements imposed by modern applications [10]. As MEC and IoT ecosystems continue to evolve, these limitations underscore the need for more flexible, adaptive, and privacy preserving strategies.

To address these challenges, recent research has pivoted toward distributed and adaptive methodologies, with deep reinforcement learning (DRL) gaining attraction. The integration of deep neural networks with reinforcement learning enables systems to autonomously learn optimal decision-making policies through interactions with dynamic environments, bypassing the need for explicit system modeling [11]. Techniques such as deep Q-networks (DQN) and actor-critic

S. Khan, A. Ali-pour, J. Gascon-Samson, and A. Leivadeas are with the Department of Software and IT Engineering, École de Technologie Supérieure (ÉTS), Université du Québec, Montréal, QC, Canada (e-mails: {sangrez.khan.1@ens., amir.ali-pour@, julien.gascon-samson@, aris.leivadeas@}etsmtl.ca).

M. Avgeris is with the Informatics Institute, Faculty of Science, University of Amsterdam (UvA), Amsterdam, The Netherlands (e-mail: m.avgeris@uva.nl).

methods excel in navigating the high-dimensional state and action spaces common in MEC task offloading scenarios [12]. However, conventional DRL approaches often overlook the intricate dependencies among tasks or simplify them into linear relationships, failing to capture the full complexity of DAG-based workflows. Additionally, their reliance on centralized data collection introduces significant communication overhead and privacy risks, while their stability and convergence can be challenging in large-scale, heterogeneous environments characterized by non-independent and identically distributed (non-IID) user behaviors. In response to these shortcomings, federated learning (FL) [13] has emerged as a privacy preserving distributed learning paradigm. FL empowers user devices and edge servers to collaboratively train models using local data, aggregating only model updates rather than raw data at a central entity. Apart from preserving user privacy, this approach reduces communication costs and leverages distributed computational resources [14]. The blending of FL and DRL offers a promising combination of adaptive decision-making with privacy and scalability. Yet, existing federated DRL frameworks have largely overlooked the complexities of dependent task offloading, particularly in scenarios involving interdependent tasks modeled as DAGs and the dynamic allocation of resources in distributed MEC settings. Recent advancements highlight the growing relevance of dependent task offloading, as applications increasingly exhibit DAG-based structures where task execution order and resource allocation are tightly coupled. These dependencies amplify the complexity of decision-making, as offloading one task may influence the feasibility and performance of subsequent tasks [15]. While some studies have employed heuristic methods to tackle this NP-hard challenge, their reliance on predefined models limits adaptability to real-time network variations. More recent efforts have begun to explore DRL-based solutions augmented with graph attention networks (GATs) [16], which leverage attention mechanisms to model task dependencies within DAGs effectively. GATs enhance the ability to capture long-term structural relationships, providing a richer representation of dependencies for offloading decisions. Nevertheless, these approaches often remain confined to single-user contexts, focus narrowly on isolated objectives such as energy or latency, while they generally inadequately address the dynamic nature of MEC environments [17], [18].

In this context, we introduce FEDORA, a federated ensemble reinforcement learning framework for DAG-based task offloading and resource allocation in MEC environments. FEDORA extends our previous work [19], where we introduced an energy-aware dependent task offloading scheme utilizing GATs combined with DRL to optimize the performance of MEC environments. In the current study, we further address the scalability, adaptability, privacy, and efficiency of task offloading and resource allocation decisions, particularly focusing on heterogeneous DAGs encountered in dense IoT deployment. For instance, in a smart surveillance system, an IoT camera may generate a DAG composed of object detection, frame encoding, and anomaly alerting tasks; in a smart manufacturing plant, sensor nodes may generate DAGs comprising signal preprocessing, defect detection, and report generation tasks [2]. Our system scenario specifically considers MEC-assisted IoT networks deployed over next-generation cellular infrastructures (e.g., 5G/6G), in application domains such as smart cities and industrial IoT settings, where IoT traffic is carried over network technologies like narrowband IoT (NB-IoT) or massive machine-type communications (mMTC). FEDORA iteratively executes three key phases: First, IoT devices generate heterogeneous, interdependent tasks and share task-specific metadata with the base station. Each base station independently collects this information within its coverage area and trains local DRL models based on the local network state and task dependencies. Second, the updated DRL model parameters from each base station are securely transmitted to a centralized aggregator, typically hosted at a MBS which employs federated averaging to synthesize a global model. Third, the aggregated global model is disseminated back to the IoT devices, enabling them to refine their local policies in a synchronized, privacy preserving manner. By addressing the NP-hard nature of dependent task offloading and leveraging federated DRL with GATs, FEDORA represents a robust, scalable, and privacy-aware solution for next-generation MEC environments. The primary contributions of this work are as follows:

- We propose a comprehensive system model specifically designed for DRL-assisted MEC environments. The model effectively captures the characteristics of heterogeneous IoT devices with interdependent tasks structured as DAGs, dynamically varying wireless conditions, and variable computational resource availability.
- We formulate the joint dependent task offloading and resource allocation challenge as a mixed-integer nonlinear programming (MINLP) problem. Considering the NP-hard nature and impracticality of traditional optimization techniques in dynamic and large-scale scenarios, we reformulate the problem as a Markov decision process (MDP).
- We develop FEDORA, a novel federated DRL framework integrating twin delayed deep deterministic policy gradient (TD3) for continuous resource allocation with multi-head deep Q-networks (DQN) for discrete task offloading decisions. Additionally, we employ GATs to accurately capture and represent complex inter-task dependencies.
- Our framework adopts a FedProx-based FL strategy, performing localized model training at MEC servers combined with periodic global model aggregation. This approach inherently ensures user data privacy, reduces communication overhead, and enhances scalability compared to centralized data aggregation methods.
- Extensive experimental evaluations demonstrate that FEDORA significantly outperforms conventional centralized methods, standard DRL approaches, and existing federated learning techniques in terms of energy efficiency, task completion latency, and QoS. Furthermore, FEDORA achieves faster convergence and lower communication overhead across various dynamic network scenarios.

The structure of the paper is as follows. Section II provides a state-of-the-art overview of the related literature. Section III introduces the system model and formulates the corresponding multi-objective optimization problem. In Section IV, we present the MDP formulation along with the proposed hybrid reinforcement learning approach, incorporating DQN and TD3, for dependent task offloading and resource allocation. Section V discusses the simulation results and performance evaluation. Finally, the conclusion of the paper is provided in Section VI.

II. Related Work

A. Traditional Centralized Methods

Conventional centralized approaches for task offloading and resource allocation in MEC predominantly utilize mathematical programming and heuristic optimization strategies. For instance, Mahmoodi *et al.* [20] introduced the joint scheduling and computation offloading (JSCO) framework, which uses the CPLEX optimizer to jointly minimize delay and energy consumption for DAG-based tasks in MEC environment. In a similar line, Liu *et al.* [21] formulated a task scheduling problem in edge computing environment and proposed the multiple applications multiple tasks scheduling (MAMTS) algorithm, which prioritizes tasks by estimated completion time, thereby reducing average delays and meeting critical deadlines. Xu *et al.* [22] developed dependency-aware task offloading for joint optimization of delay and energy consumption (DTO-JODE) scheme, targeting industry 5.0 applications, where an enhanced particle swarm optimization (PSO) algorithm is employed for efficient interdependent task offloading and server selection. Liu *et al.* [23] presented COFE, a framework designed to help mobile devices offload dependent task to a hybrid MEC cloud infrastructure. The task offloading challenge is modeled as an optimization problem aimed at minimizing the average execution time. To address this, the authors develop a heuristic algorithm based on task ranking, capable of operating in real-time. An *et al.* [24] proposed a joint optimization model for sequential task execution, by decomposing the main problem into two subproblems and solving it via the golden search method. Additionally, Liu *et al.* [25] designed ranking and foresight-integrated dynamic scheduling scheme (RFID), a scheduling solution tailored for vehicular cloud environments, which selects vehicles based on task dependencies, resource availability, and node connectivity to minimize processing time. The growing complexity of task requirements and network conditions poses significant challenges to these traditional computation offloading approaches. These existing methods typically demand numerous iterations to converge to a satisfactory local optimum, making them ineffective for real-time decision-making. Consequently, these approaches often incur substantial overhead, fall short of user expectations, and inadequately address the multi-constraint execution in MEC environments.

B. Distributed DRL-based Approaches

Recent advancements in DRL have led to the development of distributed approaches for task offloading and resource allocation, which do not require explicit knowledge of system dynamics and can adaptively optimize decisions through interactions with the environment. Liu *et al.* [26] proposed a distributed collaborative dependent task offloading strategy based on a DRL (DCDO-DRL) scheme to effectively address the challenges of offloading radiomics-based medical image diagnosis model (RIDM) tasks modeled as DAGs. DCDO-DRL uses sequence to sequence (S2S) and soft actor-critic to optimize the use of limited computing resources in hospitals by offloading tasks to edge servers. Feng *et al.* [27] addressed task dependency challenges by proposing a dependency-aware task reconfiguration and offloading framework, effectively decomposing complex multi-component tasks to facilitate efficient resource allocation across IoT devices. While these methods effectively address dynamic interactions, the inherent challenge of maintaining stable learning and convergence in the presence of non-IID data across distributed agents remains significant.

C. Federated Learning-enabled DRL Methods

FL has emerged as an effective mechanism to enhance the robustness and privacy of distributed learning models in MEC environments. Integrating FL with DRL further improves performance by addressing the challenges posed by non-IID data and privacy concerns. Xiao *et al.* [28] introduced federated deep reinforcement learning for task offloading in MEC-enabled heterogeneous networks, showing significant improvements in energy efficiency and quality of service. Wu *et al.* [29] combined FL with multi-agent reinforcement learning for vehicular edge computing, demonstrating reductions in latency and energy consumption while improving task completion rates. Zhao *et al.* [30] extended federated DRL to vehicular networks, effectively managing both task offloading and resource allocation under privacy constraints. Zhou *et al.* [31] explored federated distributed deep reinforcement learning specifically for recommendation-enabled edge caching, significantly reducing content delivery delay and improving cache hit rates. Additionally, Zhao *et al.* [14] utilized federated deep reinforcement learning for secure video offloading in Industrial Internet of Things (IIoT) networks, effectively balancing latency, energy consumption, and security considerations. Shen *et al.* [32] proposed an asynchronous federated deep reinforcement learning (FDRL) framework for task offloading in UAV-assisted vehicular networks. They introduced a dependency-aware MEC model that represents applications as DAGs, and formulated a joint optimization problem to minimize task delay and energy consumption. Proximal policy optimization (PPO), enables UAVs to collaboratively train offloading policies without sharing raw data. However, this work lacks the consideration of resource allocation. Similarly, Zhao *et al.* [33] proposed a privacy-preserving DAG task offloading framework in MEC environments using a federated deep Q-network (FDQN) with automated hyperparameter tuning via the TPE algorithm. Their approach jointly optimizes response time and energy consumption while addressing task dependencies, system heterogeneity, and privacy concerns.

In summary, as demonstrated in Table I while various recent works have explored dependent task offloading using

TABLE I: Comparison of Related Works on Dependent MEC Task Offloading and Resource Allocation

Reference	**Solution Type**	**Energy Minimizing**	**Delay Minimizing**	**Resource Allocation**	**DAG Modeling**	**Dynamic Network**	**Privacy (FL)**
Mahmoodi *et al.* [20]	Traditional	✓	✓		✓		
Liu *et al.* [21]	Traditional		✓		✓		
Xu *et al.* [22]	Traditional	✓	✓	✓	✓		
Liu *et al.* [23]	Traditional		✓		✓	✓	
An *et al.* [24]	Traditional	✓	✓	✓	✓		
Liu *et al.* [25]	Traditional		✓	✓	✓	✓	
Liu *et al.* [26]	DRL	✓	✓	✓	✓	✓	
Feng *et al.* [27]	DRL			✓	✓	✓	
Xiao *et al.* [28]	FL-DRL	✓	✓				✓
Wu *et al.* [29]	FL-DRL	✓	✓				✓
Zhao *et al.* [30]	FL-DRL		✓	✓			✓
Zhou *et al.* [31]	FL-DRL		✓				✓
Zhao *et al.* [14]	FL-DRL	✓	✓				✓
Shen *et al.* [32]	FL-DRL	✓	✓		✓		✓
Zhao *et al.* [33]	FL-DRL	✓	✓		✓		✓
FEDORA	**FL-DRL**	✓	✓	✓	✓	✓	✓

optimization, heuristic, RL, or FL approaches, they still exhibit significant limitations. Many existing models either neglect or inadequately handle complex task interdependencies, often considering only simple linear or sequential task relationships[34]. Furthermore, most of these studies do not comprehensively integrate all practical constraints such as stringent energy limitations, delay tolerance, resource allocation decisions, dynamic network conditions, and scalability in multi-user scenarios. In contrast, FEDORA uniquely incorporates GAT for capturing complex DAG-based task dependencies, leverages federated reinforcement learning for scalable and privacy preserving distributed decision-making, and rigorously considers comprehensive constraints reflective of realistic IoT-based MEC environments.

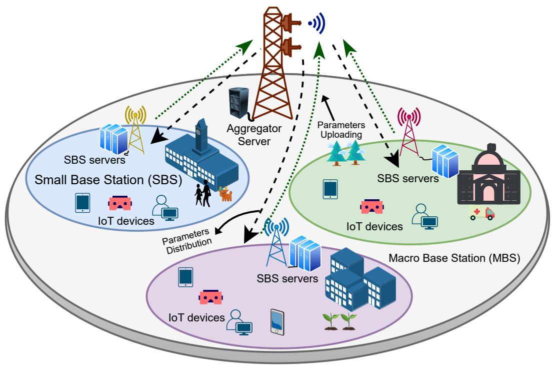


Fig. 1: Detailed system model illustrating a Multi-tier IoT architecture integrated with MEC & FL.

III. System Model

We consider a two-tier MEC architecture composed of one MBS, multiple small base stations (SBSs), multiple IoT devices within each SBS, and multiple MEC servers per SBS as depicted in Fig. 1. SBSs are denoted by $\mathbb{M} = \{1, 2, \ldots, |\mathbb{M}|\}$, where each SBS $m \in \mathbb{M}$ serves a set of IoT devices $\mathbb{N}_m = \{1, 2, \ldots, |\mathbb{N}_m|\}$. Each device $n \in \mathbb{N}_m$ executes an application that is modeled as DAG denoted as $\mathbb{D}_{m,n} = (X_{m,n}, Y_{m,n})$. In each DAG the nodes represent the dependent tasks and are indexed by $X_{m,n}$= $\{x^1_{m,n}, x^2_{m,n}, \ldots, x^i_{m,n}, \ldots, x^{|X_{m,n}|}_{m,n}\}$ while $Y_{m,n}$ represents the dependency between the tasks and is described by a

TABLE II: Summary of Main Notations

Notation	**Description**
	System Model
$\mathbb{M}$	Set of Small Base Stations (SBSs)
$\mathbb{N}_m$	Set of IoT devices under SBS m
$\mathbb{S}_m$	Set of MEC servers at SBS m
$\mathbb{D}_{m,n}$	DAG representing the application for device n under SBS m
$X_{m,n}$	Set of tasks for device n under SBS m
$x^i_{m,n}$	i-th task of the application on device n under SBS m
$c^i_{m,n}$	Required CPU cycles for task $x^i_{m,n}$
$d^i_{m,n}$	Data size of task $x^i_{m,n}$ (kB)
$\delta^{i,s}_{m,n}$	Binary offloading decision (1 if task $x^i_{m,n}$ is executed on server s; 0 otherwise)
$f^{i,s}_{m,n}$	Allocated CPU frequency for task $x^i_{m,n}$ on server/device s (GHz)
$T^{i,s}_{m,n}$	Execution time of task $x^i_{m,n}$ on server/device s (ms)
$E^{i,s}_{m,n}$	Energy consumption for task $x^i_{m,n}$ on server/device s (mJ)
$R^u_{m,n}, R^d_{m,n}$	Uplink/downlink transmission rates (Mbps)
$P^u_{m,n}, P^d_{m,s}$	Uplink/downlink transmission power (W)
$\mathcal{G}_{m,s}, \mathcal{H}_{m,n}$	Downlink/uplink channel gains (dB)
$\kappa_{m,n}$	Energy efficiency coefficient of device n's processor
$\widetilde{E}_n$	Residual energy of n IoT device
α	Weighting factor for energy-delay trade-off
	DRL & FL Components
$\mathcal{S}_t$	System state at time t (residual energy, task embeddings, channel conditions)
$\mathcal{A}_t$	Hybrid action (offloading decisions $\delta^{i,s}_{m,n}$ and CPU allocation $f^{i,s}_{m,n}$)
$\mathcal{R}_t$	Reward function balancing latency, energy, and constraints
$\theta^{(d)}_k$	Parameters of multi-head DQN for discrete offloading
$\phi_k, \psi_{k,1}, \psi_{k,2}$	Parameters of TD3 actor and critics for continuous resource allocation
$\mathbb{B}_k$	Experience replay buffer for agent k
γ	Discount factor for future rewards ($0 < \gamma < 1$)
τ	Soft update rate for target networks ($0 < \tau \ll 1$)
μ	Proximal coefficient in FedProx aggregation
$\mathcal{E}_k$	Number of local training epochs
R	Total federated training rounds
$\mathbb{M}_r \subseteq \mathbb{M}$	Subset of participating agents in round r

binary adjacency matrix $\Omega_{m,n} \in \{0,1\}^{|X_{m,n}|\times|X_{m,n}|}$, where $[\Omega_{m,n}]^{i,j} = 1$ if task $x^j_{m,n}$ depends on task $x^i_{m,n}$, and 0 otherwise. The number of CPU cycles required to execute each task $x^i_{m,n} \in X_{m,n}$ is denoted as $c^i_{m,n}$, and the data size is given as $o^i_{m,n}$. Each SBS m also contains multiple MEC servers, indexed by $\mathbb{S}_m = \{1, 2, \ldots, |\mathbb{S}_m|\}$. The execution of each task is defined by a start time, $\Pi^{i,s}_{m,n}$, and a completion time, $\mathcal{C}^{i,s}_{m,n}$, where s indicates the designated local device or edge server. We explicitly assume the use of Orthogonal Frequency Division Multiple Access (OFDMA), where each IoT device is allocated orthogonal frequency subcarriers for uplink and downlink transmissions. By design, OFDMA inherently mitigates co-channel interference among simultaneously transmitting devices [35]. Given the dependency constraints among tasks, a task can commence execution only after all preceding tasks have been successfully completed and the necessary data have been transmitted. For clarity and ease of modeling, we assume that each IoT device is capable of executing only one task at any given time, and the initial task is executed locally.

The MBS $\mathbb{A}$ serves as the central aggregator and global model manager, coordinating all SBSs in the system. It periodically receives local model updates from SBSs and aggregates them using federated aggregation. The MBS has significantly higher computational capacity compared to SBSs and devices, i.e., $F^{\mathbb{A}} \gg F_{m,s}, \quad \forall m \in \mathbb{M}, \quad s \in \mathbb{S}_m$, where $F^{\mathbb{A}}$ and $F_{m,s}$ are the computational capacity of the MBS and SBS servers, respectively. To effectively manage task execution within an SBS, we define $\delta^{i,s}_{m,n}$, a binary task placement decision variable that determines the execution location of task $x^i_{m,n} \in X_{m,n}$:

$$\delta^{i,s}_{m,n} = \begin{cases} 1, & \text{if task } x^i_{m,n} \text{ is executed} \\ & \text{on MEC server s at SBS } m, \\ 0, & \text{otherwise,} \end{cases} \tag{1}$$

where $s = 0$ corresponds to local execution at IoT device n, and $s \in \mathbb{S}_m$ represents offloading to one of the $\mathbb{S}$ MEC servers within the same SBS m. Each task must be assigned to exactly one execution location:

$$\sum_{s=0}^{|\mathbb{S}_m|} \delta^{i,s}_{m,n} = 1, \forall x^i_{m,n} \in X_{m,n}, \forall n \in \mathbb{N}_m, \forall m \in \mathbb{M}. \tag{2}$$

The amount of CPU frequency allocated to a task at its selected execution location is denoted by $f^{i,s}_{m,n}$, which serves as another decision variable in the optimization problem and is given as:

$$0 \le f^{i,s}_{m,n} \le F^{m,s}_{\max}\, \delta^{i,s}_{m,n}, \forall s \in \mathbb{S}_m, \forall n \in \mathbb{N}_m, \forall m \in \mathbb{M}. \tag{3}$$

$F^{m,s}_{\max}$ denotes the maximum processing capacity available at MEC server s in SBS m. This constraint ensures that $f^{i,s}_{m,n}$ is only allocated when the task is executed at location s.

A. Computation and Communication Model

1) Local Execution Model

When a task $x^i_{m,n}$ is processed locally on an IoT device, the execution delay is defined as:

$$T^{i,0}_{m,n} = \frac{c^i_{m,n}}{f^{i,0}_{m,n}}, \quad \forall m \in \mathbb{M}, \forall n \in \mathbb{N}_m, \forall i \in X_{m,n}, \tag{4}$$

where $c^i_{m,n}$ denotes the computational complexity in CPU cycles required for task completion, and $f^{i,0}_{m,n}$ is the processing frequency allocated to the task at the IoT device. Local execution start time is affected by its dependency structure:

$$\Pi^{i,0}_{m,n} = \max_{j \in \text{pre}(i)} \{\delta^{j,0}_{m,n}\mathcal{C}^{j,0}_{m,n} + \sum_{s=1}^{|\mathbb{S}_m|} \delta^{j,s}_{m,n}(\mathcal{C}^{j,s}_{m,n} + \mathcal{T}^{j,d}_{m,n})\}, \tag{5}$$

where pre(i) represents the set of predecessor tasks; $\mathcal{T}^d_{m,n,j}$ denotes the time required to receive the results of an offloaded predecessor task:

$$\mathcal{T}^{i,d}_{m,n} = \frac{o^i_{m,n}}{R^d_{m,n}}, \quad \forall m \in \mathbb{M}, \forall n \in \mathbb{N}_m, \forall i \in X_{m,n}, \tag{6}$$

where $o^i_{m,n}$ represents the output data size, and the downlink transmission rate, based on the Shannon capacity formula, is given by:

$$R^d_{m,n} = B_{m,n} \log_2(1 + \frac{P^d_{m,s}\mathcal{G}_{m,s}}{\sigma^2}). \tag{7}$$

The overall task completion time when executed locally is:

$$\mathcal{C}^{i,0}_{m,n} = \Pi^{i,0}_{m,n} + T^{i,0}_{m,n}. \tag{8}$$

The local execution energy consumption is modeled as:

$$E^{i,0}_{m,n} = \kappa_{m,n} c^i_{m,n} (f^{i,0}_{m,n})^2, \tag{9}$$

where $\kappa_{m,n}$ is the energy efficiency coefficient of the device.

2) Edge Server Execution Model

For tasks offloaded to an SBS edge server, the overall execution time consists of uplink transmission, edge processing, and downlink result retrieval. The uplink transmission delay is given by:

$$\mathcal{T}^{i,u}_{m,n} = \frac{o^i_{m,n}}{R^u_{m,n}}, \quad \forall m \in \mathbb{M}, \forall n \in \mathbb{N}_m, \forall i \in X_{m,n} \tag{10}$$

where the uplink transmission rate follows Shannon's capacity formula:

$$R^u_{m,n} = B_{m,n} \log_2(1 + \frac{P^u_{m,n}\mathcal{H}_{m,n}}{\sigma^2}). \tag{11}$$

After data transmission, the execution delay at the edge server depends on its allocated computing resources:

$$T^{i,s}_{m,n} = \frac{c^i_{m,n}}{f^{i,s}_{m,n}}, \tag{12}$$

The task start time at the edge server is:

$$\Pi^{i,s}_{m,n} = \max_{j \in \text{pre}(i)} \{\delta^{j,0}_{m,n}(\mathcal{C}^{j,0}_{m,n} + \mathcal{T}^{j,u}_{m,n}) + \sum_{s=1}^{|\mathbb{S}_m|} \delta^{j,s}_{m,n}\mathcal{C}^{j,s}_{m,n}\}. \tag{13}$$

The total task completion time when offloaded to the SBS is:

$$\mathcal{C}^{i,s}_{m,n} = \Pi^{i,s}_{m,n} + T^{i,s}_{m,n}. \tag{14}$$

The corresponding uplink energy consumption is:

$$E^{i,u}_{m,n} = P^u_{m,n}\, \mathcal{T}^{i,u}_{m,n}. \tag{15}$$

B. Overall System Delay and Energy Consumption

The total task completion delay considering both local and offloaded executions is formulated as:

$$T_{m,n}=\sum_{i=1}^{|X_{m,n}|}[\delta_{m,n}^{i,0}\,T_{m,n}^{i,0}\sum_{s=1}^{|\mathbb{S}_m|}\delta_{m,n}^{i,s}(\mathcal{T}_{m,n}^{i,u}+T_{m,n}^{i,s}+\mathcal{T}_{m,n}^{i,d})]. \tag{16}$$

Similarly, the total energy consumption of an IoT device n is expressed as:

$$E_{m,n}=\sum_{i=1}^{|X_{m,n}|}[\delta_{m,n}^{i,0}\,E_{m,n}^{i,0}+\sum_{s=1}^{|\mathbb{S}_m|}\delta_{m,n}^{i,s}\,E_{m,n}^{i,u}]. \tag{17}$$

C. Problem Formulation

We tackle the twofold objective of minimizing the total energy consumption and total task completion delay across all devices, in a system where each device has a set of computational tasks to process. We only consider IoT device energy consumption, under the assumption that edge servers possess abundant energy resources; therefore, server-side energy consumption is excluded. The total system energy consumption is given by:

$$E_{\text{total}}=\sum_{m\in\mathcal{M}}\sum_{n\in\mathcal{N}_m}E_{m,n}.$$

Similarly, the total system completion delay is:

$$T_{\text{total}}=\sum_{m\in\mathcal{M}}\sum_{n\in\mathcal{N}_m}T_{m,n}.$$

Thus, the joint energy-delay optimization problem is formulated as:

$$\min_{\delta_{m,n}^{i,s},f_{m,n}^{i,s}}\quad\sum_{m\in\mathcal{M}}\sum_{n\in\mathcal{N}_m}(T_{m,n}+\alpha E_{m,n}) \tag{18}$$

$$\text{s.t.}\quad E_{m,n}\le E_{m,n}^{\max},\quad\forall m\in\mathbb{M},\forall n\in\mathbb{N}_m, \tag{C1}$$

$$T_{m,n}\le T_{m,n}^{\max},\quad\forall m\in\mathbb{M},\forall n\in\mathbb{N}_m, \tag{C2}$$

$$\sum_{s=0}^{|S_m|}\delta_{m,n}^{i,s}=1,\quad\forall m\in\mathbb{M},\;\forall n\in\mathbb{N}_m,\forall i\in X_{m,n}, \tag{C3}$$

$$0\le f_{m,n}^{i,s}\le F_{\max}^{m,s}\delta_{m,n}^{i,s},\quad\forall m\in\mathbb{M},\;\forall s\in\mathbb{S}_m,\forall n\in\mathbb{N}_m,\forall i\in X_{m,n}, \tag{C4}$$

$$\sum_{n\in\mathbb{N}_m}\sum_{i\in X_{m,n}}\frac{D_{m,n,i}\delta_{m,n}^{i,s}}{R_{m,n}^{s}}\le C_m,\quad\forall m\in\mathbb{M},\forall s\in\mathbb{S}_m. \tag{C5}$$

where $0\le\alpha\le1$ in the objective function (Eq. 18) is a weighting factor to balance energy and delay. Constraint (C1) ensures that the energy consumed by device n under SBS m does not exceed its maximum allowed energy budget $E_{m,n}^{\max}$. Constraint (C2) ensures that the completion time of each task does not exceed the device's delay tolerance $T_{m,n}^{\max}$. Constraint (C3) guarantees that each task is executed at exactly one location (locally or MEC server), while (C4) limits CPU frequency allocation to the capacity of the selected execution location. Constraint (C5) ensures that the total uplink communication demand at each SBS does not exceed its capacity C_m. The problem formulated in Eq. 18 is an NP-hard MINLP due to the combinatorial explosion resulting from mixed discrete and continuous decision variables, nonlinear relationships in transmission rates, and computational complexity introduced by the interdependencies of DAGs. Consequently, traditional exact optimization methods, which typically require significant computational resources and execution time, become computationally infeasible, especially under large-scale scenarios and rapidly changing network conditions [18]. Moreover, these traditional methods fail to meet the strict real-time execution requirements critical for IoT applications [3]. To overcome these challenges, we propose a FL-based reinforcement learning solution, FEDORA, which efficiently distributes computation across multiple devices and edge servers, enhances scalability, preserves data privacy, and significantly reduces communication overhead.

IV. Federated Learning-Enabled DRL Offloading & Resource allocation

In this section, we describe the integration of FL into DRL, our framework to enable distributed, privacy preserving, and scalable task offloading in MEC. In such systems, offloading decisions are highly sensitive to the underlying task graph topology, wireless channel conditions, and device energy budgets. Centralized training approaches, which rely on aggregating all local interaction data, are impractical due to privacy concerns, bandwidth limitations, and the presence of non-IID user workloads shaped by heterogeneous DAGs. To address these challenges, we propose a FedProx- [36] based federated reinforcement learning (FRL) framework. By introducing a proximal regularization term into each agent's local objective, FedProx enhances convergence stability under statistical heterogeneity, while allowing decentralized optimization across clients. We specifically select FedProx due to its proven effectiveness in mitigating client drift and ensuring stable convergence under heterogeneous, non-IID local data distributions common in MEC systems. Unlike standard federated methods, FedProx explicitly constrains local model updates to prevent excessive divergence from the global model. This choice makes FedProx uniquely suited to maintaining robust model performance and reliability in our federated DRL setting. In our framework, as shown in Figure, 2, each edge agent trains a local DRL model on its private experiences and transmits only model parameters to a central aggregator. The aggregator fuses these updates into a global model, which is then broadcast back to all agents. This exchange avoids raw data sharing, significantly reduces communication costs, and supports privacy-aware large-scale learning. In the following we explain the main phases of our framework in detail.

A. Markov Decision Process Formulation

To effectively model our joint offloading and resource allocation problem within the proposed MEC environment,

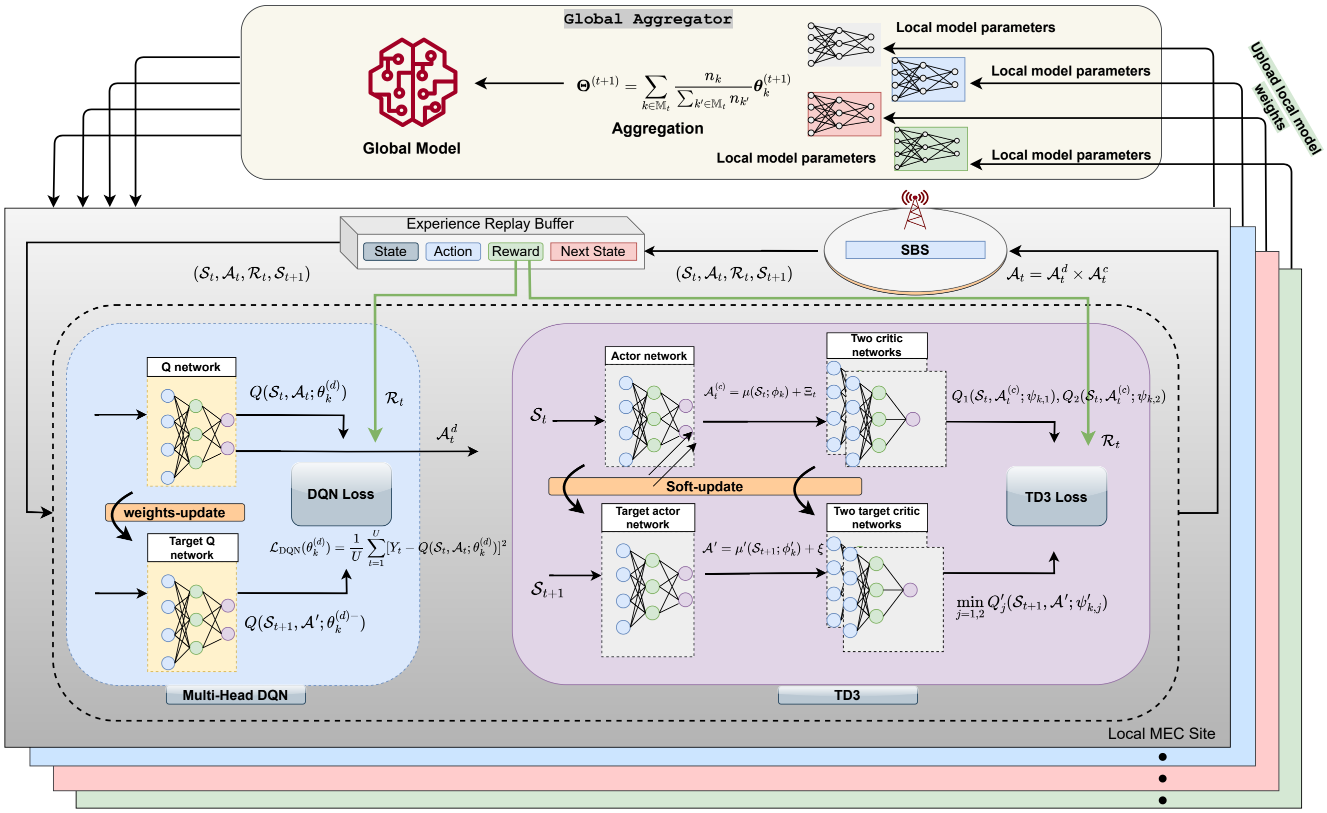


Fig. 2: FEDORA Architecture

we represent it as an MDP. An MDP provides a systematic way to formalize sequential decision-making problems, especially suitable for stochastic and dynamic environments. Specifically, an MDP is characterized by the tuple $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma\}$ [37], where $\mathcal{S}$ denote the state, $\mathcal{A}$ is for action, $\mathcal{P}$ denotes transition probability while $\mathcal{R}$ and γ represents the reward and discount factor respectively. However, due to the high-dimensional and partially continuous nature of our state space, the state transition probability distribution $\mathcal{P}$ cannot be explicitly defined. Therefore, we utilize a simplified model-free MDP formulation expressed as $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{R}\}$. In the following, we explicitly define each component in detail.

1) State ($\mathcal{S}$): In our system model, time is divided into discrete time steps t, where system observations are updated and offloading or resource allocation actions are taken. At each t, the current state $\mathcal{S}_t$ captures comprehensive information reflecting the status of IoT devices, applications, MEC servers, and wireless communication conditions. More explicitly, the state includes the residual energy level of each IoT device n, denoted as $\widetilde{E}_n(t)$, $O_n(t)$, denotes the number of tasks executed at t. The application specific embeddings (H'_n), capturing the long-term dependencies among tasks within the application DAG, are derived via a GAT model, while the channel conditions at t, are given by downlink and uplink channel gains $(\mathcal{G}_{s\,n}(t), \mathcal{H}_{n\,s}(t))$ between IoT devices and MEC servers. Thus, the state at t is formally represented as:

$$\mathcal{S}_t = \{\widetilde{E}_n(t), O_n(t), H'_n(t), \mathcal{G}_{s\,n}(t), \mathcal{H}_{n\,s}(t) \mid \forall n \in \mathbb{N}, s \in \mathbb{S}\}. \tag{19}$$

A terminal state occurs when all tasks for each IoT device have been allocated, i.e., $O_n(t) = |X_n|, \forall n \in \mathbb{N}$.

2) Action ($\mathcal{A}$): The action at decision t jointly encompasses discrete task offloading/placement decisions and the CPU continuous resource allocations. Formally, at each t, the agent selects the following combined action:

$$\mathcal{A}_t = \{(\delta^{i,s}_{m,n}(t), f^{i,s}_{m,n}(t)) \mid \forall m \in \mathbb{M}, n \in \mathbb{N}_m, i \in X_{m,n}, s \in \mathbb{S}_m \cup \{0\}\}, \tag{20}$$

3) Reward ($\mathcal{R}$): The reward function is designed to intuitively guide the DRL agent towards optimal decisions that simultaneously minimize latency and energy consumption, maximize task completion within defined constraints, and encourage compliance with resource limitations. Specifically, after executing an action $\mathcal{A}_t$ in state $\mathcal{S}_t$, the agent receives a reward $\mathcal{R}_t$, includes the combined weighted penalty of normalized average latency and energy usage. The reward positively considers the ratio of tasks successfully completed within latency and energy constraints. To encourage constraint satisfaction, the reward incorporates strong penalties and action masking mechanisms, thereby discouraging infeasible resource allocations. Violations of energy and latency constraints are also penalized explicitly, guiding the agent towards consistently feasible and efficient decisions. Formally, the reward at t is defined as:

$$\begin{aligned}\mathcal{R}_t(\mathcal{S}_t, \mathcal{A}_t) &= -(w_d \frac{T_{\text{avg}}}{T_{\max} + \epsilon} + w_e \frac{E_{\text{avg}}}{E_{\max} + \epsilon}) + \widetilde{R} \\ &- \sum_{i \in \mathcal{S}} \max(0, \sum_j A_{i,j} - 1) - V_e \frac{\Delta E}{E_{m,n}^{\max}} - V_d \frac{\Delta T}{T_{m,n}^{\max}},\end{aligned} \tag{21}$$

where each term contributes to guiding the agent:

The first term, $-\left(w_d \frac{T_{\text{avg}}}{T_{\max}+\epsilon} + w_e \frac{E_{\text{avg}}}{E_{\max}+\epsilon}\right)$, penalizes high average latency (T_{avg}) and energy consumption (E_{avg}) relative to their maximums ($T_{\max}$, $E_{\max}$). Here, $w_d > 0$ and $w_e > 0$ are weights that balance the importance of latency and energy objectives, respectively. A higher w_d prioritizes latency reduction (potentially increasing energy), while a higher w_e prioritizes energy efficiency (potentially increasing latency). $\epsilon > 0$ is small constant for numerical stability. $\widetilde{R} \in [0, 1]$ positively rewards the agent based on the ratio of tasks successfully completed within their latency and energy constraints. $-\sum_{i \in \mathcal{S}} \max(0, \sum_j A_{i,j} - 1)$ strongly penalizes infeasible resource allocations. $A_{i,j}$ is the resource allocated for task j on server i, with server capacity normalized to 1. This term activates if the sum of allocations on any server exceeds its capacity. The final two terms, $-V_e \frac{\Delta E}{E_{m,n}^{\max}}$ and $-V_d \frac{\Delta T}{T_{m,n}^{\max}}$, impose explicit penalties for violating global energy and latency thresholds. $V_e, V_d \in \{0, 1\}$ are binary indicators for energy and delay violations, respectively. $\Delta E, \Delta T$ are violation margins relative to thresholds $E_{m,n}^{\max}, T_{m,n}^{\max}$, respectively. These terms guide the agent towards consistently feasible and efficient decisions.

B. Local Training Phase

In the local training phase, each agent $k \in \mathbb{M}$ independently interacts with its local MDP, defined by the tuple $\mathcal{M}_k = (\mathcal{S}_k, \mathcal{A}_k, \mathcal{R}_k, \gamma)$, to optimize its local DRL policy parameters $\boldsymbol{\theta}_k$. Here, $\mathcal{S}_k$ represents the SBS system level state observations (as defined in Section IV-A), $\mathcal{A}_k = \mathcal{A}_k^d \times \mathcal{A}_k^c$ is a hybrid action space comprising discrete offloading decisions ($\mathcal{A}_k^d$) and continuous CPU resource allocations ($\mathcal{A}_k^c$), $\mathcal{R}_k$ denotes the reward function, and $\gamma \in (0, 1)$ is the discount factor that balances immediate and future rewards. A fully discrete approach would require discretizing the continuous resource allocation space, leading to significant approximation errors, increased computational complexity, and potentially suboptimal solutions. Conversely, a fully continuous approach would necessitate treating offloading decisions as continuous actions, followed by rounding or thresholding to discrete values, which could lead to inaccurate representations and unstable learning due to discontinuities in decision space. Therefore, given the hybrid nature of the action space, we adopt a hybrid actor-critic architecture to effectively handle both discrete and continuous actions. For discrete offloading decisions, we employ a multi-head DQN, parameterized by θ_k^d, to approximate the optimal action-value function $Q^*(\mathcal{S}_t, \mathcal{A}_t)$, where $\mathcal{A}_t \in \mathcal{A}_k^d$. For continuous CPU resource allocations, we utilize the TD3 algorithm, which complements the DQN by handling the continuous action space. The local model parameters are $\boldsymbol{\theta}_k = (\theta_k^d, \psi_{k,1}, \psi_{k,2}, \phi_k)$, where θ_k^d is the multi-head DQN for discrete actions, $\psi_{k,1}$ and $\psi_{k,2}$ are TD3 critic networks, and ϕ_k is the TD3 actor. The global parameters $\mathbf{\Theta} = (\Theta^d, \Psi_1, \Psi_2, \Phi)$ are the aggregated versions of the local models, with $\mathbf{\Theta}' = (\Theta^{d-}, \Psi_1', \Psi_2', \Phi')$ denoting their corresponding target networks.

1) Multi-head DQN for Discrete Offloading Decisions

To address the complexity of simultaneous offloading decisions for multiple tasks, the multi-head DQN architecture is designed for scalability and computational efficiency. The neural network consists of shared hidden layers that learn common feature representations from the system state $\mathcal{S}_t$, followed by multiple independent output heads. Each head corresponds to a specific task or group of related tasks and outputs Q-values for the discrete offloading decisions associated with that task. This structure reduces computational overhead and enhances the model's ability to generalize across tasks compared to traditional DQN. The multi-head DQN approximates the optimal Q-function as:

$$Q(\mathcal{S}_t, \mathcal{A}_t; \theta_k^d) \approx Q^*(\mathcal{S}_t, \mathcal{A}_t), \tag{22}$$

where $Q^*(\mathcal{S}_t, \mathcal{A}_t)$ represents the optimal expected cumulative discounted reward achievable by taking action $\mathcal{A}_t \in \mathcal{A}_k^d$ (a vector of offloading decisions) in state $\mathcal{S}_t$.

To ensure stable and efficient training, the multi-head DQN leverages an experience replay buffer $\mathbb{B}_k$, which stores historical transitions $(\mathcal{S}_t, \mathcal{A}_t, \mathcal{R}_t, \mathcal{S}_{t+1})$. These transitions consist of the observed state, action taken, reward received, and the resulting next state. During each training iteration, a mini-batch of size U is randomly sampled from $\mathbb{B}_k$. Random sampling breaks temporal correlations among sequential experiences, stabilizing gradient updates and improving learning efficiency.

To further enhance training stability, a separate target network, parameterized by θ_k^{d-}, is employed. The target network generates stable Q-value estimates and is periodically updated to align with the primary network parameters θ_k^d. Updates to the target network are performed via soft updates:

$$\theta_k^{d-} \leftarrow \tau \theta_k^d + (1 - \tau_d) \theta_k^{d-}, \quad 0 < \tau_d \ll 1, \tag{23}$$

where τ_d is a small soft update rate that ensures gradual and consistent updates to the target network.

In each training step, the multi-head DQN minimizes the mean squared error (MSE) between the predicted Q-values and the target Q-values. For a mini-batch of U transitions, the target Q-value Y_t for each transition is computed using the target network:

$$Y_t = \mathcal{R}_t + \gamma \max_{\mathcal{A}'} Q(\mathcal{S}_{t+1}, \mathcal{A}'; \theta_k^{d-}), \tag{24}$$

where γ is the discount factor, and $\mathcal{A}' \in \mathcal{A}_k^d$ represents the possible actions in the next state $\mathcal{S}_{t+1}$. The loss function for the multi-head DQN is defined as:

$$\mathcal{L}_{\text{DQN}}(\theta_k^d) = \frac{1}{U} \sum_{t=1}^{U} [Y_t - Q(\mathcal{S}_t, \mathcal{A}_t; \theta_k^d)]^2. \tag{25}$$

The network parameters θ_k^d are updated by minimizing this loss using gradient-based optimizers, such as stochastic gradient descent (SGD) or Adam, which iteratively adjust the parameters to reduce the prediction error. To balance exploration and exploitation during training, the multi-head DQN employs an epsilon greedy strategy. At each time step t, the offloading decision action $\mathcal{A}_t^d$ is selected as follows:

$$\mathcal{A}_t^d = \begin{cases} \text{random action}, & \text{if } \zeta < \Xi^d, \\ \arg\max_{\mathcal{A}} Q(\mathcal{S}_t, \mathcal{A}; \theta_k^d), & \text{otherwise.} \end{cases} \tag{26}$$

Here, $\zeta \sim \mathcal{U}(0,1)$ is a random variable drawn from a uniform distribution, and Ξ^d denotes the exploration probability at time step t. The Ξ^d is initialized with a high value (e.g., $\Xi^d = 1.0$) to encourage extensive exploration of the action space and gradually decays (e.g., to $\Xi^d = 0.01$) over the course of training, shifting the policy from exploration toward exploitation. During inference or deployment, the multi-head DQN deterministically selects the action that maximizes the Q-value:

$$\mathcal{A}_t^d = \arg\max_{\mathcal{A}} Q(\mathcal{S}_t, \mathcal{A}; \theta_k^d). \tag{27}$$

2) TD3 for Continuous Resource Allocation

For continuous CPU resource allocations, the TD3 framework optimizes resource allocation decisions, such as CPU frequency assignments $f_{m,n}^{i,s}$ for computational tasks $x_{m,n}^i$, executed either locally or at edge servers. The TD3 framework, parameterized by an actor network ϕ_k and two critic networks $(\psi_{k,1}, \psi_{k,2})$, generates continuous actions $\mathcal{A}_t^c \in \mathcal{A}_k^c$ to minimize task execution delay and energy consumption while ensuring stable training through advanced techniques like clipped double Q-learning, delayed policy updates, and target networks. The TD3 architecture consists of:

- One Actor Network ($\mu(\mathcal{S}_t; \phi_k)$): Outputs deterministic continuous actions, such as CPU frequencies, for a given system state $\mathcal{S}_t$, parameterized by ϕ_k.
- Two Critic Networks ($Q_1(\mathcal{S}_t, \mathcal{A}_t^c; \psi_{k,1})$, $Q_2(\mathcal{S}_t, \mathcal{A}_t^c; \psi_{k,2})$): Independently estimate Q-values for state-action pairs, parameterized by $\psi_{k,1}$ and $\psi_{k,2}$, reducing overestimation bias.
- Target Networks ($\mu'(\mathcal{S}_t; \phi_k')$, $Q_1'(\mathcal{S}_t, \mathcal{A}_t^c; \psi_{k,1}')$, $Q_2'(\mathcal{S}_t, \mathcal{A}_t^c; \psi_{k,2}')$): Provide stable target values for training, parameterized by ϕ_k', $\psi_{k,1}'$, and $\psi_{k,2}'$.

During training, the actor network generates a continuous action with added exploration noise:

$$\mathcal{A}_t^c = \mu(\mathcal{S}_t; \phi_k) + \Xi_t, \quad \Xi_t \sim \mathcal{N}(0, \sigma^2), \tag{28}$$

where Ξ_t is Gaussian noise with variance σ^2. To ensure feasibility, actions are clipped within allowable resource limits:

$$\mathcal{A}_t^c = \text{clip}(\mathcal{A}_t^c, \mathcal{A}_{\min}^c, \mathcal{A}_{\max}^c). \tag{29}$$

The critic networks are trained by minimizing the MSE loss:

$$\mathcal{L}_{\text{TD3}}(\psi_{k,j}) = \frac{1}{U}\sum_{t=1}^{U}[y_t - Q_j(\mathcal{S}_t, \mathcal{A}_t^c; \psi_{k,j})]^2, \quad j \in \{1,2\}, \tag{30}$$

where the target y_t is computed using the target networks:

$$y_t = \mathcal{R}_t + \gamma \min_{j=1,2} Q_j'(\mathcal{S}_{t+1}, \mathcal{A}'; \psi_{k,j}'), \tag{31}$$

Algorithm 1: FEDORA Training Phase

```
   Input: $\mathbb{M}$, $\mathbb{N}_k$, $\mathbb{S}_k$, $\mathbb{D}_{k,n}$, $\mathbb{B}$, $\mathcal{B}$, $h$, $R$, $\gamma$, $\tau$, $\Xi$, $\Xi_t$, $\sigma^2$, $\bar{\sigma}^2$, $\mu$, $\mathcal{E}_k$
1  Initialize: $\mathbf{\Theta}^{(0)} = (\Theta^d, \Psi_1, \Psi_2, \Phi)$, $\mathbf{\Theta}'^{(0)} = (\Theta^{d-}, \Psi_1', \Psi_2', \Phi')$
     $\boldsymbol{\theta}_k^{(0)} = (\theta_k^d, \psi_{k,1}, \psi_{k,2}, \phi_k)$, $\mathbb{B}_k$ $\forall k \in \mathbb{M}$
2  for $r = 1$ to $R$ do
3      for each $k \in \mathbb{M}$ in parallel do
4          Synchronize: $\boldsymbol{\theta}_k^{(r)} \leftarrow \mathbf{\Theta}^{(r-1)}$
5          for $e = 1$ to $\mathcal{E}_k$ do
6              Sample initial state $\mathcal{S}_0$ randomly from $\mathbb{S}_k$
7              for $t = 0$ to $\max_{n \in \mathbb{N}_k}(|X_{k,n}|) - 1$ do
8                  if $|\mathbb{B}_k| < U$ OR system constraints not satisfied
                    then
                       // Random exploration
9                      Select $\mathcal{A}_t^d \in \mathcal{A}_k^d$ randomly
10                     Select $\mathcal{A}_t^c \in \mathcal{A}_k^c$ randomly within
                        $(\mathcal{A}_{\min}^c, \mathcal{A}_{\max}^c)$
11                 else
                       // Exploration-exploitation
                       strategy
12                     Select $\mathcal{A}_t^d$ using Eq. (27)
13                     Select $\mathcal{A}_t^c$ using Eq. (29)
14                 Execute action $\mathcal{A}_t = (\mathcal{A}_t^d, \mathcal{A}_t^c)$, observe reward
                    $\mathcal{R}_{t+1}$ using Eq. (21), next state $\mathcal{S}_{t+1}$
15                 Store $(\mathcal{S}_t, \mathcal{A}_t, \mathcal{R}_{t+1}, \mathcal{S}_{t+1})$ in $\mathbb{B}_k$
16                 if $|\mathbb{B}_k| \geq U$ AND system constraints satisfied
                    then
                       // Training step with FedProx
                       regularization
17                     Sample mini-batch of $B$ transitions
                        $\{(\mathcal{S}_i, \mathcal{A}_i, \mathcal{R}_{i+1}, \mathcal{S}_{i+1})\}_{i=1}^B$ from $\mathbb{B}_k$
18                     Compute DQN target using Eq. (24)
19                     Compute DQN loss using Eq. (25)
20                     Compute TD3 target using Eq. (31)
21                     Compute TD3 critic loss using Eq. (30)
22                     Compute total loss with FedProx using
                        Eq. (35), Eq. (36)
23                     Update $\boldsymbol{\theta}_k = (\theta_k^d, \psi_{k,1}, \psi_{k,2}, \phi_k)$ by
                        minimizing $\mathcal{L}_k(\boldsymbol{\theta}_k)$ using Adam
24                     if $t \mod \varphi = 0$ then
25                         Update actor using Eq. (32)
26                         Update target networks using Eq. (23),
                            Eq. (34)
27      Select participating agents $\mathbb{M}_r \subseteq \mathbb{M}$
28      Aggregate at $\mathbb{A}$: $\mathbf{\Theta}^{(r)} = \sum_{k \in \mathbb{M}_r} \frac{n_k}{\sum_{k' \in \mathbb{M}_r} n_{k'}} \boldsymbol{\theta}_k^{(r)}$
29      Broadcast $\mathbf{\Theta}^{(r)}$ to all agents $k \in \mathbb{M}$
   Output: $\boldsymbol{\theta}_k = (\theta_k^d, \psi_{k,1}, \psi_{k,2}, \phi_k)$ $\forall k \in \mathbb{M}$
```

and $\mathcal{A}' = \mu'(\mathcal{S}_{t+1}; \phi_k') + \xi$, with $\xi \sim \mathcal{N}(0, \bar{\sigma}^2)$ representing the Gaussian noise added to the target actor's action for smoothed exploration. To stabilize training, TD3 employs delayed policy updates, updating the actor network parameters ϕ_k less frequently (e.g., every φ critic updates, where φ is a hyperparameter). The actor is updated using the policy gradient to maximize the Q-value from the first critic:

$$\nabla_{\phi_k} J(\phi_k) = \frac{1}{U}\sum_{t=1}^{U} \nabla_{\mathcal{A}} Q_1(\mathcal{S}_t, \mathcal{A}; \psi_{k,1})|_{\mathcal{A}=\mu(\mathcal{S}_t;\phi_k)} \cdot \nabla_{\phi_k}\mu(\mathcal{S}_t; \phi_k). \tag{32}$$

The target networks are updated softly to ensure gradual and stable training dynamics:

$$\psi_{k,j}' \leftarrow \tau_c \psi_{k,j} + (1-\tau_c)\psi_{k,j}', \quad j \in \{1,2\}, \tag{33}$$

$$\phi_k' \leftarrow \tau_c \phi_k + (1-\tau_c)\phi_k', \tag{34}$$

where $\tau_c \ll 1$ is the soft update rate. During inference, the TD3 agent deterministically selects actions $\mathcal{A}_t^c = \mu(\mathcal{S}_t; \phi_k)$, ensuring optimal CPU resource allocations without exploration noise. The local training combines both losses into a single local objective:

$$\mathcal{L}_k(\boldsymbol{\theta}_k) = \mathcal{L}_{\text{DQN}}(\theta_k^d) + \mathcal{L}_{\text{TD3}}(\psi_{k,1}, \psi_{k,2}, \phi_k). \tag{35}$$

To ensure convergence stability under statistical heterogeneity and to control client drift, we incorporate FedProx regularization into the local training objective:

$$\boldsymbol{\theta}_k^{(r)} = \arg\min_{\boldsymbol{\theta}_k} \left[\mathcal{L}_k(\boldsymbol{\theta}_k) + \frac{\mu}{2}\|\boldsymbol{\theta}_k - \boldsymbol{\Theta}^{(r-1)}\|_2^2\right], \tag{36}$$

where $\boldsymbol{\Theta}^{(r-1)}$ represents the global model parameters from the previous aggregation round $r \in R$, and $\mu > 0$ is the proximal coefficient controlling the trade-off between local training and global model consistency. A small value of the proximal coefficient μ allows local models to prioritize their own data, which may improve local performance but increases the risk of divergence in non-IID settings. In contrast, a large μ enforces closer alignment with the global model, enhancing consistency and convergence but potentially limiting the ability of local models to adapt to unique data characteristics.

C. Global Aggregation and Model Fusion

To address system and data heterogeneity across agents in the MEC framework, we employ FedProx for global aggregation and model fusion. After each agent $k \in \mathbb{M}$ completes $\mathcal{E}_k$ local training epochs, it transmits its updated local parameters $\boldsymbol{\theta}_k^{(r+1)} = (\theta_k^d, \psi_{k,1}, \psi_{k,2}, \phi_k)$ to the global aggregator $\mathbb{A}$. Unlike standard local training, FedProx modifies the local objective to include a proximal term that keeps local parameters close to the global model, enhancing stability in non-IID workloads:

$$\min_{\boldsymbol{\theta}_k} \left[\mathcal{L}_k(\boldsymbol{\theta}_k) + \frac{\mu}{2}\|\boldsymbol{\theta}_k - \boldsymbol{\Theta}^{(r)}\|_2^2\right], \tag{37}$$

where $\mathcal{L}_k(\boldsymbol{\theta}_k) = \mathcal{L}_{\text{DQN}}(\theta_k^d) + \mathcal{L}_{\text{TD3}}(\psi_{k,1}, \psi_{k,2}, \phi_k)$ is the local loss, $\boldsymbol{\Theta}^{(t)}$ is the global model at round r. The number of local epochs $\mathcal{E}_k$ may vary across agents due to computational heterogeneity, and FedProx accommodates partial participation by allowing a subset of agents to contribute in each round. The aggregator performs a federated aggregation step to compute the new global model parameters $\boldsymbol{\Theta}^{(t+1)}$:

$$\boldsymbol{\Theta}^{(r+1)} = \sum_{k \in \mathbb{M}_t} \frac{n_k}{\sum_{k' \in \mathbb{M}_r} n_{k'}} \boldsymbol{\theta}_k^{(r+1)}, \tag{38}$$

where $\mathbb{M}_r \subseteq \mathbb{M}$ is the subset of agents participating in round r, and n_k is a weighting factor proportional to the number of transitions in agent k's experience replay buffer or a priority factor based on task criticality in the MEC system. This aggregation step mirrors the weighted averaging of FedAvg [38] but is applied to the heterogeneous updates produced by the proximal term modified local training. The aggregator broadcasts the updated global model $\boldsymbol{\Theta}^{(r+1)}$ back to all agents, which update their local parameters:

$$\boldsymbol{\theta}_k \leftarrow \boldsymbol{\Theta}^{(r+1)}, \quad \forall k \in \mathbb{M}. \tag{39}$$

Each agent then resumes local training using the aggregated parameters, incorporating the proximal term to ensure alignment with the global model. The cycle of local training, global aggregation, and parameter distribution repeats for R federated rounds or until convergence. FEDORA employs a robust global aggregation mechanism based on FedProx, which introduces a proximal regularization term in the local training phase. This term penalizes significant deviations of local models from the global model, thereby mitigating the risk of client drift caused by heterogeneous task dependencies and non-IID data distributions. Specifically, the global model aggregation follows a weighted averaging scheme, where the local model parameters are combined proportionally to the quantity and quality of the experiences gathered by each agent. Such an approach effectively balances the heterogeneous contributions from diverse DAG structures, preserving performance by limiting drastic divergence among local models. Consequently, FEDORA ensures stable convergence and consistently high performance across different DAG types. The complete training process of FEDORA is outlined in Algorithm 1.

D. Convergence Analysis:

To analyze the convergence behavior of FEDORA, we consider its core components: Multi-head DQN for discrete offloading decisions, TD3 for continuous resource allocation, and FedProx for stabilizing federated updates. The Q-value update in Multi-head DQN follows the Bellman expectation equation: $Q(\mathcal{S}_t, \mathcal{A}_t) \leftarrow \mathbb{E}\left[\mathcal{R}_t + \gamma \max_{\mathcal{A}_{t+1}} Q(\mathcal{S}_{t+1}, \mathcal{A}_{t+1})\right]$, which is known to be a γ-contraction in the sup-norm [37], ensuring convergence to a unique fixed point Q^* under stable target updates and sufficient exploration. For the TD3 module, convergence toward a locally optimal deterministic policy $\mu(\mathcal{S}_t)$ is supported by the deterministic policy gradient theorem: $\nabla_\theta J(\mu_\theta) = \mathbb{E}_{\mathcal{S}_t \sim \mathcal{D}}\left[\nabla_{\mathcal{A}_t} Q(\mathcal{S}_t, \mathcal{A}_t) \nabla_\theta \mu_\theta(\mathcal{S}_t)\right]$, with clipped double Q-learning and delayed updates enhancing stability. To address client drift and statistical heterogeneity in federated settings, FedProx modifies the local optimization at each agent as $\min_{\theta_k}\left[\mathcal{L}_k(\theta_k) + \frac{\mu}{2}\|\theta_k - \Theta^{(r)}\|^2\right]$, constraining local updates and promoting convergence. These components collectively ensure that FEDORA converges reliably to an effective hybrid offloading and resource allocation policy across distributed, heterogeneous MEC agents.

V. Results and Discussion

We emulate an FL environment where multiple edge devices collaborate to train a global model while preserving data privacy. The system consists of $\mathbb{M} = 6$ SBSs and a total of $|\mathbb{N}_m| = 60$ user devices, where each SBS contains $|\mathbb{S}| = 3$ edge servers and 10 user devices. Each device is responsible for executing computational tasks represented as DAGs. The tasks are offloaded dynamically based on network conditions, resource availability, and learning-based decision-making strategies. Unlike traditional single process simulations, our setup leverages multiple microservices deployed in Docker containers, where each container represents a separate MEC site or user agent. The containers are managed using

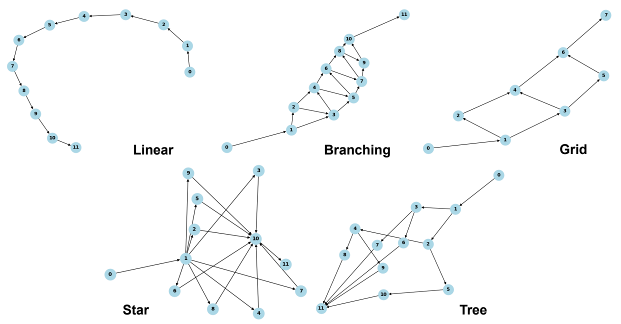


Fig. 3: Type of DAGs used for evaluation.

a Docker Compose setup, which includes the FL aggregator running in a dedicated container and handling global model aggregation using a FastAPI backend. Each participating agent runs in a separate container and is assigned a specific DAG topology. The entire system was deployed in a custom MEC emulation testbed at ÉTS to simulate realistic FL and task offloading environments. The DAG structures illustrated in Fig. 3 include:

- Linear: Sequential execution of tasks where each task depends only on the previous one.
- Branching: Tasks have multiple dependencies, allowing for parallel execution.
- Grid: A structured DAG where tasks are arranged in a 2D grid with interdependencies.
- Star: A central task connects to multiple independent tasks.
- Tree: A hierarchical task structure with branching dependencies.
- Mixed: A hybrid of the above structures to ensure generalization.

TABLE III: Simulation and Training Parameters

Parameter	Description
Federated rounds	90
Local training episodes per round	100
Tasks per DAG	Uniformly distributed: 5 to 30 tasks
Task data sizes ($o^i_{m,n}$)	Randomly assigned: 5 KB to 300 KB
Task computational workload ($c^i_{m,n}$)	Uniform distribution: 10^6 to 10^8 Hz
Device transmission power ($P^u_{m,n}$)	200 mW
Edge server transmission power ($P^d_{s\,n}$)	1 W
CPU frequency (user devices, $F^{m,0}_{\max}$)	1 GHz
CPU frequency (edge server, $F^{m,s}_{\max}$)	2.4 GHz
Channel gain ($\mathcal{G}_{s\,n}$, $\mathcal{H}_{n\,s}$)	Dynamic: -5 dB to -70 dB
DQN structure	3 fully connected layers (256 neurons each, ReLU)
DQN learning rate	0.0001 (Adam optimizer)
TD3 structure (actor and critic)	3 hidden layers (512, 256, 256 neurons)
TD3 learning rate	0.00001 (Adam optimizer)
Experience replay buffer size ($\vert\mathbb{B}\vert$)	50000
Batch size U	256
Soft update parameter (φ)	0.005
CPU	Ampere Altra (80 cores, 256 GB RAM)
GPU	NVIDIA RTX 6000 Ada Generation (48 GB memory)

In our setup, each SBS is assigned a dominant DAG type comprising 70 % of its local dataset, while the remaining 30% is a mix of other DAG structures. This setup reflects realistic non-IID distributions across MEC nodes and ensures both workload diversity and robustness testing for the federated DRL framework. Each DAG agent container or FL client is assigned an amount of NVIDIA GPU using Docker's GPU resource allocation for efficient training. The complete details of the FL setup, including the training rounds, neural network architectures, learning rates, and hardware specifics, are summarized in Table III. Task parameters, such as the number of tasks per DAG instance, data sizes, computational workloads, and wireless channel dynamics, are also detailed in the same table. These values are representative of typical settings of task offloading in MEC literature [3], [18].

To evaluate the performance of the proposed FEDORA framework, we compare it against a diverse set of baselines, grouped into three main categories: i) federated optimization, ii) FRL, and iii) heuristic baselines. These methods provide a comprehensive view of how different aggregation and learning strategies impact system performance:

i) Federated Optimization Baselines

- **FedAvg** [38]: Performs simple averaging of local model weights. It assumes IID data across clients and does not correct for drift in heterogeneous settings.
- **FedNova** [39]: Normalizes local updates based on the number of local training steps, mitigating objective inconsistency in non-IID data distributions.
- **SCAFFOLD** [40]: Incorporates control variates to address client drift and improve convergence under statistical heterogeneity.

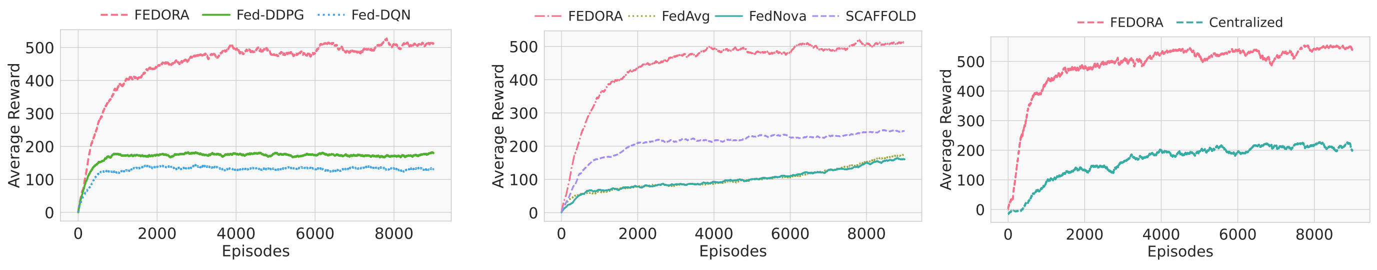
ii) FRL Baselines

- **FL-DQN** [41]: A federated version of DQN, where clients train Q-value functions locally and synchronize periodically using federated averaging.
- **FL-DDPG** [42]: Applies the DDPG actor-critic algorithm in a federated manner, suitable for continuous action spaces like CPU resource allocation. Both actor and critic networks are trained and aggregated across clients.

iii) Heuristic Baselines

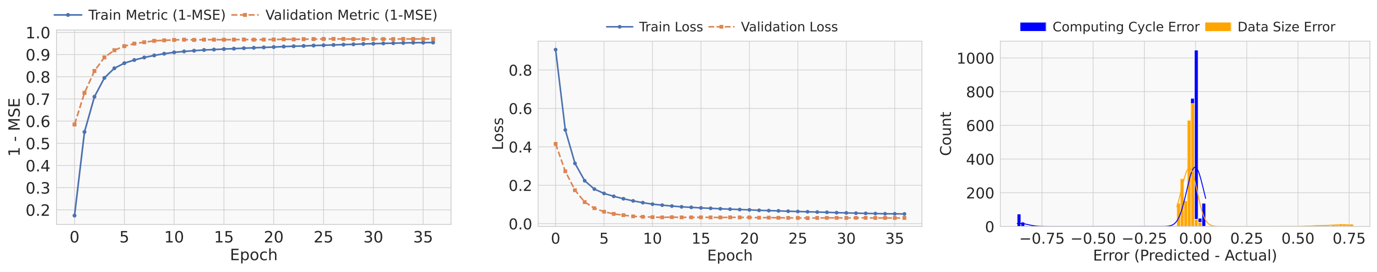
- **ALE** [17], [43], [44]: All local execution, where all the task are executed at the local device.
- **AEE** [17],[34]: All edge execution, where all tasks are offloaded to edge servers, regardless of system state.
- **ARE** [44], [34]: All random execution, where offloading and resource decisions are made randomly, without learning or task awareness.

We selected FL-DQN and FL-DDPG as baselines, as they are well-suited for handling discrete and continuous action spaces, respectively,This choice enables a clear and fair evaluation of FEDORA's effectiveness in jointly handling task offloading and resource allocation through its hybrid DRL approach. The initial set of experiments presented in Fig. 4 illustrates the training performance of FEDORA in comparison to existing DRL, federated optimization baselines, and centralized methods, measured by the average reward obtained during training episodes. The centralized methods concern training without the distributed framework of FL. Fig. 4a highlights the performance comparison among the proposed FEDORA, Fed-DQN, and Fed-DDPG methods. FEDORA exhibits rapid convergence, achieving substantial improvement within the first 2,000 training episodes, and continues to enhance performance thereafter, ultimately reaching an average reward close to 500. In contrast, Fed-DDPG and Fed-DQN methods converge to significantly lower performance levels, stabilizing at average



(a) Average reward analysis. (b) Average reward analysis. (c) Federated versus Centralized Learning.

Fig. 4: Benchmarking training performance of FEDORA against FRL, FL, and Centralized methods.



(a) GAT model accuracy analysis. (b) GAT model loss analysis. (c) GAT model feature prediction analysis.

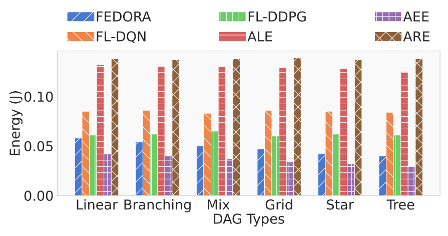
Fig. 5: Analyzing FEDORA's GAT model performance for task dependency learning.

rewards of approximately 180 and 140, respectively. These results demonstrate that FEDORA can learn more effective policies more efficiently compared to these baseline methods. The advantage of FEDORA arises primarily from its ability to handle the hybrid action space. Specifically, while FL-DQN can handle discrete actions, it struggles with continuous actions. Conversely, FL-DDPG addresses continuous actions but is ineffective with discrete ones. FEDORA integrates a multi-head DQN to efficiently handle the extensive discrete action space and leverages TD3 to manage continuous actions. This combined approach provides FEDORA with a significant advantage in terms of overall reward performance.

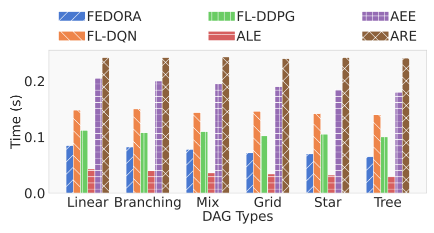
Fig. 4b, compares FEDORA against FL algorithms, including FedAvg, FedNova, SCAFFOLD. Among these methods, FEDORA consistently outperforms all FL. It achieves the highest average reward across training episodes, showcasing superior robustness in FRL scenarios. SCAFFOLD, while performing better than FedAvg and FedNova with an average reward around 250, still lags behind FEDORA in both reward and convergence consistency. FedNova, although designed to mitigate client drift through normalization techniques, exhibits slow convergence and achieves an average reward below 180 similar to FedAvg, which struggles significantly under non-IID conditions, a characteristic of our scenario. FEDORA notably outperforms SCAFFOLD in the FRL context, attributed to its robustness against client heterogeneity. The proximal term in FEDORA effectively regularizes local updates, enhancing training stability despite non-IID data and diverse computational capabilities. Conversely, SCAFFOLD's dependence on accurate control variates proves less effective under highly dynamic environments and resource constraints, resulting in less consistent convergence.

Fig. 4c contrasts FEDORA with a traditional centralized learning approach, often perceived as ideal due to its access to global information. In practice, however, centralized methods face significant challenges, including an excessively large state space due to the aggregation of information from all users. This complexity makes the centralized learning process cumbersome and less effective. In contrast, FEDORA dynamically assigns tasks between local and edge resources based on instantaneous system conditions, effectively optimizing computational and communication efficiency. This adaptive strategy results in superior learning performance compared to the centralized baseline, emphasizing FEDORA's practical advantages and robustness. FEDORA's superior performance is largely attributed to its capability to efficiently manage the challenges posed by non-IID data and partial observability across agents. Unlike traditional federated optimization or standard federated reinforcement learning methods, FEDORA promotes effective knowledge transfer among clients, accelerating policy convergence and achieving higher long-term rewards, particularly in highly heterogeneous environments.

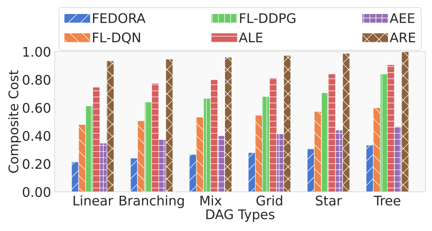
Fig. 5 demonstrates the training dynamics and predictive performance of the GAT model, designed to learn task features and structural dependencies of the DAGs. The model was trained over 35 epochs, and its performance was evaluated on a test set of 1000 samples. The results are summarized through training and validation loss as scatter plots of actual versus predicted values and error distributions, providing insights into the model's learning behavior and accuracy. Fig. 5a illustrates the training and validation accuracy over the 35 epochs. The training accuracy rises from 0.2 to 0.95, while the validation accuracy (dashed orange line) increases from 0.5 to 0.94, both plateauing close to 0.9 after 20 epochs. This smooth,



(a) Energy consumption analysis.

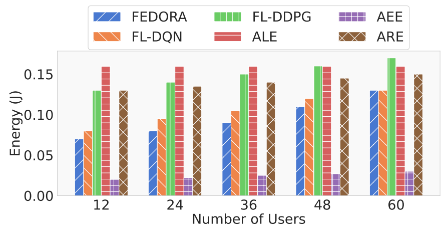


(b) Task completion time analysis.

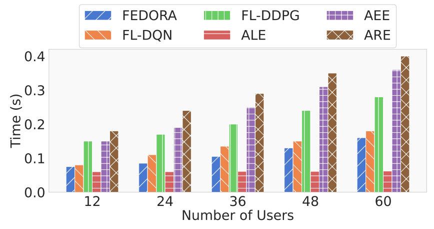


(c) Overall system cost analysis.

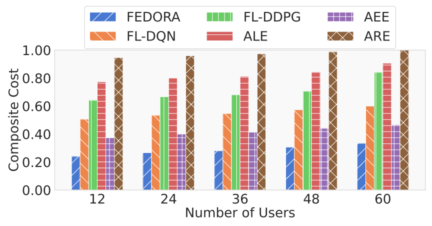
Fig. 6: Benchmarking FEDORA's performance across various DAG topologies.



(a) Energy consumption analysis.

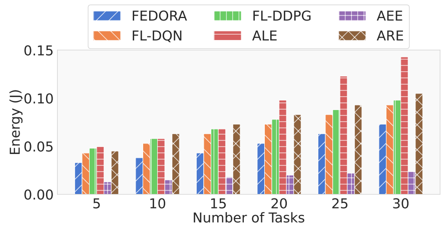


(b) Task completion time analysis.

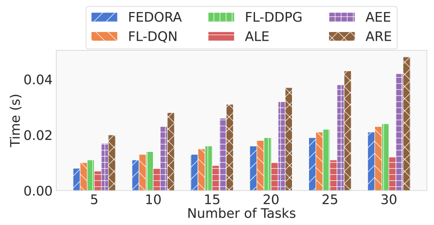


(c) Overall system cost analysis.

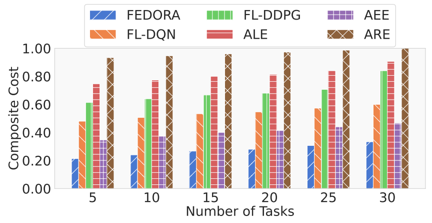
Fig. 7: Benchmarking FEDORA's performance across varying number of users.



(a) Energy performance.



(b) Task completion time performance.



(c) Overall system cost.

Fig. 8: Benchmarking FEDORA's performance across varying number of tasks.

continuous rise in accuracy, coupled with the decreasing loss and confirms the GAT model's effective learning of task features and dependencies. The small gap between the training and validation accuracy further indicates robust generalization, a critical factor for practical deployment.

Fig. 5b shows the training and validation loss of the GAT model over 35 epochs. The training loss decreases from 0.8 to below 0.05, while the validation loss follows a similar trend, starting at 0.4 and converging to around 0.05. The continual decrease in both losses indicates successful convergence, with the gradient descent algorithm effectively minimizing the loss function. The close alignment of the training and validation loss curves suggests minimal overfitting, highlighting the model's ability to generalize well to unseen DAGs.

Figure 5c depicts the error distributions for features predictions, calculated as error $=$ predicted $-$ actual. The two predicted features computing circle (CPU cycles required to execute a task) and data size (the amount of data processed) represent key structural and resource related attributes of a DAG. Both distributions are approximately normal, centered near zero, with most errors within $[-0.25, 0.25]$. The computing cycle errors (blue line) exhibit a narrower spread compared to data size (orange line), which shows slightly larger variance with errors extending to $[-0.75, 0.75]$. These two features are important as they directly influence task placement and resource allocation. Accurate prediction ensures that the model can generalize effectively to unseen DAGs. The symmetry of both distributions indicates no systematic bias, while the small error magnitudes confirm the model's predictive reliability for both attributes.

To investigate FEDORA's adaptability across diverse workflow structures, in Fig. 6 we evaluate its performance on the various DAG types. Fig. 6a presents the energy consumption across all DAG types. ALE and ARE incur the highest energy usage due to either exhaustive local computation or uncoordinated execution, respectively. AEE is energy efficient but at the expense of increased delay. FEDORA achieves a balanced energy profile, outperforming FL-DQN, FL-DDPG, and the random or static baselines across all DAG types. FL-DQN works well in task placement however, for resource allocation, which is a continuous action, FL-DQN struggles. Similarly, the FL-DDPG works well in a continuous action space, however, in task placement decisions it struggles. FEDORA, which handles both the discrete and continuous actions in efficient manner, reduces the energy consumption of the system.

The latency analysis, shown in Fig. 6b, highlights FEDORA's lower execution delays against all baselines except for ALE. The ALE strategy does not introduce a transmission delay, thus it incurs a lower delay at the expense of a high energy consumption. While ARE and AEE consistently suffer from latency due to the edge transmission overhead delay, FEDORA maintains short completion times through parallelism-aware task placement and dynamic system state adaptation. To capture the trade-off between energy consumption and latency, while accounting for constraint violations, we define the normalize composite cost as: composite cost $= (\alpha E_{avg} + (1-\alpha)T_{avg} + \mathcal{V}_{\text{tot}})$ where E_{avg} and T_{avg} are the average energy consumption and latency per device, $\alpha = 0.5$ balances the energy-latency trade-off. The term $\mathcal{V}_{\text{tot}} = \mathcal{V}_r + \mathcal{V}_e + \mathcal{V}_d$ introduced in the composite cost differ from the penalties previously described in Equation (21). Here, they represent counts of constraint violations for resource, energy and delay rather than penalty magnitudes. The composite cost results presented in Fig. 6c reinforce these observations; FEDORA consistently yields the lowest cost, indicating optimal trade-offs regardless of DAG structure. Even in complex DAGs like Grid or Tree, where task dependencies and multiple paths introduced scheduling challenges, FEDORA remains both energy-aware and latency efficient with lower system cost. This experiment confirms that FEDORA generalizes effectively across a wide range of DAG topologies, making it a robust and topology agnostic solution for MEC environments.

To evaluate the impact of the number of users per site on the performance, Fig. 7 illustrates the evaluation of the considered methods when varying the number of users from 12 to 60. Fig. 7a illustrates the average energy consumption across different user densities. As expected, AEE consistently demonstrates the lowest energy usage by offloading all tasks to the edge servers. In contrast, ALE incurs the highest energy cost due to heavy reliance on local computation. Our proposed FEDORA framework achieves a well-balanced energy profile, outperforming FL-DDPG and FL-DQN, and substantially surpassing ALE and ARE. This efficiency is due to FEDORA's hybrid task offloading strategy, which dynamically adjusts decisions based on local energy conditions and network congestion. FEDORA, by integrating a hybrid actor-critic architecture, is explicitly designed to handle both discrete and continuous actions. This makes FEDORA more adaptable and expressive in complex MEC environments.

Latency results, depicted in Fig. 7b, further support this observation. While ALE achieves low delay due to zero communication overhead, both AEE and ARE show increasing delays with higher user numbers. FEDORA maintains low completion times throughout, due to its efficient offloading and adaptive scheduling, outperforming both FL-DQN and FL-DDPG. Fig. 7c presents the composite cost results. Across all settings, FEDORA achieves the lowest overall cost, validating its effectiveness in jointly optimizing latency and energy under FL constraints.

In Fig. 8, we further evaluate FEDORA's performance against the baseline methods under varying number of tasks (from 5 to 30) to assess its robustness and efficiency in MEC environments. As illustrated in Fig. 8a, energy consumption trends reveal each method's adaptability to workload intensification. ALE shows a sharp increase in energy usage due to its on-device execution strategy, which becomes unsustainable as task volume grows. In contrast, AEE maintains consistently low energy consumption by aggressively offloading all tasks to the edge. FEDORA strikes an effective balance between these extremes, consuming as well significantly less energy than FL-DDPG and FL-DQN, while also outperforming ALE and ARE. This highlights FEDORA's adaptive offloading mechanism, which dynamically responds to rising workloads by considering both device energy states and offloading opportunities.

Latency performance, shown in Fig. 8b, further validates FEDORA's efficiency. While ARE and AEE suffer from increased delays due to server congestion and inefficient task scheduling, FEDORA sustains low task completion times through efficient decision-making. Compared to FL-DDPG and FL-DQN, FEDORA's hybrid control strategy enables more effective resource management under high task loads. The overall system efficiency, measured by the composite cost metric in Fig. 8c, confirms that FEDORA consistently achieves the lowest total cost across all task load levels. This underscores FEDORA's capability to jointly minimize latency and energy consumption, demonstrating its scalability and effectiveness in handling increasing workload intensity. In summary, this task-based evaluation shows that FEDORA delivers high performance not only with growing user density but also under escalating task load. Its adaptability to both dimensions of system load underscores its practical viability and robustness for real-time federated MEC applications. However, significant increase in the number of devices and the complexity of DAGs poses significant challenges in terms of computational and training resource requirements.

VI. Conclusion

In this paper, we introduced FEDORA, a federated ensemble reinforcement learning framework for DAG-based task offloading and resource allocation in MEC environments. Recognizing the limitations of traditional centralized approaches for practical large-scale and dynamic IoT scenarios, we reformulated the problem into an MDP, enabling an efficient solution through reinforcement learning. Our proposed FEDORA framework leverages FL and GAT to effectively capture complex task interdependencies and dynamic network states without compromising user privacy or introducing excessive communication overhead. Extensive simulation results demonstrate that FEDORA significantly enhances energy efficiency, reduces task completion latency, and substantially improves QoS, outperforming traditional methods across various dynamic scenarios. Furthermore, FEDORA exhibits rapid convergence, robust scalability, and minimal communication overhead, making it particularly well suited for practical deployment in resource-constrained and IoT systems. Future work will adapt FEDORA to support highly mobile IoT platforms such as drones and connected vehicles integrating it with forthcoming 6G continuums to meet the stringent demands of both ultra-reliable low-latency communications (URLLC) and enhanced mobile broadband (eMBB) services.

References

[1] H. Zhou, Z. Zhang, Y. Wu, M. Dong, and V. C. Leung, "Energy efficient joint computation offloading and service caching for mobile edge computing: A deep reinforcement learning approach," *IEEE Trans. Green Commun. Networking*, vol. 7, no. 2, pp. 950–961, 2022.

[2] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Commun. Surv. Tutorials*, vol. 19, no. 3, pp. 1628–1656, 2017.

[3] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Commun. Surv. Tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017.

[4] F. Saeik *et al.*, "Task offloading in Edge and Cloud Computing: A survey on mathematical, artificial intelligence and control theory solutions," *Computer Networks*, vol. 195, p. 108177, 2021.

[5] Z. Cao, X. Deng, S. Yue, P. Jiang, J. Ren, and J. Gui, "Dependent Task Offloading in Edge Computing Using GNN and Deep Reinforcement Learning," *IEEE Internet of Things Journal*, vol. 11, no. 12, pp. 21 632–21 646, 2024.

[6] X. Chen and G. Liu, "Energy-efficient task offloading and resource allocation via deep reinforcement learning for augmented reality in mobile edge networks," *IEEE Internet Things J.*, vol. 8, no. 13, pp. 10 843–10 856, 2021.

[7] H. Jiang, X. Dai, Z. Xiao, and A. Iyengar, "Joint task offloading and resource allocation for energy-constrained mobile edge computing," *IEEE Trans. Mob. Comput.*, vol. 22, no. 7, pp. 4000–4015, 2022.

[8] B. Wang, C. Wang, W. Huang, Y. Song, and X. Qin, "A survey and taxonomy on task offloading for edge-cloud computing," *IEEE Access*, vol. 8, pp. 186 080–186 101, 2020.

[9] X. Wang, C. Wang, X. Li, V. C. Leung, and T. Taleb, "Federated deep reinforcement learning for Internet of Things with decentralized cooperative edge caching," *IEEE Internet Things J.*, vol. 7, no. 10, pp. 9441–9455, 2020.

[10] J. Liu, J. Ren, Y. Zhang, X. Peng, Y. Zhang, and Y. Yang, "Efficient dependent task offloading for multiple applications in MEC-cloud system," *IEEE Trans. Mob. Comput.*, vol. 22, no. 4, pp. 2147–2162, 2021.

[11] A. Feriani and E. Hossain, "Single and multi-agent deep reinforcement learning for AI-enabled wireless networks: A tutorial," *IEEE Commun. Surv. Tutorials*, vol. 23, no. 2, pp. 1226–1252, 2021.

[12] Y.-C. Wu, T. Q. Dinh, Y. Fu, C. Lin, and T. Q. Quek, "A hybrid DQN and optimization approach for strategy and resource allocation in MEC networks," *IEEE Trans. Wireless Commun.*, vol. 20, no. 7, pp. 4282–4295, 2021.

[13] J. Konečnỳ *et al.*, "Federated learning: Strategies for improving communication efficiency," *arXiv preprint arXiv:1610.05492*, 2016.

[14] T. Zhao, F. Li, and L. He, "Secure video offloading in MEC-enabled IIoT networks: A multicell federated deep reinforcement learning approach," *IEEE Trans. Ind. Inf.*, vol. 20, no. 2, pp. 1618–1629, 2023.

[15] W. Fan *et al.*, "DNN deployment, task offloading, and resource allocation for joint task inference in IIoT," *IEEE Trans. Ind. Inf.*, vol. 19, no. 2, pp. 1634–1646, 2022.

[16] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.

[17] J. Wang, J. Hu, G. Min, W. Zhan, A. Y. Zomaya, and N. Georgalas, "Dependent task offloading for edge computing based on deep reinforcement learning," *IEEE Trans. Comput.*, vol. 71, no. 10, pp. 2449–2461, 2021.

[18] J. Li, B. Gu, Z. Qin, and Y. Han, "Graph tasks offloading and resource allocation in multi-access edge computing: A DRL-and-optimization-aided approach," *IEEE Trans. Network Sci. Eng.*, vol. 10, no. 6, pp. 3707–3718, 2023.

[19] S. Khan, M. Avgeris, J. Gascon-Samson, and A. Leivadeas, "EMD-TORA: Energy-aware multi-user dependent task offloading and resource allocation in MEC using graph-enabled DRL," *IEEE Trans. Green Commun. Networking*, 2024.

[20] S. E. Mahmoodi, R. Uma, and K. Subbalakshmi, "Optimal joint scheduling and cloud offloading for mobile applications," *IEEE Trans. Cloud Comput.*, vol. 7, no. 2, pp. 301–313, 2016.

[21] Y. Liu *et al.*, "Dependency-aware task scheduling in vehicular edge computing," *IEEE Internet Things J.*, vol. 7, no. 6, pp. 4961–4971, 2020.

[22] C. Xu *et al.*, "Energy consumption and time-delay optimization of dependency-aware tasks offloading for industry 5.0 applications," *IEEE Trans. Consum. Electron.*, vol. 70, no. 1, pp. 1590–1600, 2023.

[23] Liu, Jiagang and Ren, Ju and Zhang, Yongmin and Peng, Xuhong and Zhang, Yaoxue and Yang, Yuanyuan, "Efficient dependent task offloading for multiple applications in mec-cloud system," *IEEE Trans. Mob. Comput.*, vol. 22, no. 4, pp. 2147–2162, 2023.

[24] X. An, R. Fan, H. Hu, N. Zhang, S. Atapattu, and T. A. Tsiftsis, "Joint task offloading and resource allocation for IoT edge computing with sequential task dependency," *IEEE Internet Things J.*, vol. 9, no. 17, pp. 16 546–16 561, 2022.

[25] Z. Liu, M. Liwang, S. Hosseinalipour, H. Dai, Z. Gao, and L. Huang, "RFID: Towards low latency and reliable DAG task scheduling over dynamic vehicular clouds," *IEEE Trans. Veh. Technol.*, vol. 72, no. 9, pp. 12 139–12 153, 2023.

[26] Q. Liu, Z. Tian, N. Wang, and Y. Lin, "DRL-based dependent task offloading with delay-energy tradeoff in medical image edge computing," *Complex & Intelligent Systems*, vol. 10, no. 3, pp. 3283–3304, 2024.

[27] C. Feng, P. Han, X. Zhang, Q. Zhang, Y. Liu, and L. Guo, "Dependency-aware task reconfiguration and offloading in multi-access edge cloud networks," *IEEE Trans. Mob. Comput.*, vol. 23, no. 10, pp. 9271–9288, 2024.

[28] H. Xiao, Z. Hu, X. Zhang, A. Xu, M. Zheng, and K. Li, "Federated Deep Reinforcement Learning for Task Offloading in MEC-enabled Heterogeneous Networks," *IEEE Internet Things J.*, 2024.

[29] H. Wu, A. Gu, and Y. Liang, "Federated Reinforcement Learning-Empowered Task Offloading for Large Models in Vehicular Edge Computing," *IEEE Trans. Veh. Technol.*, 2024.

[30] X. Zhao, Y. Wu, T. Zhao, F. Wang, and M. Li, "Federated deep reinforcement learning for task offloading and resource allocation in mobile edge computing-assisted vehicular networks," *Journal of Network and Computer Applications*, vol. 229, p. 103941, 2024.

[31] H. Zhou, H. Wang, Z. Yu, G. Bin, M. Xiao, and J. Wu, "Federated distributed deep reinforcement learning for recommendation-enabled edge caching," *IIEEE Trans. Serv. Comput.*, 2024.

[32] S. Shen, G. Shen, Z. Dai, K. Zhang, X. Kong, and J. Li, "Asynchronous Federated Deep Reinforcement Learning-Based Dependency Task Offloading for UAV-Assisted Vehicular Networks," *IEEE Internet Things J.*, 2024.

[33] Z. Tong, J. Deng, J. Mei, Y. Zhang, and K. Li, "Multi-Objective DAG Task Offloading in MEC Environment Based on Federated DQN With Automated Hyperparameter Optimization," *IIEEE Trans. Serv. Comput.*, 2024.

[34] J. Chen, Y. Yang, C. Wang, H. Zhang, C. Qiu, and X. Wang, "Multitask offloading strategy optimization based on directed acyclic graphs for edge computing," *IEEE Internet Things J.*, vol. 9, no. 12, pp. 9367–9378, 2021.

[35] Y. Chen, W. Gu, and K. Li, "Dynamic task offloading for internet of things in mobile edge computing via deep reinforcement learning," *International Journal of Communication Systems*, p. e5154, 2022.

[36] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated optimization in heterogeneous networks," *Proceedings of Machine learning and systems*, vol. 2, pp. 429–450, 2020.

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

[38] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial intelligence and statistics*. PMLR, 2017, pp. 1273–1282.

[39] J. Wang, Q. Liu, H. Liang, G. Joshi, and H. V. Poor, "Tackling the objective inconsistency problem in heterogeneous federated optimization," *Advances in neural information processing systems*, vol. 33, pp. 7611–7623, 2020.

[40] S. P. Karimireddy, S. Kale, M. Mohri, S. Reddi, S. Stich, and A. T. Suresh, "Scaffold: Stochastic controlled averaging for federated learning," in *International conference on machine learning*. PMLR, 2020, pp. 5132–5143.

[41] M. Al-Naday, V. Dobre, M. Reed, S. Toor, B. Volckaert, and F. De Turck, "Federated deep Q-learning networks for service-based anomaly detection and classification in edge-to-cloud ecosystems," *Annals of Telecommunications*, vol. 79, no. 3, pp. 165–178, 2024.

[42] B. Ouyang, J. Li, and X. Chen, "DDPG-FL: A Reinforcement Learning Approach for Data Balancing in Federated Learning," in *China Conference on Networking*. Springer, 2023, pp. 33–47.

[43] G. Nieto, I. de la Iglesia, U. López-Novoa, and C. Perfecto, "Deep Reinforcement Learning-based Task Offloading in MEC for energy and resource-constrained devices," in *2023 IEEE International Mediterranean Conference on Communications and Networking (MeditCom)*. IEEE, 2023, pp. 127–132.

[44] H. Xiao, C. Xu, Y. Ma, S. Yang, L. Zhong, and G.-M. Muntean, "Edge Intelligence: A Computational Task Offloading Scheme for Dependent IoT Application," *IEEE Trans. on Wireless Communications*, vol. 21, no. 9, pp. 7222–7237, 2022.