Artificial Intelligence–Based Dependent Task Offloading in Next-Generation Edge Networks

by

Sangrez KHAN

MANUSCRIPT-BASED THESIS PRESENTED TO ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
IN PARTIAL FULFILLMENT FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
Ph.D.

MONTREAL, APRIL 30, 2026

ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

BOARD OF EXAMINERS

THIS THESIS HAS BEEN EVALUATED

BY THE FOLLOWING BOARD OF EXAMINERS

Mr. Aris Leivadeas, Thesis supervisor
Department of Software and Information Technology Engineering, École de technologie supérieure ÉTS

Mr. Julien Gascon-Samson, Thesis Co-Supervisor
Department of Software and Information Technology Engineering, École de technologie supérieure ÉTS

Mr. Pascal Giard, Chair, Board of Examiners
Department of Electrical Engineering, École de technologie supérieure ÉTS

Ms. Diala Naboulsi, Member of the Jury
Department of Software and Information Technology Engineering, École de technologie supérieure ÉTS

Mr. Rodolfo W. L. Coutinho, External Independent Examiner
Department of Electrical and Computer Engineering, Concordia University

THIS THESIS WAS PRESENTED AND DEFENDED

IN THE PRESENCE OF A BOARD OF EXAMINERS AND THE PUBLIC

ON "APRIL 20, 2026"

AT ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

ACKNOWLEDGEMENTS

This thesis represents the culmination of a profound academic journey, made possible by the unwavering support and visionary guidance of my supervisors, Prof. Aris Leivadeas and Prof. Julien Gascon-Samson. Beyond their exceptional academic mentorship, I am profoundly grateful for their generous financial support and unwavering understanding when I needed it most. Their belief in my potential provided the stability that allowed me to persevere and complete this work.

I extend my sincere appreciation to my esteemed collaborators, Dr. Marios Avgeris and Amir Ali-Pour, whose technical insights and teamwork were essential to the realization of the articles presented herein.

Finally, my deepest gratitude goes to my family for their endless patience, encouragement, and belief in me throughout this endeavor especially my brother, Jouhar Lawangeen, for his constant support, encouragement, and care throughout this journey. I also wish to acknowledge my friends Dr. Mudassir Rahim, for his guidance throughout my PhD journey, and Abdul Basit for his constant support.

Déchargement de tâches dépendantes fondé sur l'intelligence artificielle dans les réseaux périphériques de nouvelle génération

Sangrez KHAN

RÉSUMÉ

La croissance exponentielle de l'Internet des objets (IoT) et les exigences strictes en matière de latence des réseaux de cinquième génération (5G) et de future sixième génération (6G) stimulent l'évolution rapide du calcul en périphérie multi-accès (Multi-Access Edge Computing, MEC). Bien que le MEC rapproche les ressources de calcul des utilisateurs finaux afin d'alléger la charge du cœur de réseau, les applications mobiles modernes, telles que la réalité augmentée (AR), l'analyse vidéo en temps réel et la conduite autonome, ont évolué de requêtes simples et indépendantes vers des flux de travail complexes modélisés sous forme de graphes orientés acycliques (Directed Acyclic Graphs, DAG) avec des dépendances inter-tâches strictes. Les techniques d'optimisation traditionnelles et les méthodes heuristiques peinent à traiter la forte dimensionnalité, la complexité topologique et la nature dynamique de ces environnements hétérogènes. De plus, les solutions d'apprentissage existantes négligent fréquemment les dépendances structurelles entre les tâches ou ne tiennent pas compte de l'hétérogénéité statistique inhérente aux réseaux distribués en périphérie. Pour surmonter ces limitations, cette thèse exploite des techniques avancées d'intelligence artificielle (IA), notamment les réseaux neuronaux de graphes (Graph Neural Networks, GNN), l'apprentissage par renforcement profond (Deep Reinforcement Learning, DRL) et l'apprentissage fédéré (Federated Learning, FL), afin de permettre une orchestration intelligente et robuste des ressources. Dans ce contexte, la présente thèse étudie l'optimisation conjointe du déchargement de tâches dépendantes, de l'allocation des ressources et de la gestion de la mobilité afin d'améliorer l'efficacité, la fiabilité et la confidentialité du système. Plus précisément, nous proposons trois cadres fondés sur l'IA pour répondre à la complexité des applications basées sur des DAG sous des contraintes réseau réalistes.

Nous présentons d'abord un cadre Energy-Aware Multi-user Dependent Task Offloading and Resource Allocation (EMDTORA), qui intègre des réseaux d'attention sur graphes (Graph Attention Networks, GAT) à l'algorithme Twin Delayed Deep Deterministic Policy Gradient (TD3). De cette manière, le système apprend explicitement des représentations structurelles des DAG applicatifs. Cela permet à l'agent d'identifier des séquences d'exécution efficaces des tâches dépendantes, de borner strictement le temps d'achèvement des applications et d'en prioriser l'exécution. Les résultats de simulation démontrent que EMDTORA surpasse significativement les approches de référence en réduisant le coût énergie-temps pondéré (Energy-Time Cost, ETC).

Ensuite, nous abordons les défis critiques liés à la confidentialité des données et à l'hétérogénéité statistique dans les réseaux distribués. À cette fin, nous introduisons un cadre Federated ensemble reinforcement learning (FEDORA) pour le déchargement de tâches basées sur des graphes orientés acycliques (DAG) et l'allocation des ressources en MEC. Étant donné que l'entraînement centralisé conventionnel présente des risques importants pour la confidentialité et engendre

des surcoûts de communication élevés, FEDORA adopte une architecture d'apprentissage décentralisée dans laquelle les nœuds en périphérie entraînent des modèles locaux et ne partagent que des mises à jour de gradients. Afin d'atténuer la dérive des clients causée par la diversité des comportements des utilisateurs, nous utilisons l'algorithme d'agrégation FedProx, combiné à une nouvelle architecture DRL en ensemble qui dissocie les décisions discrètes de déchargement de l'allocation continue des ressources. Cette approche assure une convergence robuste et une bonne capacité de généralisation dans des environnements hétérogènes, sans nécessiter l'échange de données brutes.

Enfin, nous traitons la nature stochastique de la mobilité à grande vitesse dans le calcul en périphérie véhiculaire (Vehicular Edge Computing, VEC) en proposant un cadre Dependent Task Offloading in Vehicular Edge Computing Using Trajectory-Aware Deep Reinforcement Learning. Nous y intégrons un mécanisme conscient de la mobilité, alors que les schémas classiques, insuffisamment sensibles à la mobilité, entraînent souvent des taux élevés d'échec des tâches pour les applications sensibles aux délais. En exploitant la modélisation séquentielle basée sur les Transformers afin de prédire avec précision les fenêtres de connectivité des unités routières (Roadside Units, RSU), ce cadre permet à l'agent DRL de gérer proactivement l'exécution des tâches dépendantes. Cette capacité prédictive garantit que les tâches ne sont déchargées que lorsque leur achèvement fiable est assuré, et surpasse ainsi les méthodes traditionnelles en termes de délai et de consommation énergétique.

Mots-clés: Informatique en périphérie mobile, Apprentissage par renforcement profond, Réseaux de neurones graphiques, Apprentissage fédéré, Déchargement de tâches, 6G, Réseaux véhiculaires, Graphes acycliques dirigés

Artificial Intelligence–Based Dependent Task Offloading in Next-Generation Edge Networks

Sangrez KHAN

ABSTRACT

The exponential growth of the Internet of Things (IoT) and the stringent latency requirements of Fifth-Generation (5G) and future Sixth-Generation (6G) networks are driving the rapid evolution of Multi-Access Edge Computing (MEC). While MEC brings computational resources closer to end users to alleviate the burden on core networks, modern mobile applications such as Augmented Reality (AR), real-time video analytics, and autonomous driving have evolved from simple, independent requests into complex workflows modeled as Directed Acyclic Graphs (DAGs) with strict inter task dependencies. Traditional optimization techniques and heuristic methods often fail to address the high dimensionality, topological complexity, and dynamic nature of these heterogeneous environments. Furthermore, existing learning based solutions frequently neglect the structural dependencies of tasks or fail to account for the statistical heterogeneity inherent in distributed edge networks. To overcome these limitations, this thesis leverages advanced Artificial Intelligence (AI) techniques, including Graph Neural Networks (GNNs), Deep Reinforcement Learning (DRL), and Federated Learning (FL), to enable intelligent and robust resource orchestration. In this context, the present thesis investigates the joint optimization of dependent task offloading, resource allocation, and mobility management to enhance system efficiency, reliability, and privacy. Specifically, we propose three AI-driven frameworks to address the complexities of DAG based applications under realistic network constraints:

First we present an Energy-Aware Multi-user Dependent Task Offloading and Resource Allocation (EMDTORA) framework, which integrates Graph Attention Networks (GAT) with the Twin Delayed Deep Deterministic Policy Gradient (TD3) algorithm. This way, the system explicitly learns the structural embeddings of application DAGs. This allows the agent to identify efficient execution the sequence of dependent tasks that strictly bounds the application's completion time and prioritize their execution. Simulation results demonstrate that EMDTORA significantly outperforms standard baselines in reducing the weighted Energy-Time Cost (ETC). Following, we address the critical challenges of data privacy and statistical heterogeneity in distributed networks. Specifically, we introduce a Federated ensemble reinforcement learning (FEDORA) framework for directed acyclic graph (DAG)-based task Offloading and resource allocation in MEC. Since, conventional centralized training poses severe privacy risks and incurs high communication overheads, FEDORA employs a decentralized training architecture where edge nodes train local models and share only gradient updates. To mitigate client drift caused by diverse user behaviors, we utilize the FedProx aggregation algorithm combined with a novel Ensemble DRL architecture that decouples discrete offloading decisions from continuous resource allocation. This approach ensures robust convergence and generalization across heterogeneous environments without requiring raw data exchange. Finally, we tackle the stochastic nature of high speed mobility in Vehicular Edge Computing (VEC) by proposing a Dependent Task Offloading in Vehicular Edge Computing Using Trajectory-Aware Deep Reinforcement Learning and

incorporate a mobility-aware scheme which traditionally often leads to high task failure rates for delay sensitive applications. By leveraging Transformer-based sequence modeling to accurately predict Roadside Unit (RSU) connectivity windows, this framework enables the DRL agent to proactively manage dependent task execution. This predictive capability ensures that tasks are only offloaded when reliable completion is guaranteed, thereby outperforms traditional methods in terms of delay, and energy.

Keywords: Mobile Edge Computing, Deep Reinforcement Learning, Graph Neural Networks, Federated Learning, Task Offloading, 6G, Vehicular Networks, Directed Acyclic Graphs

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

Page

LIST OF ALGORITHMS

Page

LIST OF ABBREVIATIONS

5G	The fifth generation
6G	The sixth generation
AI	Artificial Intelligence
AP	Access point
AR	Augmented reality
AWGN	Additive white Gaussian noise
BS	Base station
BW	Bandwidth
C-V2X	Cellular vehicle-to-everything
CPU	Central processing unit
CSI	Channel state information
D2D	Device-to-device
DAG	Directed acyclic graph
DDPG	Deep deterministic policy gradient
DL	Deep learning
DNN	Deep neural network
DRL	Deep reinforcement learning
eMBB	Enhanced mobile broadband
EMDTORA	Energy-aware multi-user dependent task offloading

ETC	Energy-time cost
eURLLC	Extreme ultra-reliable low-latency communication
FedAvg	Federated averaging
FEDORA	Federated ensemble reinforcement learning
FedProx	Federated proximal
FeMBB	Further enhanced mobile broadband
FL	Federated learning
GAT	Graph attention network
Gbps	Gigabits per second
GenAI	Generative artificial intelligence
GHz	Gigahertz
GNN	Graph neural network
HIRS	Hybrid intelligent reconfigurable surface
IID	Independent and identically distributed
IoT	Internet-of-Things
IIoT	Industrial Internet-of-Things
IRS	Intelligent reconfigurable surface
ISAC	Integrated sensing and communication
ITS	Intelligent transport system
KPI	Key performance indicator

LLM	Large language model
LOS	Line-of-sight
LSTM	Long short-term memory
MAC	Medium access control
MARL	Multi-agent reinforcement learning
MEC	Multi-access edge computing
MIMO	Multiple-input-multiple-output
ML	Machine learning
mmWave	Millimeter wave
mMTC	Massive machine type communication
MSE	Mean squared error
NLOS	Non line-of-sight
NOMA	Non-orthogonal multiple access
Non-IID	Non-independent and identically distributed
NTN	Non-terrestrial network
QoS	Quality of service
RIS	Reconfigurable intelligent surface
RL	Reinforcement learning
RNN	Recurrent neural network
RSU	Roadside unit

SAGIN	Space-air-ground integrated network
SDN	Software defined network
SemCom	Semantic communication
SINR	Signal-to-interference-plus-noise ratio
SNR	Signal-to-noise ratio
Tbps	Terabits per second
TD3	Twin delayed deep deterministic policy gradient
THz	Terahertz
TIRS	Terrestrial intelligent reconfigurable surface
UAV	Unmanned ariel vehicle
UE	User equipment
URLLC	Ultra-reliable low-latency communication
V2E	Vehicle-to-everything
V2I	Vehicle-to-infrastructure
V2V	Vehicle-to-vehicle
V2X	Vehicle-to-everything
VEC	Vehicular edge computing
VR	Virtual reality

INTRODUCTION

0.1 Background and Motivation

The dawn of the fifth generation (5G) and the upcoming sixth generation (6G) networking eras has resulted a fundamental paradigm shift in how computational services are delivered. This evolution is not merely an incremental improvement in speed, but a structural transformation in the consumption of network and computational resources. The proliferation of the Internet of Things (IoT) has driven an exponential increase in connected devices, ranging from smart industrial sensors and wearable health monitors to autonomous vehicles and augmented reality (AR) headsets (Bhatia, Mallikarjuna, Gautam *et al.*, 2023).

As these devices evolve, the applications they support are transitioning from simple, atomic data transmissions to complex, computation intensive workflows. Modern applications, such as real time video analytics, cooperative autonomous driving, and interactive virtual reality (VR), are rarely monolithic (Kang, Li, Lin, Fan & Cai, 2024). Instead, they are composed of granular, interdependent sub tasks that must be executed in a strict topological order. These applications are mathematically modeled as Directed Acyclic Graphs (DAGs), where the output of one task serves as the mandatory input for subsequent tasks, creating rigorous precedence constraints (Sun, Theile, Qin *et al.*, 2024).

To meet stringent Quality of Service (QoS) requirements specifically ultra low latency and high reliability, Multi-Access Edge Computing (MEC) has emerged as a critical enabler. By decentralizing computational resources and deploying them at the edge of the Radio Access Network (RAN), MEC brings processing power closer to the data source. This significantly mitigates the propagation latency and backhaul congestion typically associated with centralized cloud computing (Huynh, Pham, Nguyen *et al.*, 2021).

However, orchestrating resources in next generation edge networks presents formidable challenges. The environment is highly dynamic, characterized by time varying wireless channels, stochastic user mobility, and heterogeneous device capabilities. Furthermore, the complexity of task dependencies in DAG based applications renders traditional resource management techniques obsolete. Simple heuristic algorithms and conventional convex optimization methods often fail to capture the intricate relationships between tasks, nor do they scale effectively in dense, multi-user scenarios (Huang, 2025; Shao, Qian, Li, Jiang & Jia, 2025; Fan, Ge, Zhang, Wu & Luo, 2023). Consequently, there is a pressing need for intelligent, adaptive, and scalable resource management frameworks capable of autonomously optimizing dependent task offloading.

To address these complexities, Deep Reinforcement Learning (DRL) has gained prominence as a robust tool for sequential decision-making (Mnih *et al.*, 2015). Yet, standard DRL approaches often treat task inputs as flat vectors, inherently losing vital structural information regarding task dependencies (Dai, Lyu, Cheng *et al.*, 2024). This thesis is motivated by the potential of integrating Graph Neural Networks (GNNs) (Scarselli, Gori, Tsoi, Hagenbuchner & Monfardini, 2008; Kipf & Welling, 2016) with DRL to create GNN agents capable of perceiving the topological structure of applications, thereby bridging the gap between theoretical optimization and practical, intelligent deployment in edge networks. While basic GNN-DRL agents have been explored for static scheduling (Dai *et al.*, 2024; Sun *et al.*, 2024), this thesis focus on advancing them for highly dynamic and decentralized edge environments by introducing federated learning and vehicular trajectory-awareness.

0.2 Problem Statement

Despite the extensive literature on computation offloading in MEC, several critical gaps remain that hinder the efficient deployment of complex IoT applications:

Complexity of Task Dependencies: The majority of existing offloading schemes assume that tasks are independent or follow simple sequential models. In reality, applications like video analytics or autonomous driving navigation consist of complex DAG structures (Peng, Zhang, Zhang, Zhang & Yang, 2024). Ignoring these dependencies leads to suboptimal scheduling, where tasks are offloaded without their necessary input data, causing execution stalls and increased latency (Huynh *et al.*, 2021). The problem of partitioning and scheduling DAG based applications across heterogeneous edge servers while optimizing for energy and delay is known to be NP-hard (Sun *et al.*, 2024).

Scalability and Privacy Concerns: While DRL has shown immense promise in solving optimization problems, centralized training approaches face significant deployment hurdles. Collecting raw task data from thousands of user devices at a central server incurs massive communication overhead and raises severe privacy concerns, as users are increasingly reluctant to share sensitive application data (McMahan, Moore, Ramage, Hampson & y Arcas, 2017a; Lim *et al.*, 2020). Furthermore, existing distributed learning frameworks often struggle to handle the statistical heterogeneity (Non-independent and identically distributed (Non-IID) data) inherent in diverse IoT environments (Lu, Pan, Dai, Si & Zhang, 2024).

High Mobility in Vehicular Environments: In Vehicular Edge Computing (VEC), the high velocity of vehicles creates a highly volatile network topology. Connections between vehicles and Roadside Units (RSUs) are transient and short lived. Conventional optimization methods that rely on static snapshots of the network fail to anticipate handovers, leading to task interruptions and execution failures. There is a distinct lack of frameworks that seamlessly integrate mobility prediction with decision making to ensure the reliable execution of dependent tasks (Zhang, Wang, Zhu, Cao & Liu, 2025b).

0.3 Research Objectives

The primary objective of this thesis is to design, develop, and validate intelligent resource management frameworks that address the complexities of dependent task offloading in 5G and beyond edge environments.

Long Term Objectives

The long term vision of this investigation is to develop a holistic ecosystem for edge computing where intelligent agents autonomously optimize resource allocation for complex applications. We aim to bridge the gap between theoretical optimization and practical deployment by integrating graph representation learning with distributed artificial intelligence, enabling networks that are not only reactive but also predictive and privacy preserving.

Short Term Objectives

To achieve the long term vision, the specific short term objectives of this thesis are:

- **Develop a Graph Aware Learning Framework:** To design a mechanism capable of capturing the complex topological dependencies of DAG-based applications using Graph Attention Networks (GAT) to extract meaningful embeddings that inform DRL decision making.
- **Optimize the Energy-Delay Trade-off:** To formulate multi-objective optimization problems that minimize the weighted sum of energy consumption and execution time which is a vital balance to ensure both extended device battery life and the ultra-low latency required by modern applications by jointly optimizing discrete offloading decisions and continuous resource allocation.

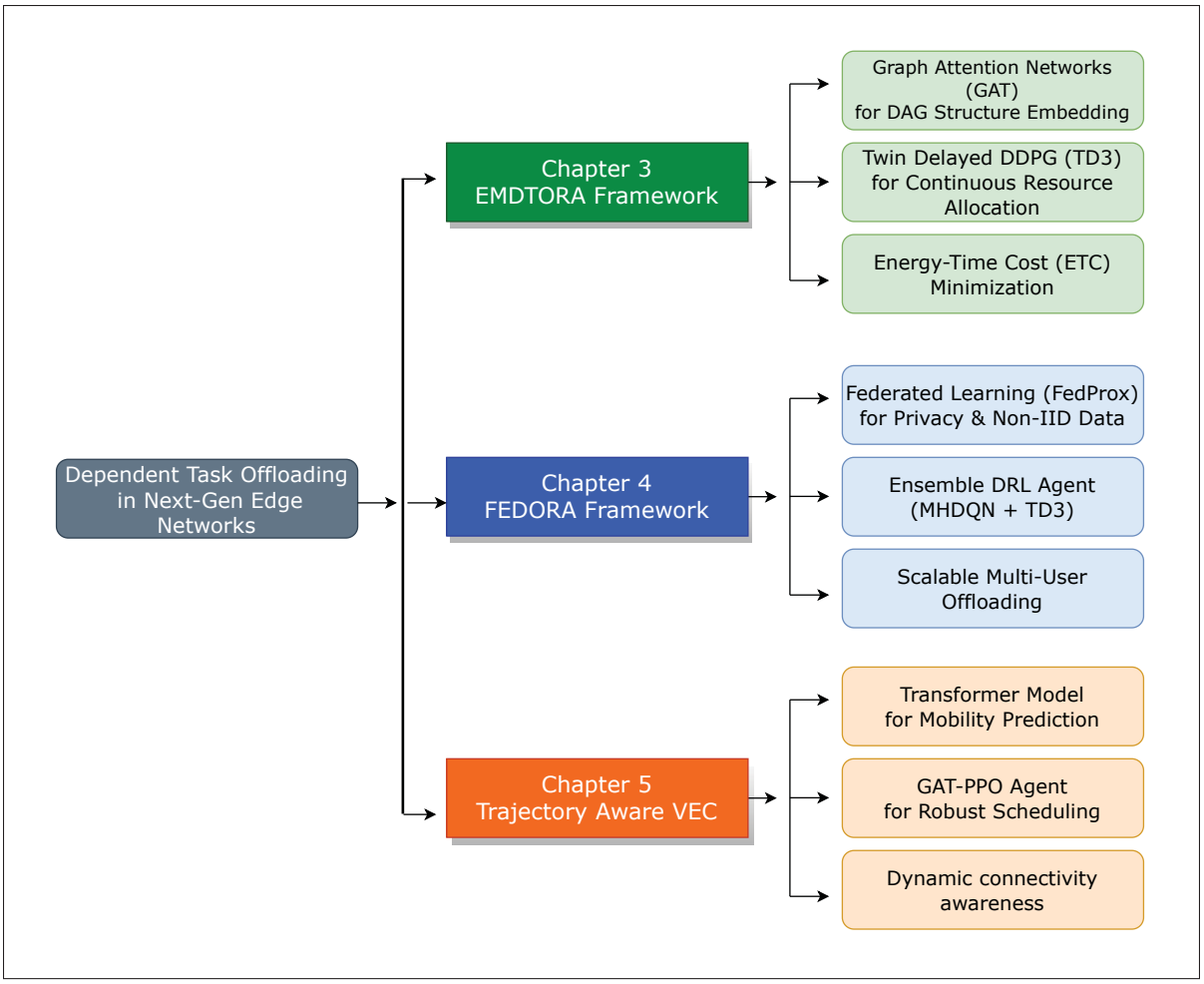

Figure 0.1 Outline of the present thesis contributions

- **Enable Privacy Preserving Scalability:** To propose a distributed learning architecture based on Federated Learning (FL) that allows edge nodes to collaboratively train global models without sharing raw user data, ensuring robustness against Non-IID data.
- **Integrate Mobility Awareness:** To tackle the stochastic nature of vehicular networks by incorporating Transformer-based trajectory prediction into the offloading logic, thereby reducing task failure rates during high-speed handovers.

0.4 Summary of Contributions

The dissertation is structured as shown in Fig. 0.1 of three major contributions to the field of Edge Computing, specifically addressing the challenges of dependent task offloading, privacy-preserving resource allocation, and mobility management in next-generation networks. These contributions correspond to the three core chapters of this dissertation:

EMDTORA: Energy-Aware Multi-User Dependent Task Offloading (Chapter 3)

To address the limitations of existing schemes that ignore task dependencies or treat them as simple sequential chains, we introduce EMDTORA, a centralized framework optimized for static IoT environments.

- We formulate the dependent task offloading problem as a Markov Decision Process (MDP) aimed at minimizing the weighted sum of energy consumption and execution delay (Energy-Time Cost).
- We propose a novel architecture that couples GAT with the Twin Delayed Deep Deterministic Policy Gradient (TD3) algorithm. The GAT component effectively encodes the complex topological structure of application DAGs into feature embeddings, while the TD3 agent manages the hybrid action space consisting of discrete offloading decisions and continuous CPU frequency scaling.
- Extensive simulations demonstrate that EMDTORA significantly outperforms baseline algorithms, including standard Deep Deterministic Policy Gradient (DDPG) and heuristic approaches, by successfully balancing the trade off between energy efficiency and delay reduction under dynamic network conditions.

FEDORA: Federated Ensemble Reinforcement Learning (Chapter 4)

Addressing the critical scalability and privacy concerns associated with centralized training in massive, heterogeneous IoT deployments, we present FEDORA, a distributed, privacy preserving resource management framework.

- We design a robust FL architecture utilizing the FedProx optimization algorithm. Unlike standard aggregation methods, FedProx incorporates a proximal term to effectively mitigate client drift caused by the statistical heterogeneity (Non-IID data) inherent in diverse IoT devices.
- We introduce an Ensemble DRL approach to handle the complex hybrid action space of MEC. We integrate Multi-Head Deep Q-Networks (MHDQN) to optimize discrete offloading decisions and TD3 agents to control continuous resource allocation. This decomposition allows for specialized learning of distinct action types while maintaining a cohesive global policy.
- By enabling edge devices to train models locally and share only model updates with a global aggregator, FEDORA preserves user data privacy and significantly reduces communication overhead. Experimental results confirm that FEDORA achieves superior convergence speed and generalization compared to independent learning and standard Federated Averaging (FedAvg) schemes.

Trajectory-Aware Offloading in Vehicular Edge Computing (Chapter 5)

To tackle the stochastic nature of high-mobility vehicular networks where task failures due to high mobility are frequent, we propose a Trajectory Aware offloading framework for VEC.

- We integrate a Mobility Aware Transformer model that leverages self attention mechanisms to capture long range temporal dependencies in vehicle movement. This module accurately predicts future trajectories and estimates RSU coverage windows over short horizons.

- We develop a GAT-Proximal Policy Optimization (PPO) agent that fuses task dependency graphs with predicted mobility embeddings. This allows the agent to make proactive offloading decisions that explicitly account for the remaining duration of connectivity.
- By aligning task execution times with RSU availability, the proposed framework drastically reduces task failure rates. Evaluations using real world mobility traces demonstrate that this method consistently outperforms non predictive and compute-aware baselines in terms of reliability and energy efficiency.

0.5 Thesis Organization

The remainder of this thesis is organized as follows. Chapter 2 provides a comprehensive review of the state-of-the-art literature, covering MEC, DRL, GNN, and FL. Following, Chapter 3 details the EMDTORA framework, focusing on the fundamental problem of offloading dependent tasks in multi-user scenarios. Chapter 4 presents the FEDORA framework, expanding the solution to a distributed, privacy preserving architecture. Chapter 5 introduces the Trajectory-Aware VEC framework, addressing the specific constraints of high-mobility vehicular environments. Finally, Chapter 6 concludes the thesis, summarizing the key findings and suggesting directions for future research.

0.6 Related Publications

The research reported herein was also reported in the following published and accepted research articles. Journal publications are denoted by J, while conference proceedings are marked with C.

J1: **S Khan,** M Avgeris, J Gascon-Samson, A Leivadeas, "EMDTORA: Energy-aware multi-user dependent task offloading and resource allocation in MEC using graph-enabled DRL" *IEEE Transactions on Green Communications and Networking*, 2024.

J2: **S Khan,** A Ali-Pour, M Avgeris, J Gascon-Samson, A Leivadeas, "FEDORA: Federated Ensemble Reinforcement Learning for DAG-based task offloading and resource allocation in MEC" *IEEE Internet of Things Journal*, 2025.

C1: **S Khan,** M Avgeris, A Ali-Pour, J Gascon-Samson, A Leivadeas, "Dependent Task Offloading in Vehicular Edge Computing Using Trajectory-Aware Deep Reinforcement Learning" accepted in *2026 IEEE International Conference on Communications (ICC)*.

CHAPTER 1

BACKGROUND AND LITERATURE REVIEW

This chapter establishes the theoretical foundations and reviews the state-of-the-art literature relevant to this thesis. We first introduce essential background concepts, including MEC, task dependency modeling via DAGs, DRL, GNNs, FL and VEC. Building on these foundations, we then present a comprehensive literature review that identifies the broader research gaps in dependent task offloading.

1.1 Multi-Access Edge Computing (MEC)

The proliferation of the IoT and the rapid commercialization of 5G networks have fundamentally transformed the telecommunications landscape. We are currently transitioning from an era of simple connectivity to an era of ubiquitous intelligence, often referred to as the Internet of Everything (IoE). As highlighted by recent research into efficient resource allocation, the number of connected devices is projected to grow exponentially, generating massive volumes of data at the network edge (Elkawkagy *et al.*, 2025). This data explosion is primarily driven by emerging, resource intensive applications such as autonomous driving, AR, VR, and smart city surveillance and all of which impose stringent requirements on latency, reliability, and bandwidth (Chen, Cheng *et al.*, 2024c; Adhikari, Khwaja, Jaseemuddin & Anpalagan, 2024).

Traditionally, Mobile Cloud Computing (MCC) served as the dominant architecture for processing data generated by mobile devices. In the MCC paradigm, mobile devices (MDs) offload their computational tasks to centralized cloud datacenters (e.g., Amazon AWS, Google Cloud) located deep within the internet. While public clouds offer virtually unlimited storage and processing capabilities, this centralized model faces the following inherent limitations in the context of next generation networks:

- **High Latency:** The physical distance between User Equipment (UE) and remote cloud datacenters results in significant propagation latency. Furthermore, data must traverse multiple hops through the Wide Area Network (WAN), incurring unpredictable routing and

queuing delays that are unacceptable for mission critical applications requiring millisecond reaction times (Oikonomou, Plastras, Tsoumatidis, Skoutas & Rouskas, 2024).

- **Backhaul Congestion:** Offloading massive volumes of raw data to the cloud places an immense burden on the mobile backhaul network, frequently leading to network congestion and degraded QoS for all users (Naval & Pushparajesh, 2025).
- **Privacy and Security:** Transmitting sensitive user data such as healthcare records or real-time location traces to remote servers raises significant privacy concerns and increases exposure to potential cyber attacks during transmission.
- **High Operational Costs:** Constantly sending massive amounts of data, such as continuous video streams or vehicle sensor readings, to distant cloud servers is very expensive due to high internet bandwidth and data transfer fees. MEC reduces these costs by processing the data locally and only sending the most important information to the main network (Cao *et al.*, 2024a).

To mitigate these critical issues, MEC has emerged as a pivotal technology. MEC fundamentally shifts the computational paradigm by decentralizing resources. By deploying storage and computing capabilities at the edge of the Radio Access Network (RAN), MEC brings processing power into close proximity with mobile users (e.g., at Base Stations (BS), Access Points (AP), or RSU) (Dustdar *et al.*, 2024).

1.1.1 Evolution from Cloud to Edge (The 5G/6G Drivers)

Historically, the evolution of wireless networks has been driven by the demand for higher data rates. However, the transition to 5G and upcoming 6G networks represents a much broader mission: supporting ultra-reliable low-latency communications (URLLC) alongside enhanced mobile broadband (eMBB) (Chen *et al.*, 2024c).

In the context of 6G, the network is envisioned to support applications requiring data rates of up to several terabits per second (Tbps) and ultra-low latency. While advanced physical layer technologies are being explored to meet these demands, they face significant challenges in

mission-critical, latency-sensitive applications like autonomous vehicular navigation regarding path loss and limited transmission range (Yin, Guan, Liu, Jin & Zhang, 2024). Furthermore, while emerging networks will provide vastly more backhaul capacity, the exponential explosion of data generated by the IoE is projected to continuously outpace these capacity upgrades.

Therefore, MEC is not just highly relevant in current-generation networks for alleviating existing backhaul constraints; it becomes exponentially more critical as networks evolve. By physically bringing the cloud closer to the user, MEC serves as the essential architectural enabler that bridges the gap between advanced wireless speeds and actual application requirements. Processing data at the edge facilitates the following concepts:

1. **Ultra-Low Latency:** By eliminating the physical distance and round-trip propagation time to the centralized core network, MEC enables the true real-time interactivity required for the tactile internet and autonomous navigation (Naval & Pushparajesh, 2025).
2. **Context Awareness:** Edge servers situated at the RAN have real-time access to radio network information (e.g., user location, cell load). This allows for highly optimized, context-aware service delivery that centralized, distant clouds simply cannot achieve (Tong, Chen & Zhu, 2025).
3. **Backhaul Optimization:** MEC enables local data filtering and analytics. Instead of blindly sending raw, high-volume data across the network, edge nodes process it locally and transmit only essential metadata, preventing future backhaul links from being overwhelmed despite their increased capacity (Oikonomou *et al.*, 2024).

1.1.2 MEC System Architecture

The MEC system is typically modeled as a hierarchical structure comprising three distinct layers: the Terminal Layer, the Edge Layer, and the Cloud Layer (Gu, Zhao, Han, Zheng & Song, 2023; Yu *et al.*, 2024).

1.1.2.1 Terminal Layer (User Equipment)

This layer consists of heterogeneous IoT devices and mobile terminals, such as smartphones, connected vehicles, and smart sensors. These devices act as the primary sources of computational tasks. While they possess some local processing power, they are inherently constrained by limited battery life and computational capacity (Zheng, Saad *et al.*, 2024b).

1.1.2.2 Edge Layer

The Edge Layer comprises MEC servers deployed at network access points, such as cellular BSs or RSUs. Each BS is equipped with a MEC server possessing a specific computing resource capacity. These edge servers communicate with user devices via wireless uplinks/downlinks and interface with one another via high speed optical fibers or wireless backhaul links. This layer is primarily responsible for handling delay-sensitive tasks and providing real time feedback to users (Dustdar *et al.*, 2024).

1.1.2.3 Cloud Layer

The Cloud Layer represents the traditional centralized cloud datacenter. It possesses virtually infinite computational and storage resources. This layer is reserved for handling delay tolerant tasks, long-term data storage, and large-scale data analytics that exceed the localized capacity of the edge layer (Oikonomou *et al.*, 2024).

1.1.3 Computation Offloading Classifications

The core functional mechanism of MEC is computation offloading. The transfer of computational tasks from resource constrained mobile devices to resource rich edge servers. Offloading strategies are generally classified based on the granularity of the task partitioning (Zhou *et al.*, 2024b).

1.1.3.1 Binary Offloading

In the binary offloading model, a computational task is treated as an indivisible, atomic unit. The offloading decision is represented by a binary variable $a_i \in \{0, 1\}$:

- If $a_i = 0$, the entire task is executed locally on the mobile device.
- If $a_i = 1$, the entire task is offloaded to the MEC server.

While straightforward to implement, binary offloading lacks the flexibility required for complex, modern applications particularly those involving large task sizes or highly constrained network bandwidths. The complexity of mobile applications has escalated with the capabilities of end user devices. While early MEC research predominantly modeled workloads as simple, atomic tasks where a single request could be executed independently. This model is increasingly insufficient for modern use cases. Applications such as real-time video analytics, immersive VR, and cooperative autonomous driving are structured as complex workflows composed of numerous interdependent sub tasks (Zhang, Luo, Chen, Shen & Guo, 2024d).

1.1.4 From Monolithic to Fine-Grained Task Decomposition

The structural evolution of mobile applications has increasingly shifted from monolithic architectures to component-based, fine-grained task models. In a monolithic paradigm, an application is built as a single, indivisible computational unit. Consequently, computation offloading decisions are strictly binary: the entire application is either executed locally or migrated in its entirety to the edge server. This lack of granularity often leads to inefficient resource utilization; for instance, a small, computationally intensive but delay-sensitive component might force the migration of a massive, data-heavy application, consuming excessive radio bandwidth (Xiao *et al.*, 2024c).

Conversely, modern edge-enabled applications decompose these monolithic structures into a collection of dependent, fine-grained computational subtasks typically modeled as DAGs which are optimized for radio level and network level orchestration. This paradigm enables true Partial Offloading, where the mobile device and the MEC server can dynamically partition the DAG,

transmitting only specific sub-tasks over the Radio Access Network (RAN) for remote execution while keeping others local.

1.1.4.1 Partial Offloading

Partial offloading allows an application to be partitioned into smaller, modular components. A subset of these components is executed locally on the device, while the remainder is offloaded to the edge. This approach enables parallel processing and highly fine-grained resource management. Partial offloading is further categorized into Independent task offloading and Dependent task offloading. In independent task offloading, the application subtasks are entirely independent of one another and can be executed in any order or in parallel.

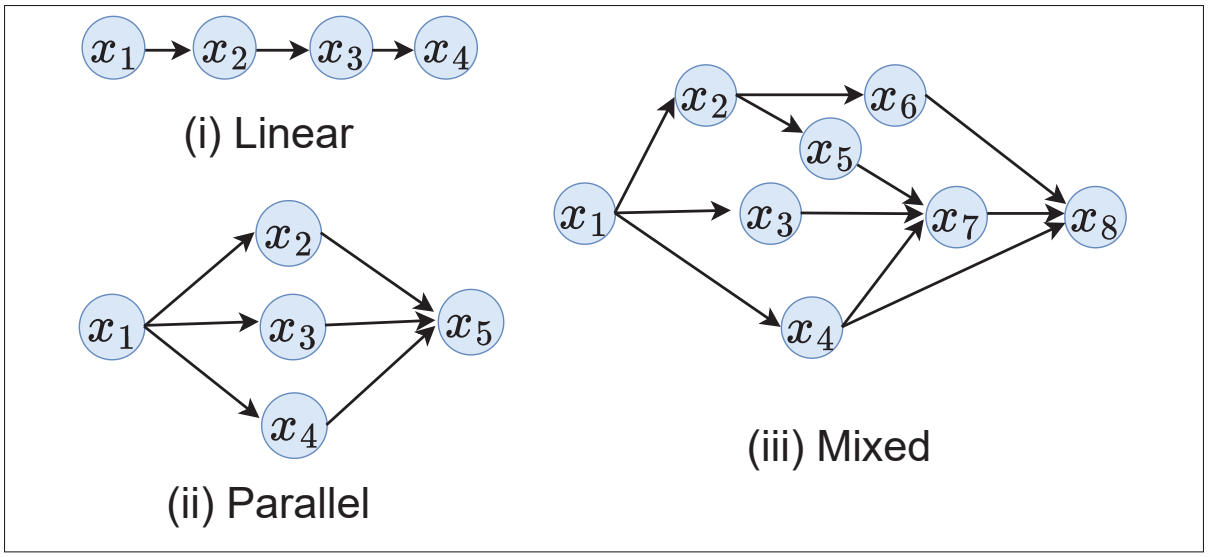

Figure 1.1 Task Interdependency Forms

1.1.4.2 Dependent task offloading

Advanced applications such as facial recognition, video analytics, or interactive VR consist of interdependent sub tasks with strict precedence constraints (Mao, You, Zhang, Huang & Letaief, 2017a). These applications are mathematically modeled as DAGs $G = (V, E)$, where the vertices V represent the sub tasks and the edges E represent the data dependencies between

them. There is a logical relationship between tasks within an IoT application that allows their execution to be dependent on the results of their counterparts. This is known as a task dependency relationship.Since some components serve as mandatory inputs for others, their execution sequence cannot be assigned arbitrarily. Instead, a strict execution order must be established by traversing the DAG topology, typically through a topological sort to ensure all predecessor dependencies are fully resolved before any subsequent task begins execution. It is necessary to employ more sophisticated task models to understand this interdependence. The structure of dependent tasks of an application as shown in Fig. 1.1 can be a sequential, parallel model, or general model of subtask dependency. In native mobile applications, the first and last steps, such as collecting I/O data and displaying computation results, are normally executed locally. Additionally, the vertices of the task-call graph can also specify the computational workloads and resources required by each procedure, including the number of CPU cycles and the amount of memory required. For example, gesture recognition applications as shown in Figure 1.2 consists of a number of tasks that are interdependent (Li *et al.*, 2024a).

In a DAG based model, the execution order introduces highly complex constraints involving computation time, transmission time, and idle waiting time. For example, if a parent task is executed on the mobile device and its child task is offloaded to the edge, the output data of must be transmitted over the wireless channel, incurring an unavoidable transmission delay (Zheng *et al.*, 2024a). Efficiently scheduling DAG-based applications requires determining the optimal execution order and identifying the bottleneck tasks.

Since the graph is acyclic, it is possible to produce a topological sort that is a linear ordering of vertices such that for every directed edge (v_i, v_j), vertex v_i appears before v_j in the ordering. This sorting is fundamental for sequential decision-making processes, such as those used in DRL agents (e.g., Sequence-to-Sequence models), ensuring that a scheduler always considers a parent task before its children (Zhang *et al.*, 2025a).

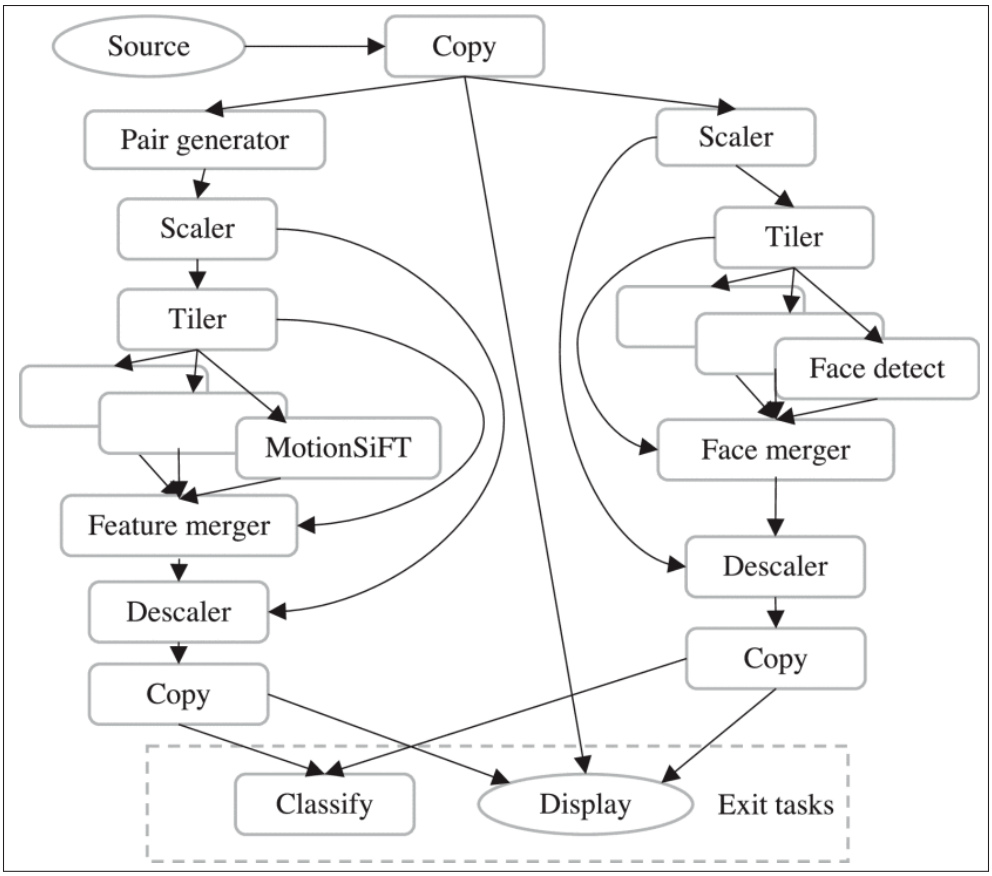

Figure 1.2 Gesture recognition application
Taken from Li *et al.* 2024a

The rank of a task, often used to determine priority in topological sorting, can be calculated recursively based on the computation cost and the maximum rank of its successors:

$$rank(v_i) = \bar{w}_i + \max_{v_j \in Suc(v_i)} (rank(v_j) + \bar{c}_{i,j}). \tag{1.1}$$

where $\bar{w}_i$ is the average execution time and $\bar{c}_{i,j}$ is the average communication cost (Liu *et al.*, 2024c). The performance of a DAG-based application is bounded by its critical path (CP). The CP is defined as the longest path from the entry node to the exit node, where the length is the sum of computation and transmission delays along the path.

Tasks lying on the critical path are paramount; delaying any task on the CP directly increases the total completion time of the application. Conversely, tasks on non-critical paths have a certain amount of slack time. Optimal offloading strategies often focus on migrating critical tasks to high-performance edge servers to reduce execution time, or keeping consecutive critical tasks on the same node to eliminate transmission delays (Chen *et al.*, 2024a).

1.1.5 Challenges in Dependent Task Offloading

Task offloading is primarily intended to minimize delay, improve energy consumption, optimize bandwidth utilization, and balance load, however, there are certain challenges associated with task offloading in order to achieve these objectives. Network, resource allocation, and task management challenges are some of these challenges. Dynamic network conditions, such as interference, fading, shadowing, and noise, can significantly impact the network throughput of users. In resource scheduling, optimal task selection is critical and is the basis of efficient task offloading and resource allocation that can guarantee QoS and other application parameters:

- **Varying Network Conditions**

  As a result of the dynamic nature of the mobile network and the significant variation in network conditions, offloading tasks become more complex. Moreover, wireless channels are greatly affected by noise, fading, interference, and shadowing phenomena, which can adversely affect the transmission quality and throughput of wireless signals. Before offloading a task to the edge of the network, it is crucial to predict and analyze network conditions.

- **Edge/Cloud Dynamics**

  The dynamics between the edge and cloud depend fundamentally on where orchestration occurs (Taleb *et al.*, 2017). Centralized orchestration requires edge nodes to continuously stream state metrics to the cloud to construct a global view, enabling optimal decisions but incurring severe latency and communication overhead. Conversely, distributed orchestration, essential for highly mobile vehicular networks pushes decision making directly to the edge (Ning *et al.*, 2020). Because these autonomous nodes rely solely on a local perspective,

achieving efficient, globally coordinated task management without a central cloud controller remains a primary challenge.

- **Offloading Decisions**

  The selection of tasks to offload to the edge or cloud is a significant challenge when it comes to task offloading. In order to offload tasks intelligently, effective methods must be used. Poor selection will adversely affect the network's performance in terms of delay, cost, energy consumption, and resource waste.

- **Resource Management**

  It is important to note that compared to cloud computing, the resources at the edge of the network are limited and should be managed efficiently. In addition to task selection, server-side resource allocation is also imperative to improving the churn rate and network density.

To solve the complex scheduling challenges in distributed MEC environments, the next section introduces GNNs. Specifically, we will explore how GNNs provide an ideal mathematical tool for understanding the complicated structures of both the edge networks and the dependent tasks running on them.

1.2 Graph Neural Networks (GNNs)

The representation learning capability of Deep Learning (DL) has revolutionized various domains, particularly Computer Vision (CV) and Natural Language Processing (NLP). However, the data structures in these domains such as images and text sequences exist in Euclidean space with regular, grid-like structures. In contrast, the dependencies between computational tasks in MEC are modeled as DAGs. These graph structures are fundamentally non-Euclidean, characterized by irregular connections and a lack of fixed spatial ordering. To effectively capture the topological information inherent in dependent task offloading, this thesis leverages the specialized architecture of GNNs.

1.2.1 Limitations of CNNs and RNNs on Non-Euclidean Data

While Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) are powerful, they face intrinsic limitations when applied to graph-structured data (Marwani & Kaddoum, 2024).

1.2.1.1 Convolutional Neural Networks (CNNs)

CNNs are designed to extract multi-scale localized spatial features from data represented on regular 2D grids such as images. The success of CNNs relies on three key priors: local connectivity, shared weights, and a hierarchical structure (Ponzi & Napoli, 2025):

- **Translation Invariance:** In a grid, the number of neighbors for a pixel is fixed (e.g., 8 neighbors). This allows for the definition of a learnable filter (kernel) of a fixed size (e.g., 3×3) that can be shifted across the image. The statistical properties of the data are assumed to be invariant to translation.
- **Irregular Neighborhoods:** In a graph $G = (V, E)$, the number of neighbors (degree) varies for each node $v \in V$. There is no natural ordering of neighbors; a node might have 2 neighbors while another has 100. Consequently, it is mathematically not connivent to apply a fixed-size convolutional filter directly to a graph without a predefined ordering or grid structure (Pistilli & Averta, 2023).

Attempting to force graph data into a grid format (e.g., via adjacency matrices fed into a CNN) results in a loss of permutation invariance. If the nodes are reindexed, the matrix changes, potentially confusing a standard CNN, whereas the underlying graph topology remains identical.

1.2.1.2 Recurrent Neural Networks (RNNs)

RNNs, including Long Short-Term Memory (LSTM) networks, process data sequentially. They are effective for 1D sequences (e.g., time-series or text) where there is a strict linear precedence:

- **Lack of Total Ordering:** While DAGs possess a partial ordering such as topological sorting, they do not have a unique total ordering. Flattening a DAG into a sequence to feed into an

RNN can result in the loss of structural information, specifically parallel branches. If task A and task B can run in parallel, an RNN must process them in an arbitrary sequence (e.g., $A \rightarrow B$), potentially inferring a false dependency that does not exist (Wang *et al.*, 2024b).

GNNs address these limitations by generalizing deep learning operations to non-Euclidean domains. GNNs define computation directly on the graph structure, maintaining permutation invariance and naturally handling varying node degrees.

1.2.2 Spectral vs. Spatial GNNs

The development of GNNs has generally followed two distinct paradigms: spectral-based approaches and spatial-based approaches.

1.2.2.1 Spectral Graph Theory Approaches

Spectral approaches define the graph convolution operation in the Fourier domain by utilizing the eigen-decomposition of the graph Laplacian. Let A be the adjacency matrix and D be the diagonal degree matrix. The normalized graph Laplacian is defined as $L = I_N - D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$. The convolution of a graph signal $x \in \mathbb{R}^N$ with a filter g_θ is defined as:

$$g_\theta \star x = U g_\theta(\Lambda) U^T x. \tag{1.2}$$

where U is the matrix of eigenvectors of L, and Λ is the diagonal matrix of eigenvalues. While theoretically sound, spectral methods face significant challenges in MEC applications:

1. **Computational Complexity:** The eigen-decomposition required to find U has a complexity of $O(N^3)$, which is prohibitively expensive for large-scale graphs.
2. **Domain Dependency:** The learned filters depend on the Laplacian basis, meaning a model trained on one graph structure cannot easily generalize to a graph with a different structure. This is critical in MEC, where task graphs vary across different applications (Pistilli & Averta, 2023).

1.2.2.2 Spatial Approaches

Spatial approaches, in contrast, define convolutions directly on the graph by aggregating features from spatially close neighbors. This is analogous to the standard CNN convolution but operates on an arbitrary neighborhood structure. The general propagation rule for spatial GNNs can be expressed as a message passing function:

$$h_v^{(l+1)} = \sigma\left(\sum_{u \in \mathcal{N}(v)} \psi\left(h_u^{(l)}, h_v^{(l)}\right)\right). \tag{1.3}$$

where $h_v^{(l)}$ is the feature vector of node v at layer l, $\mathcal{N}(v)$ is the set of neighbors, and ψ is a learnable function. Spatial methods are generally preferred for task offloading problems because they are computationally efficient (often linear in the number of edges) and capable of inductive learning generalizing to unseen graph topologies (Zhao, Perazzone, Verma & Segarra, 2024b).

1.2.3 Graph Convolutional Networks (GCN)

The Graph Convolutional Network (GCN) bridges the gap between spectral and spatial methods. It effectively approximates the spectral convolution using a first order approximation of Chebyshev polynomials, resulting in a spatially localized operation. To avoid explicit eigen-decomposition, the filter g_θ can be approximated by a truncated expansion of Chebyshev polynomials $T_k(x)$ up to K^{th} order. By limiting $K = 1$ (considering only immediate 1-hop neighbors) and approximating the largest eigenvalue $\lambda_{max} \approx 2$, the convolution operation simplifies to:

$$g_\theta \star x \approx \theta_0 x + \theta_1 (L - I_N)x = \theta_0 x - \theta_1 D^{-\frac{1}{2}} A D^{-\frac{1}{2}} x. \tag{1.4}$$

To prevent numerical instabilities and exploding/vanishing gradients when stacking multiple layers, GCN employs the renormalization trick:

$$I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \rightarrow \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}. \tag{1.5}$$

where $\tilde{A} = A + I_N$ is the adjacency matrix with added self-loops, and $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$.

1.2.3.1 Layer-wise Propagation

The final layer-wise propagation rule for a GCN is:

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right). \tag{1.6}$$

where $H^{(l)}$ is the matrix of node features at layer l, $W^{(l)}$ is the layer-specific trainable weight matrix, and σ is a non-linear activation function (e.g., ReLU).

1.2.3.2 Limitations for Dependent Task Offloading

While GCNs are efficient, Eq. (1.6) reveals a critical limitation: the aggregation is isotropic. The contribution of neighboring node j to node i is determined solely by the structural degrees $\sqrt{\tilde{d}_i\tilde{d}_j}$. This means GCN treats all neighbors as equally important by normalizing by degree. In DAG based task offloading, not all dependencies are equal. Some tasks lie on the critical path that is the longest sequence of dependent tasks that determines the total application completion time. A delay in a critical parent task has a much more severe impact than a delay in a non critical task. Therefore, the learning model must be able to assign different importance weights to different predecessors, which necessitates an attention mechanism (Wang *et al.*, 2024b).

1.2.4 Graph Attention Networks (GAT) and Attention Mechanisms

To address the shortcomings of static weighting in GCNs, GAT incorporate the attention mechanism into the propagation step, allowing the network to learn dynamic weights for neighbors based on their node features. This capability is crucial for the frameworks proposed in this thesis (EMDTORA and FEDORA), as it enables the encoder to identify and prioritize critical tasks within the dependency graph.

1.2.4.1 The Attention Mechanism

For a given node pair (i, j), GAT computes an attention coefficient e_{ij} that indicates the importance of node j's features to node i. This is computed using a shared attentional mechanism $a : \mathbb{R}^{F'} \times \mathbb{R}^{F'} \rightarrow \mathbb{R}$:

$$e_{ij} = a(Wh_i, Wh_j). \tag{1.7}$$

where $W \in \mathbb{R}^{F' \times F}$ is a weight matrix applied to every node. In the standard GAT implementation, the mechanism a is a single layer feedforward neural network parametrized by a weight vector $\mathbf{a}$, followed by a LeakyReLU nonlinearity:

$$e_{ij} = \text{LeakyReLU}\left(\mathbf{a}^T [Wh_i || Wh_j]\right). \tag{1.8}$$

Here, $||$ represents the concatenation operation. The attention coefficients are then normalized across all neighbors $\mathcal{N}_i$ using the softmax function to make them comparable:

$$\alpha_{ij} = \text{softmax}_j(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k \in \mathcal{N}_i} \exp(e_{ik})}. \tag{1.9}$$

1.2.4.2 Feature Aggregation

Once the normalized attention coefficients α_{ij} are computed, they serve as weights for the linear combination of the neighbors' features. The update rule for node i is:

$$h'_i = \sigma\left(\sum_{j \in \mathcal{N}_i} \alpha_{ij} W h_j\right). \tag{1.10}$$

Unlike GCN, where the weights are fixed structural constants, here α_{ij} depends on the specific features of the tasks (e.g., workload, data size). This allows the model to learn, for instance, that a predecessor task generating a large volume of data (high transmission delay risk) should be attended to more heavily than a computationally light predecessor.

1.2.4.3 Multi-Head Attention

To stabilize the learning process and capture different types of dependencies (e.g., one head focusing on computational intensity, another on data transmission size), GAT employs Multi-Head Attention. K independent attention mechanisms execute the transformation of Eq. (1.10), and their results are concatenated:

$$h'_i = \Big\|_{k=1}^{K} \sigma\left(\sum_{j \in \mathcal{N}_i} \alpha_{ij}^{(k)} W^{(k)} h_j\right). \tag{1.11}$$

This multi-head approach significantly increases the expressive power of the network. In the context of our proposed frameworks, this allows the DRL agent to extract a high level embedding of the entire application DAG that encapsulates both the structural properties

1.3 Deep Reinforcement Learning

DRL represents the intersection of Reinforcement Learning (RL) and DL. While traditional RL provides a framework for decision-making agents to learn optimal control policies through trial-and-error, it struggles with the "curse of dimensionality" in environments with high-dimensional state spaces. DRL addresses this by utilizing Deep Neural Networks (DNNs) as robust function approximators to estimate value functions or policies, enabling agents to solve complex tasks ranging from Atari games to robotic control and network resource management (Mnih *et al.*, 2015). In RL, an agent interacts with an environment in a series of discrete timesteps, $t = 0, 1, 2, \ldots, T$, to fulfill a task. Observing the environment at time t, the agent observes the set of possible states of the environment $s_t \in \mathbb{S}$. Afterward, the agent performs an action $a_t \in \mathbb{A}$, which contains a listing of all possible actions that could be taken by the agent.

At the time $t + 1$, the environment transitions into a new state s_{t+1} and yields a scalar reward r_{t+1} as a result of the state-action pair (s_t, a_t). The agent's fundamental objective is not just to maximize the immediate reward, but to maximize the expected cumulative discounted reward

over its entire trajectory. Mathematically, the total return R_t that the agent seeks to maximize at the current time step t is formulated as:

$$R_t = \sum_{\tau=t}^{\infty} \gamma^{\tau-t} r_{\tau+1}. \tag{1.12}$$

To fully contextualize this objective function, the parameters are defined as follows:

- R_t: The total accumulated return expected from the current time step t onward.
- τ: The index representing future time steps.
- $r_{\tau+1}$: The immediate scalar reward received from the environment at future step $\tau + 1$.
- $\gamma \in [0, 1]$: The discount factor, which determines the present value of future rewards. A value of γ close to 0 renders the agent strictly prioritizing immediate rewards, whereas a γ closer to 1 makes the agent heavily weighting long-term strategic gains.

An agent typically follows a specific decision policy π, which dictates the mapping from states to actions. RL methods govern how this policy evolves in accordance with the agent's accumulated experience. Through continuous interaction, the agent eventually learns an optimal decision policy π^*, which successfully maximizes the long-term accrued return R_t for any given state s.

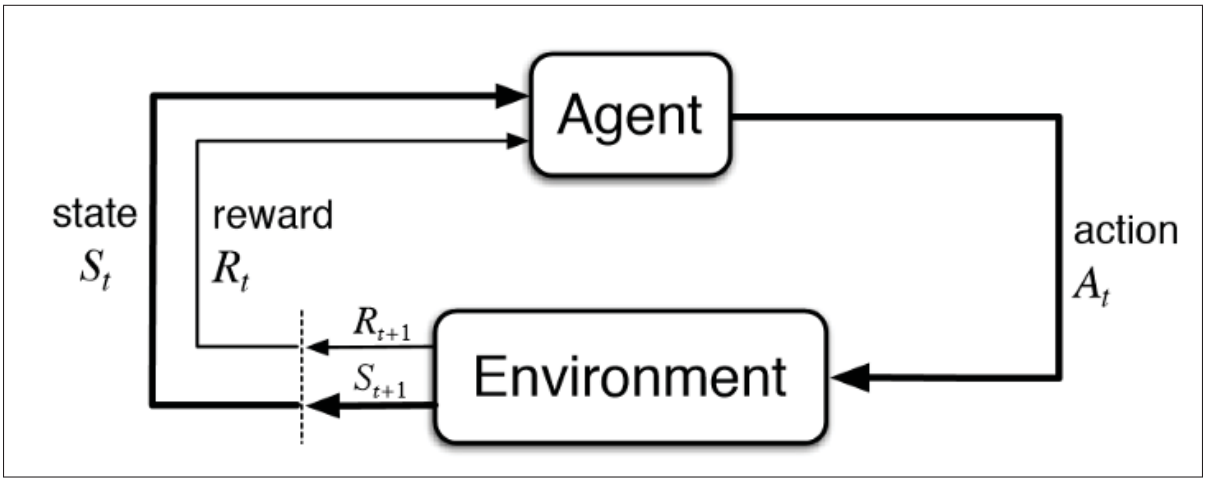

Figure 1.3 Generic RL model
Taken from Sutton & Barto 2018a

We can combine RL with several machine learning algorithms to efficiently reach the optimal solution in complex problems with high-dimensional data. For example, Principal Component Analysis (PCA) is an unsupervised learning technique that can be applied with RL methods for dimensionality reduction (François-Lavet, Henderson, Islam, Bellemare & Pineau, 2018). Likewise, RL methods combined with deep learning algorithms, which are famous for the efficient extraction of essential features and their ability to find non-linear relationships between input and output data, can be used to approximate the action-value function $q(s, a)$ and the policy function $\pi(a \mid s)$. A generic RL model is depicted in Figure 1.3. In the following sections, we explain these concepts in more detail.

1.3.1 Markov Decision Process

A mathematical framework called the MDP is used to model decision-making problems. For example, in Figure 1.3, the dynamics of the RL environment are modeled using the MDP. MDPs involve interaction between two entities: the agent and the network environment, which are independent of the decision-making process. An agent interacts discretely with its environment. It creates interaction sequences during discretized time steps $t = [0, 1, 2, 3...]$. Each time step, for example, the agent observes the state of the environment S_t. In response to the agent's actions, the environment changes to the next state S_{t+1}, and the agent receives a reward R_{t+1}. The interaction cycle continues until a predetermined timestep, or terminal state is reached. The MDP then forms observation sequences O_t, actions A_t, and rewards R_t after the interactions have concluded. The Markov property states that future outcomes depend solely on the present state. Consequently, the current state alone provides all the necessary information for decision-making, rendering past history mathematically irrelevant. The state S_t and reward R_t at time instant t in this framework are random variables that depend only on previous actions and states.

$$P(s' \mid s, a) = p(s_{t=s'} \mid S_{t-1} = s, A_{t-1} = a) = \sum_{r \in R} P(s', r \mid s, a). \tag{1.13}$$

The agent's main objective is to maximize the rewards it receives from the environment at different time steps $R_{t+1}, R_{t+2}, R_{t+3}.....$ In general terms, the rewards can be represented by G_t ,

as shown in equation 1.14, by taking different actions in observed states.

$$G_t = R_{t+1} + \gamma \times R_{t+2} + \gamma^2 \times R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k \times R_{t+k+1}. \tag{1.14}$$

If γ is set to 0, the agent cares only about the actions that produce immediate reward and discards all other actions that result in future rewards. Alternatively, $\gamma = 1$ suggests that the agent cares about all the actions based on the sum of all rewards, including immediate rewards and the rewards that it will encounter in the future (Sutton & Barto, 2018b).

1.3.2 Policy Function

The policy function is used to map the states of the environments to the agent's actions, and it can be broadly classified into two categories: behavior policy and target policy. Behavior policy is used for choosing an action. Conversely, the Target policy is the policy that is to be estimated by a model for improvement and deployment. Behavior policy can further be classified as stochastic, deterministic, or random policy. At any given state, the RL agents following a stochastic policy π choose an action from the probability distribution such that $a = \pi(s)$. On the other hand, in deterministic policy, the agent selects only one action $a = \pi(s)$ for the encountered state. Finally, as the name suggests, in a random policy, the agents choose random actions a = rand(A) from the list of actions to maximize the rewards. In initial interactions between an agent and the environment, the agent will always explore new actions randomly to assess and obtain good actions that generate better rewards. However, when the agent has information about some actions, the agent acts greedily in the exploration step by choosing the actions guaranteed to result in higher rewards. A famous exploration strategy in RL theory is the epsilon greedy (ε - greedy) approach. In (ε - greedy), the agent chooses actions randomly with probability ε, thereby exploring the environment. Alternatively, the agent acts greedily with probability $1 - \varepsilon$ by only taking the best-perceived actions, thus exploiting. The value of ε is another hyper-parameter that can be adjusted according to the demand for exploration and exploitation. There is no terminal state for continuing tasks as they do not have a logical final state. They will never be completed. As an example, a continuous task is when the agent learns

how to drive, and it never stops learning, while an episodic task has a logical ending: the ending state is called the terminal state. In chess, the agent's terminal state is win, lose, or tie. In an episodic task, each episode is independent of other episodes. Therefore, after each episode, the environment resets to its initial state, irrespective of its previous state.

1.3.3 State value and Action value functions

An essential feature in finding an optimal policy is knowing each state's value. The state value function is the measure of the goodness of states. An agent can measure the state's worth from the estimated reward of that state. As shown in equation 1.15, the state value can be calculated as the total estimated rewards that the agent can obtain by starting in a state and then following policy π after that.

$$v_\pi s = \mathbb{E}[R_t \mid S_t = s] \forall s \in (S). \tag{1.15}$$

Dynamic programming uses the Bellman equations to calculate the value functions in the MDP framework. For example, the Bellman value function (equation 1.16) relates the current state with the possible future states.

$$V_\pi(s) = \sum_a \pi(a \mid s) \sum_{s',r} P(s', r \mid s, a)[r + \gamma \times V_\pi(s')]. \tag{1.16}$$

In solving MDP, our goal is to find an optimal value function, which the agent can later use to find optimal policies. The optimal value function (equation 1.17) is the maximum state value.

$$V_*(s) = \max_\pi \times V_\pi(S). \tag{1.17}$$

The action-value function shows the impact of different actions taken in each state by following a policy π. In RL, we are interested in finding optimal policies by finding an optimal action-value function that is impossible without the state-value function. Equation 1.18 describe the

state-value function.

$$q_\pi(s, a) = \mathbb{E}[R_t \mid s_t = s, a_t = a]. \tag{1.18}$$

The Bellman equation 1.19 for the action-value function shows the relationship between the current action-value in a state and its possible successor action-value pairs (Sutton & Barto, 2018b).

$$q_\pi(s, a) = \max_{s',a} p(s', r \mid s, a)[r + \gamma \times \sum_{a'} \pi(a', s') q\pi(s', a'))]. \tag{1.19}$$

The maximum value of an action in a given state can be achieved for convergence to the optimal action-value function, as shown in equation 1.20 and 1.21.

$$q_*(s, a) = \max_{\pi} \times q_\pi(s, a). \tag{1.20}$$

$$q_*(s, a) = \sum_{s',a} p(s', r \mid s, a)[r + \gamma \times \max_{a}' q_\pi(s', a'). \tag{1.21}$$

1.3.4 Model-free, and Model-based algorithms

RL agents can optimize the policy of the model to achieve the highest reward in different ways. The most common methods are classified as model-based optimization and model-free optimization. The critical difference between the two methods is the availability and access to the dynamics of the environment. We know the rewards function and the state transition probabilities beforehand in the model-based optimization method. Thus, using these two notions, we can easily estimate the policy. Value Iteration (VI) is one method of model-based learning which uses the Bellman equations to compute the optimal policy of the MDP. In contrast, a model-free method is used in cases without prior information about the model. Therefore, in model-free methods, the agent follows a random policy and improves it step by step until

an optimum policy is enacted. Examples of model-free methods are Monte Carlo (MC) and Temporal Difference (TD), which do not require complete knowledge of the environment to compute value functions and optimal policies.

1.3.5 Dynamic Programming

Although Bellman equations for the state value function and action-value function can help us solve RL problems efficiently, when the state space increases, it is not feasible to compute the values for several reasons. First, estimating both value functions requires a lot of computation resources. Secondly, it takes a lot of time to solve the Bellman equations for each possible action in each environment state. Dynamic programming (DP) was introduced to compute the value function and optimal policy iteratively to solve computation time and complexity issues. The DP algorithms consist of policy evaluation, policy improvement, and iterations. However, in DP algorithms, it is assumed that the transition probabilities and reward function are known to us a priori. To optimize the value function of the current state, dynamic programming uses the expected value of the following state by a method called bootstrapping. Policy evaluation, also known as a prediction problem, is the task of estimating a value function from a given policy. equation 1.22 shows the formula to calculate and updates the state value function.

$$V_\pi(s) = \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a)[r + \gamma \times V_\pi(s')]. \tag{1.22}$$

Policy improvement is the task of improving the existing policy. Equation 1.23 outlines the update rule for the policy improvement step. In this algorithm, the policy is improved by behaving greedily in relation to the value function of the current policy.

$$\begin{aligned} \pi_(s) &= \arg\max_a \times q_\pi(s, a). \\ &= \arg\max_a \mathbb{E}[R_{t+1} + \gamma \times V_\pi S_{t+1} \mid S_t = s, A_t = a]. \\ &= \arg\max_a \sum_{s',a} p(s', r \mid s, a) \left[r + \gamma \times V_\pi(s')\right]. \end{aligned} \tag{1.23}$$

Finally, when we have multiple policies to choose from, we can use Policy Iteration to find the optimal policy. Policy evaluation and improvement steps are repeatedly iterated until convergence is achieved in the shape of optimal policy, which can maximize the rewards (Sutton & Barto, 2018b).

1.3.5.1 Q-Learning

Q-learning is a classical algorithm from the class of model-free algorithm however, it differs from the classical State-Action-Reward-State-Action (SARSA) algorithm because Q-learning is an off-policy method. Off-Policy refers to the fact that this algorithm uses a different policy for choosing an action (Behavior Policy) than the policy which is being estimated (Target Policy). The Q-learning algorithm stores the mapping between states and actions in the form of a table.

Figure 1.4 depicts the basic model of Q-learning algorithm. The update equation for Q- learning is given below.

$$Q(s,a) = Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right]. \tag{1.24}$$

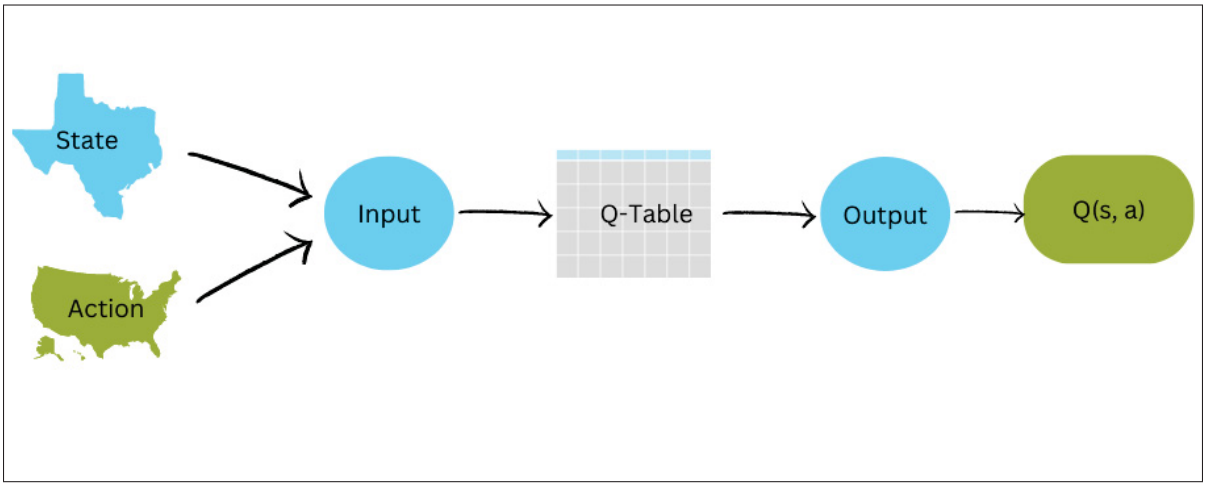

Figure 1.4 Q-Learning

1.3.5.2 Deep Q-Netwroks

It takes a lot of computational time and resources to store and access a large Q-table when there is a large input state and action space, so simple Q-learning does not perform well. In order to achieve better convergence, DNNs are usually combined with Q-learning. Instead of manually calculating each Q-value, the neural network approximates the Q-value by using a neural network combined with the action-value function. In the learning process, the DQN optimizes the weights Θ in order to minimize the error estimated by the loss function (given in equation 1.25). Error or loss is defined as the difference between predicted results and actual results. In addition to being computationally efficient, DNN approximations of Q-values are almost equal to true values.

$$L(\Theta) = ((r + \gamma \max_{a(t+1)} Q(S_{t+1}, a_{t+1}, \Theta^{target}) - Q(s_t, a_t, \Theta^{predicted}))^2. \tag{1.25}$$

In the above equation, the target Q value is $(r + \gamma \max_{a(t+1)} Q(S_{t+1}, a_{t+1}, \Theta^{target} t)$ and the predicted Q value is $Q(s_t, a_t, \Theta^{predicted})$

1.3.5.3 Double DQN (DDQN)

Standard DQN suffers from overestimation bias because the maximization operator ($\max_{a'}$) in the target calculation tends to select overestimated Q-values. (Van Hasselt, Guez & Silver, 2016) proposed Double DQN (DDQN) to mitigate this. DDQN decouples the action selection from action evaluation. The online network θ selects the greedy action, while the target network θ^- evaluates its value:

$$Y_t^{DDQN} = r + \gamma Q(s', \arg\max_{a'} Q(s', a'; \theta); \theta^-). \tag{1.26}$$

This simple modification significantly reduces overestimation and improves performance stability, a technique widely adopted in modern edge offloading schedulers.

1.3.5.4 Dueling DQN

The Dueling DQN architecture, proposed by (Wang *et al.*, 2016), modifies the neural network structure to better represent the underlying value functions. It recognizes that for many states, the value of the state itself is more important than the specific action taken.

The network splits into two streams: one estimates the scalar state-value $V(s)$ and the other estimates the advantage function $A(s, a)$. These are combined at the output layer:

$$Q(s, a) = V(s) + \left(A(s, a) - \frac{1}{|\mathcal{A}|} \sum_{a'} A(s, a') \right). \tag{1.27}$$

This architecture allows the agent to learn the state value function efficiently, which is particularly beneficial in multi-user MEC environments with massive action spaces.

1.3.5.5 Policy-Gradient Methods

Value based methods like DQN are limited to discrete action spaces. For problems with continuous action spaces (e.g., allocating specific power levels or fractional CPU frequencies), value-based methods require discretization, which leads to the curse of dimensionality (Qadeer & Lee, 2023). Policy gradient methods directly parameterize the policy $\pi_\theta(a|s)$ and optimize θ via gradient ascent on the expected return $J(\theta) = \mathbb{E}[G_0]$ (Sutton & Barto, 2018a).

1.3.6 Actor-Critic Architectures

Actor Critic methods combine the benefits of value-based and policy based approaches. They maintain two networks: an Actor $\pi_\theta(s)$ that decides which action to take, and a Critic $Q_w(s, a)$ (or $V_w(s)$) that evaluates the action produced by the actor. The critic reduces the variance of policy updates by replacing the stochastic return G_t with a value estimate (Sutton & Barto, 2018a).

1.3.6.1 Deep Deterministic Policy Gradient (DDPG)

DDPG is an off policy actor-critic algorithm designed specifically for continuous action spaces. It combines the deterministic policy gradient with the stability mechanisms of DQN (replay buffer and target networks) (Lillicrap *et al.*, 2015a; Qadeer & Lee, 2023).

- **Critic Update:** Minimizes the loss between the Q-value predicted by the critic and the target value $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$.
- **Actor Update:** Updates the policy parameters θ^{μ} using the chain rule to maximize the expected Q-value:

$$\nabla_{\theta^{\mu}} J \approx \frac{1}{N} \sum_{i} \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^{\mu}} \mu(s|\theta^{\mu})|_{s_i}. \tag{1.28}$$

DDPG employs soft updates for target networks ($\theta' \leftarrow \tau\theta + (1-\tau)\theta'$) to ensure stability, making it a popular choice for continuous resource allocation in MEC.

1.3.6.2 Twin Delayed DDPG (TD3)

Despite its success, DDPG is prone to value overestimation, similar to Q-learning, which can degrade policy performance. (Fujimoto, Hoof & Meger, 2018a) introduced TD3 to address this. TD3 incorporates three key improvements:

1. **Clipped Double Q-Learning:** TD3 uses two critic networks (Q_1, Q_2) and uses the minimum value ($\min(Q_1, Q_2)$) to calculate the target, effectively penalizing overestimation.
2. **Target Policy Smoothing:** Gaussian noise is added to the target actions during training. This acts as a regularizer, preventing the policy from exploiting sharp peaks in the Q-function.
3. **Delayed Policy Updates:** The actor and target networks are updated less frequently than the critic (e.g., one actor update for every two critic updates), allowing the value estimates to settle before the policy is updated.

1.3.6.3 Proximal Policy Optimization (PPO)

For on policy learning, ensuring monotonic improvement without destabilizing the policy is challenging. PPO (Schulman, Wolski, Dhariwal, Radford & Klimov, 2017a) addresses this by employing a trust region method that constrains the policy update. PPO uses a clipped surrogate objective function:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[\min \left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right]. \tag{1.29}$$

where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ is the probability ratio and ϵ is a hyperparameter determining the clip range (e.g., 0.2). This objective prevents the new policy from deviating too far from the old policy, ensuring stable and reliable convergence. PPO is widely adopted in modern edge networking for its balance of sample efficiency, stability, and ease of implementation compared to complex second order method (Gholipour, De Assuncao, Agarwal, Gascon-Samson & Buyya, 2023; Dong *et al.*, 2024).

1.4 Federated Learning (FL) for Distributed Edge Intelligence

As discussed in previous sections, DRL offers a powerful paradigm for solving the at hand NP-hard combinatorial optimization problem. (Sun *et al.*, 2024). However, traditional DRL approaches generally assume a centralized training architecture, where all training data (e.g., user task profiles, DAG topologies, and mobility traces) is collected at a central server to train a global agent. In the context of massive IoT, this centralized paradigm faces two important bottlenecks:

1. **Privacy Leakage:** Transmitting the exact topological structure of an application (i.e., its DAG representation) to a central orchestrator raises severe privacy concerns. Even without accessing raw payload data, exposing a DAG's specific execution times and dependency patterns can inadvertently reveal proprietary algorithmic logic and allow malicious actors to infer sensitive user behaviors through execution pattern analysis (He, Zhang & Lee, 2020).

2. **Communication Overhead:** In the training phase of the learning agents, the frequent transmission of high-dimensional raw data (e.g., state vectors and DAG adjacency matrices) to central server consumes scarce uplink bandwidth, resulting in network congestion and increasing transmission latency (Zhang *et al.*, 2024b).

To address these challenges, FL, first proposed by Google (McMahan *et al.*, 2017b), has emerged as a decentralized learning framework. The core principle of FL is to bring the code to the data, rather than the data to the code. In an FL-enabled MEC system, edge devices or Small Base Stations (SBSs) act as distributed clients that train models locally using their private data. They share only the model updates (gradients or weight parameters) with a central aggregator, which synthesizes a global model. These aggregated global weights are then broadcast back to the clients, enabling each distributed edge device to immediately benefit from the collective training experience of the entire network. This approach decouples model training from direct access to raw data, enabling privacy-preserving distributed intelligence (Khan, Ali-Pour, Avgeris, Gascon-Samson & Leivadeas, 2025a).

1.4.1 Privacy-Preserving Distributed Training

The standard FL training process in a MEC environment operates in communication rounds $t = 1, 2, \ldots, T$. Let $\mathcal{K} = \{1, 2, \ldots, K\}$ denote the set of participating edge nodes (clients). Each client k possesses a local dataset $\mathcal{D}_k$ consisting of $n_k = |\mathcal{D}_k|$ samples. The goal is to minimize a global objective function $f(w)$, where w represents the global model parameters:

$$\min_{w} f(w) = \sum_{k=1}^{K} \frac{n_k}{n} F_k(w). \tag{1.30}$$

where $n = \sum_k n_k$ is the total number of samples across all clients, and $F_k(w)$ is the local objective function of client k:

$$F_k(w) = \frac{1}{n_k} \sum_{i \in \mathcal{D}_k} \mathcal{L}(x_i, y_i; w). \tag{1.31}$$

Here, $\mathcal{L}(\cdot)$ is the loss function.

The privacy-preserving nature of this framework stems from the fact that the raw data samples (x_i, y_i) never leave the local device k. Only the model weights w or the gradients ∇w are transmitted over the wireless channel. While this prevents direct data leakage, recent studies suggest that gradient inversion attacks could potentially reconstruct data from updates (Guo *et al.*, 2025).

However, for task offloading decisions (which are less semantically sensitive than raw images or text), the aggregation of weights from multiple users provides a strong layer of anonymity in secure aggregation contexts.

1.4.2 The Federated Aggregation Cycle

The complete FL lifecycle in an edge network is an iterative process executed over communication rounds $t = 0, 1, \ldots, T-1$. To minimize the global objective without centralizing data, the orchestration relies on a synchronous protocol consisting of four distinct mathematical stages per round:

1. **Global Model Broadcast (Downlink):** At the beginning of round t, the central server determines the set of participating clients $\mathcal{K}_t \subseteq \mathcal{K}$, which may consist of all available clients or only a subset, and broadcasts the current global model weights w_t. This step is critical as it synchronizes the starting point for all edge devices, allowing them to benefit from the collective knowledge aggregated in previous rounds.
2. **Local Training (Computation):** Each selected client $k \in \mathcal{K}_t$ initializes its local model $w_{t,0}^k \leftarrow w_t$. The client then minimizes its local empirical risk using Stochastic Gradient Descent (SGD) over its private dataset $\mathcal{D}_k$ for E local epochs. The local update rule at local step e with learning rate η is:

$$w_{t,e}^k = w_{t,e-1}^k - \eta \nabla \mathcal{L}(w_{t,e-1}^k; \xi). \tag{1.32}$$

where ξ is a mini-batch sampled from $\mathcal{D}_k$. After E epochs, the client obtains its new local model $w_{t,E}^k$, which we denote simply as w_{t+1}^k.

3. **Model Upload (Uplink):** The participating clients transmit their locally updated weights w_{t+1}^k back to the central server via the wireless uplink channel.
4. **Global Aggregation:** The central server synthesizes a new global model w_{t+1} using a specific aggregation algorithm, and the cycle repeats.

In the following section, we briefly explain the main aggregation methods used in FL.

1.4.2.1 Federated Averaging (FedAvg)

Proposed by McMahan *et al.* (2017b), FedAvg is the baseline aggregation protocol. It performs a weighted average of the local models based on the proportional size of each client's dataset. The global update rule is formulated as:

$$w_{t+1} = \sum_{k \in \mathcal{K}_t} \frac{n_k}{n_t} w_{t+1}^k. \tag{1.33}$$

where $n_t = \sum_{k \in \mathcal{K}_t} n_k$. While highly efficient under Independent and Identically Distributed (IID) data assumptions, FedAvg frequently suffers from weight divergence when applied to the highly skewed, Non-IID data typical of vehicular user profiles.

1.4.2.2 Federated Proximal (FedProx)

To address the statistical heterogeneity inherent in MEC, advanced algorithms such as FedProx (Li *et al.*, 2020) have been developed. FedProx modifies the local training phase by introducing a proximal regularization term to the local objective function $F_k(w)$. The modified local objective $h_k(w; w_t)$ is defined as:

$$\min_w h_k(w; w_t) = F_k(w) + \frac{\mu}{2} \|w - w_t\|^2. \tag{1.34}$$

where $\mu > 0$ is a tunable proximal parameter. This term actively penalizes local weights that deviate too far from the global model w_t during the E local epochs. By mathematically constraining the local updates, FedProx limits the impact of statistical heterogeneity and guarantees more stable convergence in volatile edge networks.

1.4.2.3 Stochastic Controlled Averaging (SCAFFOLD)

While FedProx mitigates client drift by mathematically restricting local updates, SCAFFOLD (Karimireddy *et al.*, 2020) addresses the drift variance in local gradients caused by Non-IID data using control variates. The algorithm maintains a global control variate c (representing the global update direction) and a local control variate c_k for each client k (representing the local gradient direction). During the local training phase, the standard SGD step is modified to dynamically correct the drift:

$$w_{t,e}^k = w_{t,e-1}^k - \eta \left(\nabla \mathcal{L}(w_{t,e-1}^k; \xi) - c_k + c \right). \tag{1.35}$$

By explicitly estimating and subtracting the divergence between the local and global data distributions at every step, SCAFFOLD significantly reduces the required communication rounds and achieve stable convergence.

1.4.2.4 Federated Normalized Averaging (FedNova)

In MEC environments, edge devices possess vastly different computational capacities. Consequently, fast devices may execute significantly more local SGD steps than slower devices within the same communication time window. Standard FedAvg implicitly over-weights clients that perform more updates, leading to objective inconsistency and a skewed global model. FedNova (Wang, Liu, Liang, Joshi & Poor, 2020b) elegantly solves this by normalizing the local model updates prior to aggregation. If client k performs a_k local steps, yielding an accumulated local update

$\Delta_t^k = w_t - w_{t+1}^k$, the FedNova global aggregation rule is formulated as:

$$w_{t+1} = w_t - \tau_{\text{eff}} \sum_{k \in \mathcal{K}_t} \frac{n_k}{n_t} \left(\frac{\Delta_t^k}{a_k} \right). \tag{1.36}$$

where τ_{eff} scales the effective global learning rate. By normalizing the updates by the inverse of the number of local steps a_k, FedNova mathematically guarantees that fast and slow devices contribute equitably to the global objective, making it exceptionally well-suited for asynchronous and heterogeneous IoT networks.

1.4.3 System Trade-offs

While FL effectively mitigates privacy leakage and the massive bandwidth consumption associated with raw data transmission, it introduces complex system-level trade-offs that differ significantly from Centralized Learning (CL).

1.4.3.1 Accuracy Degradation vs. Privacy

In a CL paradigm, the centralized agent computes gradients over the true global data distribution, allowing for optimal descent steps. In FL, especially with Non-IID data distributions across edge devices, the system experiences client drift (Zhao *et al.*, 2018). During local training, each client's model optimizes toward the local minimum of its specific dataset rather than the global minimum. When these divergent models are aggregated, the resulting global model often exhibits a drop in ultimate accuracy compared to a centrally trained counterpart. This accuracy gap is the fundamental theoretical cost of maintaining decentralized privacy.

1.4.3.2 Synchronization Frequency

A critical hyperparameter in FL is the synchronization frequency, governed by the number of local epochs relative to the total global communication rounds. This dictates the trade-off between computation and communication (Wang *et al.*, 2019b).

1.4.3.3 Frequent Synchronization

Executing fewer local epochs before aggregation tightly couples the local models to the global objective, drastically reducing client drift and requiring fewer total round to reach target accuracy. However, frequent synchronization exponentially increases the number of uplink/downlink transmissions, resulting communication overhead and network latency.

Consequently, deploying FL in intelligent MEC systems requires dynamic optimization strategies to balance local computational resources against communication overhead, ensuring model convergence within the strict constraints of the environment.

1.5 Vehicular Edge Computing (VEC) and Mobility

The integration of MEC with Vehicular Ad-hoc Networks (VANETs) has given rise to a specialized paradigm known as VEC (Liu, Chen, Pei, Maharjan & Zhang, 2021c; Zhang, Letaief *et al.*, 2023). As vehicles evolve into computers on wheels, equipped with sophisticated onboard units (OBUs) and diverse sensors (LiDAR, cameras, RADAR), they generate massive volumes of data that require real-time processing (Zhang *et al.*, 2023). While MEC brings computational resources closer to these mobile users, the unique characteristics of the vehicular environment specifically high-speed mobility and dynamic network topology introduce complex challenges that distinguish VEC from standard static edge computing (Li *et al.*, 2024c).

1.5.1 Characteristics of VEC Environments

VEC environments are characterized by harsh communication conditions and stringent QoS requirements. Understanding these characteristics is essential for designing robust task offloading frameworks.

1.5.1.1 High Mobility and Dynamic Topology

The most defining feature of VEC is the high velocity of vehicles, which can range from stationary such as in traffic jams to very high speeds such as on highways. This mobility causes the network topology to change rapidly and unpredictably. The wireless channel conditions between the vehicle and the RSU are subject to fast fading and Doppler shifts, leading to unstable connections and varying data rates. Consequently, the connection window that is, the duration for which a vehicle remains within the communication range of a specific RSU is often short (e.g., a few seconds) (Li *et al.*, 2024c).

1.5.1.2 Heterogeneous Computing Resources

The VEC ecosystem typically comprises three layers of computing resources, forming a hybrid computing architecture (Liu *et al.*, 2021c; Tariq *et al.*, 2024):

- **Local Computing:** The vehicle's own On-Board Unit (OBU), which has limited battery and processing power but zero transmission delay.
- **Edge Computing (V2I):** Fixed RSUs deployed along the road, offering significant computational capacity but subject to transmission latency and coverage constraints.
- **Vehicular Cloudlets (V2V):** Clusters of nearby vehicles sharing their idle resources. While cost-effective, V2V links are highly volatile due to the relative motion between vehicles (Tariq *et al.*, 2024).

1.5.1.3 Delay-Sensitive and Computation-Intensive Workloads

Modern vehicular applications, such as autonomous driving path planning, cooperative perception, and high definition map updates, are both computation intensive and delay sensitive. For instance, safety-critical applications often require URLLC with end-to-end delays below 10 ms. The resource-constrained OBU often cannot meet these demands alone, necessitating efficient offloading strategies (Zhang *et al.*, 2023).

1.5.2 Handover Management and Service Migration

In a static MEC scenario, a user connects to a BS and remains associated for a long duration. In VEC, the high speed of vehicles necessitates frequent handovers between RSUs (Ali *et al.*, 2025). If a computation task offloaded to an RSU is not completed and the result returned before the vehicle leaves the RSU's coverage area, the task fails, leading to severe performance degradation (da Costa *et al.*, 2023).

1.5.3 Sequence Modeling for Trajectory Prediction

Reliable task offloading in VEC is fundamentally dependent on the ability to predict the vehicle's future location and the duration of connectivity (coverage window) with the RSU. If the system can accurately forecast that a vehicle will leave the RSU coverage in t_{out} seconds, it can reject tasks that require time $T > t_{out}$ or proactively trigger migration (Alotaibi *et al.*, 2024; Zhao *et al.*, 2025).

1.5.3.1 Limitations of Recurrent Neural Networks (RNNs)

Historically, mobility prediction has been treated as a time-series forecasting problem. RNNs and their variants, LSTM and Gated Recurrent Units (GRU), have been the standard tools (Al-Molegi *et al.*, 2025). LSTMs process data sequentially, maintaining a hidden state to capture temporal dependencies. However, RNNs suffer from two major limitations in the context of VEC (Al-Molegi *et al.*, 2025):

1. **Long-Range Dependencies:** As the prediction horizon increases (e.g., predicting the trajectory for the next steps), LSTMs struggle to retain information from earlier timesteps due to the vanishing gradient problem, making them less effective for long-term planning (Al-Molegi *et al.*, 2025).
2. **Sequential Processing:** RNNs cannot process time-steps in parallel, which limits their training efficiency and ability to model complex, non-linear traffic interactions (e.g., sudden lane changes or braking caused by distant traffic) (Kim *et al.*, 2025).

1.5.3.2 Transformer Models and Self-Attention

To overcome these limitations, this thesis advocates for the use of Transformer architectures (Vaswani *et al.*, 2017). Originally designed for NLP, Transformers rely entirely on Self-Attention mechanisms rather than sequential recurrence (Kim *et al.*, 2025). The self-attention mechanism allows the model to weigh the significance of different input positions relative to others, regardless of their distance in the sequence. This enables the Transformer to capture long-range dependencies in vehicle trajectories more effectively than LSTMs. Transformers process the entire sequence of historical positions simultaneously, allowing for faster inference and training (Vaswani *et al.*, 2017; Al-Molegi *et al.*, 2025).

1.6 Literature Review and Research Gaps

In this section, we present a detailed analysis of state-of-the-art research, highlighting related work and identifying key research gaps in the context of MEC and VEC. This section is structured by categorizing relevant literature according to each of our contributions.

Since the thesis follows an article based structure, it is important to note that each of the subsequent technical chapters (Chapters 3, 4, and 5) includes its own dedicated and detailed related work section, specifically tailored to the respective methodologies of EMDTORA, FEDORA, and our trajectory-aware VEC framework. The objective herein is to provide a broader, updated literature review. This section synthesizes the most recent state-of-the-art advancements to establish the broader research gaps addressed by this thesis.

1.6.1 Dependent Task Offloading in MEC

Most previous studies on MEC offloading focused on optimizing computational resource allocation to enhance latency or system throughput for independent, atomic tasks. However, these studies did not consider task dependencies. Several recent studies (Li *et al.*, 2024b; Chen, Cao, Sahni, Jiang & Liang, 2024d; Ye, Li, Gao & Wen, 2025) addressed dependent task scheduling in multi-user MEC networks. For instance, the authors in (Chen *et al.*, 2024d)

proposed a dependency-aware reinforcement learning approach for dynamic task offloading in edge computing. Similarly, (Ye *et al.*, 2025) introduced a Hierarchical Deep Q-Network (HDQN) framework that simultaneously manages service placement and task offloading for dependent tasks. Furthermore, (Zhang *et al.*, 2024e) utilized a MDP to optimize execution sequences based on subtask priorities in end-edge-cloud collaboration scenarios. To specifically address the complex topology of these applications, recent research has begun utilizing GNNs. The authors in (Wang *et al.*, 2024a) proposed a task scheduling method based on GAT and DRL to minimize the makespan of user tasks represented as DAGs. Their performance analysis assumes that the GAT encoder can be adequately pretrained using unsupervised learning to stabilize the multi-discrete action DRL scheduler. Similarly, (Wu *et al.*, 2024b) introduced a Graph Convolutional Network (GCN) augmented DRL model for resource-limited edge scenarios, while (Liu *et al.*, 2024b) formulated a soft cooperation offloading method in MEC-enabled IoT to maximize the utility of DAGs. In another relevant study (Xie, Cui, He & Guizani, 2024), the authors constructed a task-priority-transformer dependent task offloading (TPTDTO) method with refined priority features.

However, the studies reviewed above generally address single-objective optimization or lack comprehensive energy-delay tradeoff models under strict multi-user interference constraints. Moreover, methods that flatten DAG structures into standard neural networks (Li *et al.*, 2024b) are inefficient for complex applications, leading to suboptimal scheduling where bottleneck tasks are starved of resources. In contrast to the aforementioned studies, our first contribution, presented in Chapter 2, formulates the multi-user dependent task offloading problem as a multi-objective optimization problem. To the best of our knowledge, EMDTORA (Khan, Avgeris, Gascon-Samson & Leivadeas, 2025c) is the first work to deeply integrate GAT with a continuous DRL actor-critic framework to explicitly embed complex non-Euclidean DAG topologies into the agent's decision-making process, successfully balancing energy consumption and delay minimization under dynamic network conditions.

1.6.2 Federated Reinforcement Learning under Statistical Heterogeneity

Recent studies have explored the potential of FL to enhance privacy and scalability in edge networks. Specifically, several previous studies have analyzed FL convergence, communication efficiency, and decentralized DRL training (Hasan *et al.*, 2025; Chen *et al.*, 2024e; Zhang, Zhao, Liu, Gao & Xu, 2024c; Kim *et al.*, 2024). For instance, in (Chen *et al.*, 2024e), the authors examined a FL framework with experience-driven model migration in heterogeneous edge networks to alleviate the influence of Non-IID issues. Furthermore, the authors in (Zhang *et al.*, 2024c) studied a leader FL optimization approach using DRL for distributed satellite edge intelligence. In another relevant study (Kim *et al.*, 2024), the authors analyzed a federated reinforcement learning framework via a committee mechanism for resource management in 5G networks, ensuring reliable aggregation of local gradients. A critical bottleneck in edge-based FL is the presence of Non-IID data. The study in (Ahmed *et al.*, 2025) highlighted this challenge, proposing a Deep Deterministic Policy Gradient (D4PG) model based on a FL algorithm to adjust the participation of dynamic user equipment under extreme Non-IID datasets. Moreover, (Xiao *et al.*, 2024b) focused on federated DRL for task offloading in MEC-enabled heterogeneous networks, where each user device optimizes offloading decisions locally. Additionally, (Jia *et al.*, 2025) investigated a federated multi-agent reinforcement learning approach for age-of-information (AoI) minimization in UAV-enabled systems.

While these studies optimized FL frameworks to improve network performance, they largely overlooked the joint optimization of discrete and continuous action spaces in distributed reinforcement learning under extreme statistical heterogeneity. MEC applications require simultaneous decision-making for discrete actions (e.g., target server selection) and continuous actions (e.g., CPU frequency allocation). The aforementioned works either assumed simplified discrete action spaces or failed to address the severe training instability that occurs when hybrid-action DRL agents are subjected to Non-IID data. However, in our model, we consider a hybrid action space under strict Non-IID conditions. In our second contribution, presented in Chapter 4, we propose a comprehensive federated ensemble framework FEDORA (Khan *et al.*, 2025a). While previous studies suffered from unmitigated client drift, our approach integrates

TD3 for continuous allocation with multi-head DQNs for discrete decisions, fortified by the FedProx aggregation algorithm to ensure highly stable convergence.

1.6.3 Proactive Mobility Management in Vehicular Edge Computing

Recent research has focused on supporting computation offloading in VEC networks, addressing challenges in intermittent connectivity, latency, and dynamic topologies (Zhao *et al.*, 2025; Wu, Jia, Pang & Zhao, 2024c; Li *et al.*, 2025; Zheng *et al.*, 2025). Several studies proposed optimization frameworks for VEC offloading using DRL and trajectory prediction (Zhao *et al.*, 2025; Li *et al.*, 2025; Wu *et al.*, 2025). In (Wu *et al.*, 2025), the authors aimed to minimize task offloading delay by proposing a Bi-LSTM-based model to predict vehicle trajectories. Their framework converted predicted grid areas into candidate execution servers to reduce the impact of prediction deviations. Similarly, (Zhao *et al.*, 2025) proposed an asynchronous DRL (A3C) approach combined with trajectory prediction to forecast the s a vehicle is likely to encounter, boosting task completion rates. Furthermore, the study in (Li *et al.*, 2025) investigated a Mobility-Aware Cooperative Multi-Agent (MCMA) DRL approach for task migration-assisted computation offloading. The authors incorporated an Informer-based multi-step vehicular trajectory prediction module to enhance collaboration among servers. Additionally, in (Zheng *et al.*, 2025), the authors proposed a cloud-edge-vehicle framework leveraging a spatio-temporal multi-head self-attention LSTM model for optimized task offloading. Other studies, such as (Chen *et al.*, 2024b) and (Zhang *et al.*, 2024a), focused on prioritized incentive-based offloading and trust-based secure task offloading using digital twins, while (Park *et al.*, 2025) introduced a practical online framework satisfying hard deadlines in VEC systems. Furthermore, (Liu *et al.*, 2025) proposed an intelligent task offloading algorithm based on a meta-reinforcement learning framework to address dynamic network conditions.

Compared to the above studies, most previous works focused on simple recurrent models (e.g., LSTMs) that struggle with long-range sequence dependencies. In addition, the predictive models used in the aforementioned studies are often insufficient for scheduling multi stage dependent DAGs across multiple RSUs without stalling the application workflow. To address

these gaps, our third contribution, presented in Chapter 5 (Khan, Avgeris, Ali-Pour, Gascon-Samson & Leivadeas, 2025b), proposes a comprehensive trajectory-aware task offloading framework. While previous studies focused on reactive handovers or independent tasks, our approach unifies a state-of-the-art mobility-aware Transformer with a GAT-enabled DRL agent to proactively derive RSU coverage intervals, thereby eliminating task orphaning and drastically enhancing service reliability for highly mobile users.

CHAPTER 2

EMDTORA: ENERGY-AWARE MULTI-USER DEPENDENT TASK OFFLOADING AND RESOURCE ALLOCATION IN MEC USING GRAPH-ENABLED DRL

Sangrez Khan[1], Marios Avgeris[2], Julien Gascon-Samson[1], and Aris Leivadeas[1]

[1] Department of Software and IT Engineering, École de Technologie Supérieure, 1100 Notre-Dame Ouest, Montréal, Québec, Canada H3C 1K3
[2] Informatics Institute, Faculty of Science, University of Amsterdam (UvA), Amsterdam, The Netherlands

Paper published in *IEEE Transactions on Green Communications and Networking*, December 2024

Abstract

The dawn of the 5G/6G networking era has led to the widespread adoption of Multi-Access Edge Computing (MEC), a paradigm shift that brings computational resources at the network's edge to enhance device performance and longevity. Additionally, the proliferation of Internet of Things (IoT) has facilitated the development of complex multi-user, multi-edge server environments. In these settings, the interdependence of application tasks makes computational offloading and resource allocation decision-making challenging, but crucial for optimizing energy efficiency. As a response, in this paper, we propose an Energy-Aware Multi-user Dependent Task Offloading and Resource Allocation (EMDTORA) scheme for IoT-MEC infrastructures. First, we formulate an offline, task offloading, multi-objective optimization problem that aims to minimize the user devices' energy consumption and experienced delay under given constraints. Given the NP-hardness of the problem, we devise an online framework that combines a Graph Attention Networks (GAT)-based mechanism, which captures the in-depth dependency structure of the applications, with an actor-critic off-policy Deep Reinforcement Learning (DRL) algorithm to approximate the optimal solution. Through extensive simulation we highlight the potency of the proposed scheme, as EMDTORA outperforms various baselines by successfully balancing the trade-off between energy consumption and delay minimization, under dynamic network conditions.

2.1 Introduction

In recent years, advancements in user equipment and communication technologies have given rise to innovative immersive applications (Mao *et al.*, 2017a). As these applications grow in complexity, so does the demand for computational resources to support their advanced functionalities. Despite the enhanced capabilities of the new devices, they still suffer from increased energy consumption and latency when executing compute-hungry applications. The initial response of offloading these tasks to Cloud Computing infrastructures gradually gave way to the Multi-access edge Computing (MEC) paradigm. This shift occurred mainly because the delay imposed by the distant cloud was deemed ineffective for meeting real-time Quality of Service (QoS) requirements (Sabella *et al.*, 2019). MEC revolutionizes the way we approach computational offloading by bringing computing and storage resources at the edge of the network. This shift significantly mitigates network congestion, reduces latency, and enhances energy efficiency by facilitating in-proximity processing (Saeik *et al.*, 2021a). The offloading of tasks across multiple devices or edge servers enables us to balance the computational load more effectively, ensuring that no single device becomes overloaded and energy depleted. This distribution leads to more efficient resource utilization.

However, an often-overlooked factor during computational offloading is the dependency between the tasks that comprise modern applications. In this context, formulating a task offloading policy requires meticulous consideration of which tasks should be offloaded to MEC servers and which should be executed on local devices, while taking into consideration their interdependencies. This decision should be also guided by task profiles (e.g., data processing requirements and CPU usage for data execution), along with the running state of the MEC infrastructure (Wang *et al.*, 2021a). This emerging problem is called *dependent task offloading* and is NP-hard (Wang, Wang, Huang, Song & Qin, 2020a). Many studies during the last few years have proposed heuristic approaches to address it (Lee, Ko, Kim & Pack, 2020), (Shu, Zhao, Han, Min & Duan, 2019), however, their performance heavily relies on accurate analytical modeling. As a result, these approaches cannot adapt easily to environmental dynamicity and/or emerging requirements and constraints introduced by the evolution of MEC and Internet of Things (IoT)

(Sundar & Liang, 2018), as this necessitates re-approaching the solution anew. Given the strict modern applications' QoS requirements and the MEC infrastructure's complexity, this can be an unrealistic, time-consuming assumption.

Deep Reinforcement Learning (DRL), i.e., the convergence of Deep Neural Networks (DNN) and Reinforcement Learning (RL), stands out as a promising solution for such complex sequential decision-making problems (Chen, Xing, Xiao, Xu & Tao, 2021b). Its efficacy in handling large state and action spaces positions DRL-based mechanisms as efficient alternatives for the task offloading problem (Zhao, He, Huang & Li, 2022), (Nieto, de la Iglesia, López-Novoa & Perfecto, 2023). Nonetheless, and as stated above, most existing solutions tend to overlook the inherent task dependencies or just consider a simple linear task interdependency (Wang *et al.*, 2021a). Only a few recent studies address the dependency structure of applications, utilizing DAGs to model the underlying task relationships (Wang, Zhao, Liu & Kato, 2019a). However, these works exhibit certain limitations: 1) their applicability is primarily limited to single-user scenarios (Wang *et al.*, 2021a), (Xiao *et al.*, 2022), (Li, Gu, Qin & Han, 2023), 2) they consider either energy consumption or delay satisfaction as their objective, resulting in suboptimal offloading decisions (Tang & Wong, 2020), and 3) they do not adequately account for the dynamic nature of network conditions (Liu *et al.*, 2023a), (Zhao, Xu, Zhao, Qiao & Huang, 2021), (Zhou, Chen, Jiang & Wu, 2022a). Addressing these limitations is crucial for providing a robust and scalable dependent task offloading solution. Additionally, the structure of the interconnection of the application tasks has to be considered. In this regard, Graph Attention Networks (GATs) (Veličković,Petar, 2017), due to their attention mechanism, can efficiently transfer data through the dependencies and focus on the areas of greater impact, therefore enhancing the ability of capturing the long-term, in-depth features and structural information of graphs; these encoded features and interdependencies can be exploited for informed offloading decisions (Shao *et al.*, 2021).

To fill these gaps, we combine GAT and DRL into a novel multi-user, multi-edge, dependent task computational offloading and resource allocation scheme. The proposed *Energy-Aware Multi-user Dependent Task Offloading and Resource Allocation* (*EMDTORA*) mechanism

combines both energy and delay minimization criteria in a multi-objective optimization problem for resource-constrained MEC environments. Energy consumption minimization on IoT devices is targeted directly through smart task execution decisions, while on the MEC infrastructure it is considered indirectly through optimal resource allocation. In detail, in EMDTORA, the application tasks are modelled as DAGs, where vertices and edges denote tasks and their dependencies, respectively. A GAT-based task embedding scheme is used to capture the tasks' inherent dependencies and features, allowing to transform the DAGs into useful vector embeddings. Due to its NP-hardness, the offline multi-objective optimization problem is re-formulated as an online Markov Decision Process (MDP). Since the dual nature of the joint problem of task offloading (discrete) and resource allocation (continuous) results in a hybrid action space, to solve the MDP we devise a multi-modal, Twin Delayed Deep Deterministic Policy Gradient (TD3) algorithm (Fujimoto, Hoof & Meger, 2018b). This technique provides our mechanism with the required adaptability to adjust to dynamic network conditions. To improve the training efficiency, we complement the mechanism with an experience replay buffer. The main contribution of our work is threefold:

- In contrast to most prior works that generally have optimized either energy consumption or delay independently, EMDTORA addresses both objectives jointly. This is particularly vital for practical real-world IoT applications where these two factors play a major role.
- While DRL has been applied in MEC, the integration of GAT for task dependency modeling is new. Hence, conventional approaches may neglect the complexity of scenarios with intense dependency among tasks. The use of GAT helps our framework to concentrate more on the critical dependency between tasks, thereby making a more informed decision on offloading.
- Most existing works have focused on single-user scenarios or have assumed static network conditions. In contrast, EMDTORA is evaluated in multi-user environments with varying network conditions, showing its scalability and robustness.
- Our framework not only takes decisions about the offloading of tasks but also dynamically allocates CPU resources, in consideration of the prevailing network conditions and the needs of the task.

- We evaluate the efficiency and efficacy of EMDTORA through extensive simulations under various representative scenarios and benchmarks against baselines from the literature. The results unequivocally show that the proposed framework outperforms the other approaches in terms of joint energy consumption and delay minimization.

The rest of the paper is structured as follows. Section 2.2 provides an overview of the related works. In Section 2.3, we present the system model and the formulation of the offline multi-objective optimization problem. Section 2.4 outlines the MDP formulation accompanied by the details of the GAT-based task embedding and TD3-based dependent task offloading and resource allocation algorithms. Section 2.5 presents the results of our simulations. Finally, our conclusions and plans for future work are summarized in Section 2.6.

2.2 Related Works

2.2.1 Optimization/heuristic-based approaches

Recently, attention around the dependent task offloading problem in MEC environments has been increasing. Optimization and heuristic-based techniques used in traditional computational offloading have been adapted to these new requirements. For example, in (Liu *et al.*, 2023a), the authors propose COFE, a framework that enables mobile devices to offload tasks with dependent constraints to a MEC-Cloud hybrid infrastructure. The authors first formulate the dependent task offloading problem as an average makespan minimization problem and then devise a heuristic ranking-based algorithm to solve it in real time. Although the simulations showcase satisfactory delay minimization, the energy consumption is not taken into consideration, and the network conditions are considered static. Similarly, (Zhao *et al.*, 2021) prove that the problem of dependent task offloading combined with service caching at the MEC side is NP-hard, while they propose a convex programming and a favorite successor algorithm, for a homogeneous MEC server setting. Energy consumption minimization is not considered in this work either.

In terms of dependent tasks, (An *et al.*, 2022) propose a joint dependent task offloading and resource allocation solution for IoT-enabled MEC environments. Here, the original nonconvex problem is first transformed into a convex one, and then a Golden Search-based algorithm is used to minimize energy consumption and delay. When network conditions allow, they prove that the derived policy will converge to the offline policy, however their technique is limited only to sequential dependency among tasks. In (Shen, Hu & Xia, 2022), a dependency-aware task offloading and service caching is proposed for VEC. This dual-nature problem is formulated as a mixed-integer non-linear programming (MINLP) problem and a dynamic programming-based solution is devised to minimize both energy consumption and task computation time. However, the resource allocation decisions are limited to the user/vehicle side.

2.2.2 RL-based approaches

Being a transformative approach for efficient dependent task offloading, reinforcement learning has inspired several recent works in the field. For instance, an intelligent dependent task offloading scheme for MEC is proposed by (Wang *et al.*, 2021a). The authors use a dependency-capturing sequence-to-sequence neural network for deriving the optimal offloading decision, with the goal to minimize delay and energy consumption. However, the proposed scheme exclusively addresses the task placement problem, neglecting resource allocation. Additionally, its applicability is limited to single-user scenarios and the resources at the edge are considered unlimited, which constitutes an impractical assumption. An actor-critic-based DRL solution for minimizing energy consumption and delay during the offloading of dependent tasks is proposed in (Chen *et al.*, 2021a). The authors here use a DNN for state embedding and an actor-critic network for the RL. Once more, the proposed solution does not consider the resource allocation problem nor addresses any dynamic wireless channel conditions. Likewise, in (Xiao *et al.*, 2022), the authors propose CODIA, an Actor-Critic-based, RL mechanism for dependent IoT task offloading which, however, can adapt to dynamic environments. Efficient resource allocation is also overlooked in this single-user oriented work.

Table 2.1 Comparison between related works on dependent MEC task offloading

Reference	**Users**	**Energy Min.**	**Delay Min.**	**Resource Alloc.**	**Adv. Dependency Apprehending**	**Dynamic Net. Conditions**
Liu *et al.* (2023a)	Multiple		✓			
Zhao *et al.* (2021)	Multiple		✓			
An *et al.* (2022)	Multiple	✓	✓	✓		✓
Shen *et al.* (2022)	Multiple	✓	✓	✓		
Wang *et al.* (2021a)	Single	✓	✓		✓	✓
Chen *et al.* (2021a)	Multiple	✓	✓		✓	
Xiao *et al.* (2022)	Single	✓	✓			✓
Zhou *et al.* (2022)	Multiple	✓		✓		
Qu *et al.* (2021)	Multiple	✓	✓			✓
Li *et al.* (2023)	Single	✓	✓	✓		✓
Liu *et al.* (2023b)	Multiple		✓			
Lin *et al.* (2022)	Multiple		✓		✓	✓
EMDTORA (Ours)	**Multiple**	✓	✓	✓	✓	✓

Zhou *et al.* (2022a) present a Digital Twin-enabled MEC architecture that jointly optimizes task offloading and service caching while considering task dependency and edge server collaboration.

The problem is formulated as a MINLP problem and is solved by a DRL-based algorithm, with a focus on reducing energy consumption. Delay minimization is not considered, and the task dependencies are limited to sequential ones. In (Qu, Wu, Li & Jiao, 2021) the authors propose the DMRO algorithm, which uses deep meta-reinforcement learning to make efficient offload decisions in the IoT and DNN computing. This approach overcomes the NP-hardness challenge of optimal dependent task offloading and achieves improved adaptability and speed while jointly minimizing energy consumption and delay in dynamic environments. Nonetheless, resource allocation consideration is left for future work. The work in (Li *et al.*, 2023) proposes a graph-based dependent task offloading scheme for the MEC-enabled Internet-of-Vehicles (IoV), which minimizes a weighted sum of delay-energy consumption. The combination of a DRL with an optimization module allows for fast adaptability and scalability in dynamic environments. However, this work does not capture the interdependency between the tasks in the same depth that our GAT-based approach does, and its applicability is limited to a single user. Liu *et al.* (2023b) propose a deadline violation ratio minimization computation offloading scheme with task migration and merging (DVRMO-MM). The framework uses a Deep Deterministic Policy Gradient (DDPG) mechanism to find the optimal task offloading policies under completion time constraints. However, the suggested study does not cater for energy consumption minimization or optimizing the allocation of resources, nor does it consider the in-depth task interdependencies. On the other hand, Lin, Lin, Chen & Cheng (2022), devise a DRL-based multi-task dependency offloading algorithm that uses a GAT to extract the dependency information. Specifically, they combine Long Short-Term Memory (LSTM) with Deep Q-Network (DQN) techniques to minimize the experienced delay. Despite the promising results, this work does not address resource allocation nor energy minimization.

2.2.3 Discussion

Table 2.1 provides a consolidated analysis of the literature works on dependent task offloading for MEC infrastructures, categorizing them based on the criteria discussed above. To the best of our knowledge, EMDTORA is the only multi-user, multi-edge server solution that deals with

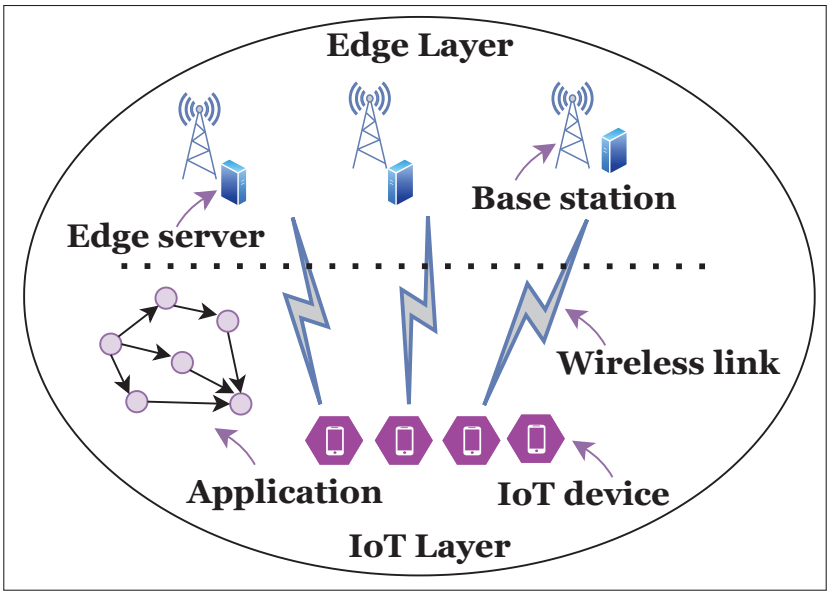

Figure 2.1 MEC Environment Overview

both the minimization of the energy consumption and the end-to-end application delay of IoT devices for jointly tackling dependent task offloading and resource allocation under dynamic network conditions, while considering the intrinsic interdependencies between the tasks.

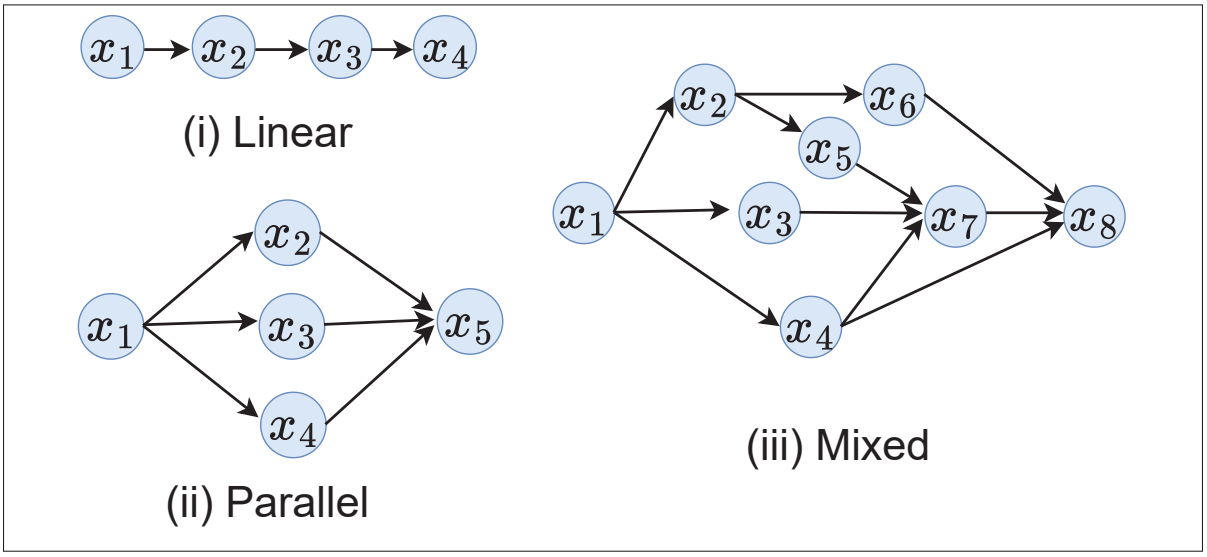

Figure 2.2 Task Interdependency Forms

2.3 System Model

We consider a MEC infrastructure comprising multiple edge servers, where various IoT devices are connected to the edge through base stations, as shown in Fig. 2.1. Let the set of users/devices be denoted by $\mathbb{N} = \{1, 2, \ldots, n, \ldots, |\mathbb{N}|\}$ and the set of MEC servers by $\mathbb{S} = \{1, 2, \ldots, s, \ldots, |\mathbb{S}|\}$. We use $s = 0$ to denote IoT device-related variables and constants. Each IoT device and MEC server has a maximum CPU frequency denoted as F_n^0 and F^s, respectively. Additionally, the IoT devices have an energy consumption limit for each application execution, E_n, for energy efficiency purposes. The inter-edge server transmission delay is considered negligible. Inspired from (Xiao *et al.*, 2022) and (Qian *et al.*, 2022), we assume that Orthogonal Frequency Division Multiplexing (OFDMA) (Li *et al.*, 2019) is employed as a multiple access method and the total bandwidth, B, is divided into OFDMA orthogonal subcarriers and each user is assigned to his own set of subcarriers.

The users execute computationally intensive applications comprised of multiple dependent tasks. The edges in a DAG represent the direction in which the data flows in the activities and they make sure that dependent activities are only allowed to start as soon as the relevant data is produced by its preceding activity. This interdependency can have various forms, typical examples of which are depicted in Fig. 2.2. For instance, a face recognition application consists of the following five dependent tasks: object acquisition, face detection, preprocessing, feature extraction and classification (Zhao *et al.*, 2021). For the sake of simplicity, we assume that at any given moment, each IoT device is limited to executing only one application. As a result, the same identifier notation n is used to represent both the application and the device, implying a one-to-one relationship between them during operation. We model each application as a DAG $V_n = (X_n, Y_n)$ which consists of tasks denoted by $X_n = \{x_{1\,n}, x_{2\,n}, \ldots, x_{i\,n}, \ldots, x_{|X_n|\,n}\}$. Y_n is the set of links representing the structural dependencies between the tasks. In detail, we assign a link from task $x_{i\,n} \in X_n$ to task $x_{j\,n} \in X_n$ if and only if data transmission occurs between them. Using these dependencies we introduce the application's *adjacency matrix*, which represents the structural relationships between the tasks. We define it as $Z_n = [\zeta_{1\,n}, \ldots, \zeta_{|X_n|\,n}]$, such that $\zeta_{i\,j} = 1$ if tasks $x_{i\,n}, x_{j\,n} \in X_n$ have a dependency among them, and 0 otherwise. Each application comes

with an execution QoS requirement, Ξ_n, designating the maximum end-to-end delay tolerance. Also, a task x_{in} is attributed a vector of resource requirements $h_{in} = [w_{in}, c_{in}]$ indicating the size of the task, and the CPU cycles required for its execution, respectively. Assuming that these requirements are the features of the application's tasks we define the application's *feature matrix* as $H_n = [h_{1n}, \ldots, h_{|X_n|n}]$. Additionally, each task's execution is characterized by a start time, η^s_{in}, and a completion time, β^s_{in}, where $s \in \mathbb{S} \cup \{0\}$ is the selected execution device/server. Due to the tasks' dependencies, a task can start executing only when all its predecessor tasks have been completed and the corresponding data have been transmitted to it. For simplicity, we assume that each IoT device can execute one task at each moment. Finally, without loss of generality, we assume that only one task from each device can be executed remotely on a specific server at each given moment. However, there is no restriction regarding parallel remote execution of tasks coming from different devices.

To accommodate the joint task placement and resource allocation we additionally introduce two binary variables: i) d^s_{in} to denote the placement decision for task $x_{in} \in X_n$ and ii) f^s_{in} to denote the CPU clock frequency tuning for said task. The former indicates whether the task is to be executed locally on-device, $d^0_{in} = 1$, or offloaded on edge server $s \in \mathbb{S}$, $d^s_{in} = 1$. The joint placement and resource allocation decision for all the tasks of the IoT devices is given as:

$$\rho = \{(d^s_{in}, f^s_{in}) \mid \forall i \leq |X_n|, \forall n \in \mathbb{N}, \forall s \in \mathbb{S} \cup \{0\}\}. \tag{2.1}$$

Bellow we describe the models used for the local on-device execution, the offloaded edge server execution and the energy consumption, while Table 2.2 summarizes the notation used.

2.3.1 On-device execution

We define the on-device execution time of task x_{in} as:

$$\gamma^0_{in} = c_{in}/f^0_{in}, \tag{2.2}$$

Table 2.2 Summary of Main Notations

Notation	Description
$\mathbb{N}$	Set of users/devices/applications
$\mathbb{S}$	Set of MEC servers
F_n^0, F^s	Max CPU freq. of device n and server s (GHz)
$V_n = (X_n, Y_n)$	DAG of application n = (tasks, dependencies)
$x_{in} \in X_n$	i^{th} task of application n
Z_n	Application n's adjacency matrix
E_n	Device n's energy limit (mJ)
Ξ_n	App n's max delay tolerance (ms)
$h_{in} = [w_{in}, c_{in}]$	i^{th} task's resource needs = [size, cycles] (kB, Hz)
H_n	Application n's feature matrix
$\eta_{in}^s, \beta_{in}^s$	i^{th} task's start and end time on device/server s (ms)
β_n	Application n's completion time (ms)
(d_{in}^s, f_{in}^s)	i^{th} task's placement decision and clock frequency
ρ	Placement/resource allocation for all tasks
γ_{in}^s	i^{th} task's exec. duration on device/server s (ms)
$\tau_{in}^{(d)}, \tau_{in}^{(u)}$	i^{th} task's downlink/uplink time (ms)
o_{in}	i^{th} task's output size
$\Psi_n^{(d)}, \Psi_n^{(u)}$	Device n's downlink/uplink rate ($Mbps$)
B_n	Device n's bandwidth ($Mbps$)
$P_{sn}^{(d)}, P_{ns}^{(u)}$	Downlink/Uplink transmission power between IoT device $n \in \mathbb{N}$ and MEC server $s \in \mathbb{S}$ (W)
$\mathcal{G}_{sn}, \mathcal{H}_{ns}$	Downlink/Uplink channel gain between IoT device $n \in \mathbb{N}$ and MEC server $s \in \mathbb{S}$ (dB)
$\epsilon_{in}^0, \epsilon_{in}^{(u)}$	On-device/Uplink energy consumption for IoT device $n \in \mathbb{N}$ for executing/offloading task x_{in} (mJ)
ϵ_n	Device n's total energy (mJ)
Ω	Objective function
$E_n^{(r)}$	Device n's residual energy (mJ)
O_n	Number of placed tasks for application n
$\mathcal{S}_t, \mathcal{A}_t, \mathcal{R}_t$	State, action, and reward at slot t
H_n'	Long-term task dependencies for app n
$\mathbb{B}$	Experience replay buffer
$\dot{\mathcal{R}}_t$	Expected cumulative reward at t
π	Placement policy/primary actor network
ϕ, θ	Actor/critic network weights
Λ	TD3 learning rate

where f^0_{in} is the CPU clock frequency of the IoT device $n \in \mathbb{N}$ allocated for this task. The start time η^0_{in} of the task is defined as:

$$\eta^0_{in} = \max_{j \in pre(i), s \in \mathbb{S}} \{(1 - d^0_{jn})\beta^0_{jn} + d^s_{jn}(\beta^s_{jn} + \tau^{(d)}_{jn})\}, \tag{2.3}$$

where $j \in pre(i)$ denotes a predecessor task and $\tau^{(d)}_{jn}$ is the time taken to download the output of the task to the local device, i.e., the downlink transmission duration, in case the predecessor task was offloaded at the edge. The latter is calculated as:

$$\tau^{(d)}_{in} = o_{in}/\Psi^{(d)}_n, \tag{2.4}$$

where o_{in} is the output length of task $x_{in} \in X_n$ that depends on h_{in}, and $\Psi^{(d)}_n$ is the downlink transmission rate given by:

$$\Psi^{(d)}_n = B_n \log_2(1 + \frac{P^{(d)}_{sn}\mathcal{G}_{sn}}{\sigma^2}). \tag{2.5}$$

B_n is the available bandwidth for this device, $P^{(d)}_{sn}$ is the downlink transmission power, $\mathcal{G}_{sn}$ is the downlink channel gain between the edge server $s \in \mathbb{S}$ and user device $n \in \mathbb{N}$, and σ^2 is the variance of the Additive White Gaussian Noise (AWGN). We note here that $\mathcal{G}_{sn} \in [-5db, -40db]$ is considered a stochastic variable to account for a realistic network environment modeling. Accordingly, the on-device completion time for task $x_{in} \in X_n$ is given by:

$$\beta^0_{in} = \eta^0_{in} + \gamma^0_{in}. \tag{2.6}$$

2.3.2 Edge server execution

The remote edge server execution time of the MEC offloaded task $x_{in} \in X_n$ is calculated as:

$$\gamma^s_{in} = c_{in}/f^s_{in}, \tag{2.7}$$

where $f_{i\,n}^{s}$ is the allocated CPU clock frequency of the edge server $s \in \mathbb{S}$ that undertakes the execution of the task. The start time $\eta_{i\,n}^{s}$ of the task is calculated as:

$$\eta_{i\,n}^{s} = \max_{j \in pre(i), s \in \mathbb{S}} \{(1 - d_{j\,n}^{0})(\beta_{j\,n}^{0} + \tau_{j\,n}^{(u)}) + d_{j\,n}^{s} \beta_{j\,n}^{s}\}, \tag{2.8}$$

where $\tau_{j\,n}^{(u)}$ is the time taken to offload the task to the edge server, i.e., the uplink transmission duration. The latter is given as:

$$\tau_{i\,n}^{(u)} = o_{i\,n} / \Psi_{n}^{(u)}, \tag{2.9}$$

where $\Psi_{n}^{(u)}$ is the uplink transmission rate given by:

$$\Psi_{n}^{(u)} = B_{n} \log_{2}(1 + \frac{P_{n\,s}^{(u)} \mathcal{H}_{n\,s}}{\sigma^{2}}). \tag{2.10}$$

$P_{n\,s}^{(u)}$ is the uplink transmission power and $\mathcal{H}_{n\,s}$ is the uplink channel gain. Similar to the downlink channel gain, $\mathcal{H}_{s\,n} \in [-5db, -40db]$ is also considered a stochastic variable in this range. Thus, the remote completion time for task $x_{i\,n} \in X_{n}$ offloaded on MEC server $s \in \mathbb{S}$ is calculated as:

$$\beta_{i\,n}^{s} = \eta_{i\,n}^{s} + \gamma_{i\,n}^{s}. \tag{2.11}$$

Finally, the completion time for an application $n \in \mathbb{N}$ is calculated by [1]:

$$\beta_{n} = \sum_{s=0}^{\mathbb{S}} d_{|X_{n}|\,n}^{s}\, \beta_{|X_{n}|\,n}^{s}. \tag{2.12}$$

2.3.3 Energy consumption

During local execution, the on-device energy consumption for task $x_{i\,n} \in X_{n}$ is calculated as:

$$\epsilon_{i\,n}^{0} = \kappa_{n}\, c_{i\,n}\, (f_{i\,n}^{0})^{2}, \tag{2.13}$$

[1] We remind that $s = 0$ refers to on-device execution.

where κ_n is the effective switch capacitance constant determined by the processor architecture of the local IoT device n. The total energy consumption ϵ_n at n is then calculated as the sum of the energy consumed during local execution and during uploading the output data:

$$\epsilon_n = \sum_{i=1}^{|X_n|} \Big\{ (1 - d_{i\,n}^0)\epsilon_{i\,n}^0 + \sum_{s=1}^{\mathbb{S}} d_{i\,n}^s \Big(\sum_{j \in \mathrm{pre}(i)} (1 - \sum_{s=1}^{\mathbb{S}} d_{j\,n}^s)\epsilon_{j\,n}^{(u)} \Big) \Big\}, \tag{2.14}$$

where $\epsilon_{j\,n}^{(u)}$ is the uplink energy consumption and is given as:

$$\epsilon_{j\,n}^{(u)} = P_{n\,s}^{(u)} \tau_{i\,n}^{(u)}. \tag{2.15}$$

In Eq. 2.14 when the placement decision for task $x_{i\,n}$ is on-device execution, then only the first part of the equation is considered. However, if the placement decision is to offload at a remote edge server $s \in \mathbb{S}$ and the predecessor task $x_{i\,n}$ was executed on-device then we only consider the uplink transmission energy. We also note that for receiving the data, we assume that energy is mainly consumed in the server/base station side which is out of this work's scope. Additionally, the IoT device energy consumption for receiving the data is considered negligible (Wang, Fang, Ding & Xiong, 2021b).

2.3.4 Problem formulation

Given the set of available MEC servers $\mathbb{S}$ and IoT devices $\mathbb{N}$, we formulate the problem of joint dependent task offloading with resource allocation and aim to find a feasible per task placement (on-device or remote) that minimizes the weighted sum of the IoT devices' energy consumption

and end-to-end experienced delay. The said problem can be formulated as follows:

$$\underset{\rho}{\text{maximize}} \quad \Omega = \sum_{n=1}^{\mathbb{N}} (\Gamma\ \epsilon_n + \ \beta_n), \tag{2.16a}$$

$$\text{subject to} \quad \epsilon_n \leq E_n, \quad \forall n \in \mathbb{N}, \tag{2.16b}$$

$$\beta_n \leq \Xi_n, \quad \forall n \in \mathbb{N}, \tag{2.16c}$$

$$d^0_{1,n} = 1, \quad \forall n \in \mathbb{N}, \tag{2.16d}$$

$$d^0_{|X_n|,n} = 1, \quad \forall n \in \mathbb{N}, \tag{2.16e}$$

$$\textstyle\sum_{s=0}^{\mathbb{S}} d^s_{i\,n} = 1, \quad \forall i \in [1, |X_n|],\ \forall n \in \mathbb{N}, \tag{2.16f}$$

$$\textstyle\sum_{i=1}^{|X_n|} \mathbb{1}\{t \in [\eta^0_{i\,n}, \beta^0_{i\,n}]\}\ f^0_{i\,n} \leq F^0_n, \quad \forall n \in \mathbb{N}, \forall t \in [1, T], \tag{2.16g}$$

$$\textstyle\sum_{n=1}^{\mathbb{N}} \sum_{i=1}^{|X_n|} \mathbb{1}\{t \in [\eta^s_{i\,n}, \beta^s_{i\,n}]\}\ f^s_{i\,n} \leq F^s, \quad \forall s \in \mathbb{S}, \forall t \in [1, T]. \tag{2.16h}$$

where $0 \leq \Gamma \leq 1$ is a coefficient that introduces a bias for the energy consumption against completion time minimization. In our case we consider a well-balanced objective between the two. Constraints (2.16b) and (2.16c) guarantee that the maximum energy consumption and QoS tolerances of each device and application are not violated, respectively. Constraints (2.16d) and (2.16e) limit the first and last task of each application to be executed at the device that initiated them. Constraint (2.16f) ensures that each task will be executed in only the local device or one of the MEC servers. Finally, constraints (2.16g) and (2.16h) make sure that the available CPU resources are not oversubscribed at any given moment; here, $\mathbb{1}\{\cdot\}$ is the indicator function which is equal to 1 if the enclosed condition is satisfied and 0 otherwise. The indicator function is checked for every time instance t, between the start ($\eta^0_{i\,n}$) and end time ($\beta^0_{i\,n}$) of a task i over the maximum time horizon T for placing the $\mathbb{N}$ applications.

2.4 Multi-user Dependent Task Offloading and Resource Allocation

The problem in Eq. (2.16a) is an NP-hard MINLP due to the integer offloading decision variables d, continuous resource allocation variables f, and the recursive structure for calculating completion times β. Unlike binary offloading decisions $(0, 1)$, our formulation uses a vector where each task is either executed locally or offloaded to one of the available MEC servers $\mathbb{S} = \{1, 2, \ldots, s, \ldots, |\mathbb{S}|\}$, which significantly increases the complexity. The resource allocation is a continuous vector, further complicated by the task dependencies and the dynamic network conditions introduced by the stochasticity of the channel gains $\mathcal{G}$ and $\mathcal{H}$. Traditional algorithms struggle with such dynamic environments (Li *et al.*, 2023), while finding a solution within a polynomial runtime is practically infeasible. Recent advancements in model-free reinforcement learning offer a viable alternative for approaching such problems. DRL excels by learning adaptable policies without re-solving the problem for each change. Moreover, DRL with GATs effectively captures non-linear task dependencies within DAGs, ensuring more precise offloading decisions.

To this end, we first formulate the problem as a model-free MDP, split into two phases and then propose two complementary to each other mechanisms to solve it efficiently. The first mechanism implements a GAT-based task embedding scheme that captures the tasks' features and long-term, in-depth dependencies to transform the application DAGs into useful vector embeddings for the second mechanism. The second mechanism is a TD3-based algorithm that solves the online MDP representation of the optimization problem. The two mechanisms comprise EMDTORA, an approach that is adaptive enough to handle uncertainties and still provide effective dependent task placement policies within the given context. EMDTORA uses a two-tier mechanism that separates the GAT training from the DRL model training. This separation enhances the training phase and improves convergence stability, as DRL does not affect GAT learning and vice versa. Moreover, the two-step method simplifies optimization and reduces computational costs, making it suitable for resource-limited environments.

2.4.1 Markov Decision Process formulation

A typical MDP is defined as a tuple that encompasses five critical elements: $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma\}$. Here, $\mathcal{S}$ denotes the *states* of the environment, encapsulating all possible scenarios that a solver agent might encounter. The set of *actions* available to the agent is represented by $\mathcal{A}$. The *transition probabilities* $\mathcal{P}$, quantify the probability of transitioning from one state to another, given a specific action. The *rewards* received as results of the actions are denoted by $\mathcal{R}$ and γ represents the discount factor, which modulates the importance of future rewards (Sutton & Barto, 2018b). The decision-making in this process is considered slotted in timeslots $t \in \{0, 1, 2, \ldots\}$. As the state in our case is high-dimensional and partly continuous, $\mathcal{P}$ cannot be accurately obtained, thus the MDP is simplified as a model-free process $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{R}\}$, with the following definitions:

2.4.1.1 State

At each decision-making time t, the agent observes the MEC system state. We define the system state as a combination of information regarding the IoT devices and the MEC servers, the application features and structure, as well as the wireless channel conditions:

$$\mathcal{S}_t = \{E_n^{(r)}, O_n, H_n, Z_n, \mathcal{G}_{s\,n}, \mathcal{H}_{n\,s} \mid \forall n \in \mathbb{N}, \forall s \in \mathbb{S}\}, \tag{2.17}$$

where $E_n^{(r)} = E_n - \epsilon_n$ is the residual energy for IoT device $n \in \mathbb{N}$ and O_n is a counter tracking the number of already placed tasks of application n. A state is terminal when $O_n = |X_n|, \forall n \in |\mathbb{N}|$. We should note here that eventually in our implementation, instead of merely using each application's features H_n and structural information Z_n derived from its DAG V_n, we will utilize a higher-level embedding H'_n that captures the long-term in-depth task dependencies through GAT-based embedding. More details on this are given in the following subsection.

2.4.1.2 Action

We define a valid action as the placement decision for the i^{th} task of each IoT device $n \in \mathbb{N}$. Since we are jointly tackling dependent task offloading and resource allocation, the action taken by the solver agent at timeslot t (the tasks are being allocated sequentially at each time slot, so $i = t$) is based on the decision tuple defined in Eq. (2.1):

$$\mathcal{A}_t = \{(d^s_{t\,n}, f^s_{t\,n}) \mid \forall n \in \mathbb{N}, \forall s \in \mathbb{S} \cup \{0\}\}, \tag{2.18}$$

The validity of $\mathcal{A}_t$ is guaranteed by enforcing constraints (2.16d)-(2.16h) during selecting an action.

2.4.1.3 Reward

After performing action $\mathcal{A}_t$ on state $\mathcal{S}_t$, the agent gets a feedback which directly determines its strategy. As our optimization objective is to minimize both the energy consumption and end-to-end delay of an application, we formulate the reward as the scaled improvement in the evaluation of the objective function Eq. (2.16a). To embed the remaining tolerance constraints, the reward is severely penalized if the action results in $E^{(r)}_n = E_n - \epsilon_n < 0$ or $\beta_n > \Xi_n$:

$$\mathcal{R}_t = \begin{cases} 0, & \text{if } \exists n \in \mathbb{N} : \big(\epsilon_n > E_n\big) \vee \big(\beta_n > \Xi_n\big), \\ \Delta(\Omega_t - \Omega_{t+1}), & \text{otherwise.} \end{cases} \tag{2.19}$$

where Δ is a scaling constant coefficient.

2.4.2 GAT-based task embedding

In the dependent task offloading problem, the interdependencies between tasks must be captured in depth to ensure the optimal allocation of resources and efficient execution of tasks. Graph Neural Networks (GNNs) (Kipf & Welling, 2016) have emerged as a key tool to aid this process by enabling the accurate representation and analysis of complex interconnection relationships

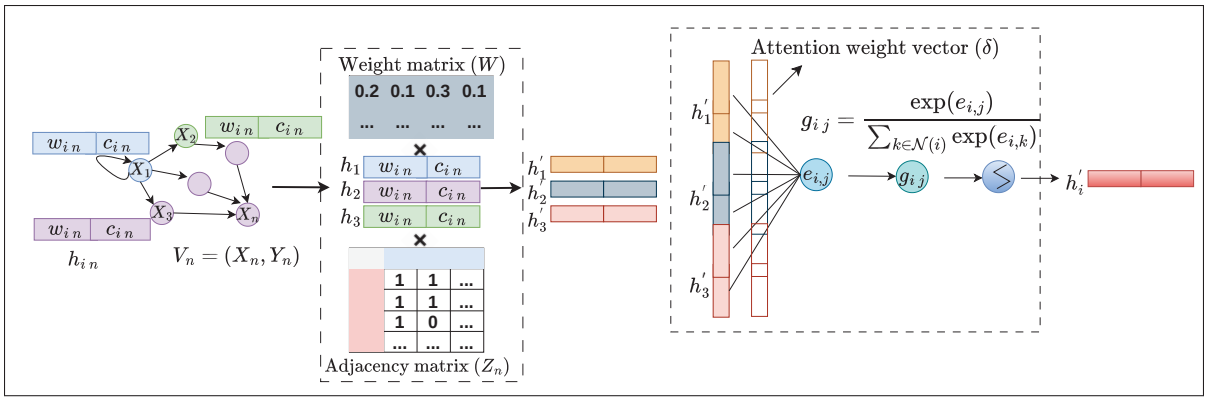

Figure 2.3 GAT-based higher-level task embedding (one application, single layer pass)

between the various tasks. One particularly effective approach pertinent to the domain of GNNs is the utilization of GATs (Veličković, 2017); this technique is especially relevant in contexts where tasks are represented as nodes in a DAG, with edges modeling their relationships. GATs leverage attention mechanisms to weigh the influence of neighboring nodes-tasks, thereby producing more refined and meaningful embeddings of them. Inspired by the above, we introduce a GAT-based task embedding mechanism that captures the tasks' features and long-term in-depth dependencies without requiring manual feature engineering. Our objective is to offload an entire application, which is modeled as a known DAG at the time of offloading. Unlike dynamic scenarios where tasks arrive over time, the known structure of a DAG allows GAT to process the entire structure and effectively generate embeddings capturing local and global dependencies within tasks. With respect to these embeddings, the offloading decision and resource allocation are guided to let the task be executed efficiently on the local device or on the edge servers. For better understanding, an overview of this mechanism for a single layer is depicted in Fig. 2.3 and the formal outline of the algorithm's steps are given in Algorithm 2.1. This mechanism relies on two inputs, the application's feature matrix H_n and the adjacency matrix Z_n. To enhance readability in this section, since we will be describing the mechanism's operation on the tasks of a single application $n \in \mathbb{N}$, we temporarily drop the subscript n from the notation. The goal of this mechanism is to enrich each feature vector $h_i \in H$ with knowledge of its dependant tasks features.

Algorithm 2.1 GAT-Based Higher-Level Task Embedding

Input: Graph dataset $\mathcal{D}$, no. of epochs E, learning rate α
Initialize Encoder and decoder parameters $\theta_{\text{enc}}, \theta_{\text{dec}}$ **for** *epoch* = 1 ***to*** E **do**
  for *each batch* $\mathcal{B}$ *in* $\mathcal{D}$ **do**
    Initialize total loss $\mathcal{L}_{\text{total}} = 0$
    for *each DAG* V_n *and feature matrix* H *in batch* $\mathcal{B}$ **do**
      $H' \leftarrow \text{Encoder}(V_n, H; \theta_{\text{enc}})$
      $\hat{H} \leftarrow \text{Decoder}(V_n, H'; \theta_{\text{dec}})$
      $\mathcal{L}_{\text{feature}} \leftarrow \frac{1}{N} \sum_{i=1}^{N} \left\| h_i - \hat{h}_i \right\|_2^2$
      $\mathcal{L}_{\text{structure}} \leftarrow -\frac{1}{|\mathcal{E}_n|} \sum_{(i,j)\in\mathcal{E}_n} \log \sigma(e_{ij})$
      $\mathcal{L}_{\text{total}} \leftarrow \mathcal{L}_{\text{total}} + (\mathcal{L}_{\text{feature}} + \mathcal{L}_{\text{structure}})$
    end for
    Compute gradients: $\nabla_{\theta_{\text{enc}},\theta_{\text{dec}}} \mathcal{L}_{\text{total}}$
    Update encoder parameters: $\theta_{\text{enc}} \leftarrow \theta_{\text{enc}} - \alpha \nabla_{\theta_{\text{enc}}} \mathcal{L}_{\text{total}}$
    Update decoder parameters: $\theta_{\text{dec}} \leftarrow \theta_{\text{dec}} - \alpha \nabla_{\theta_{\text{dec}}} \mathcal{L}_{\text{total}}$
  end for
end for
Output: Trained encoder and decoder parameters $\theta_{\text{enc}}, \theta_{\text{dec}}$

To this end, we use a graph convolutional layer to compute a set of new task features, $h'_i \in H'$, that captures the long-term relationships between said tasks. This enriched feature vector replaces H in the MDP state formulation of Eq. (2.17). The architecture of GAT supports information propagation across multiple hops inside the task dependency graph (Veličković, 2017). The formulation in Eq. 2.20 casts a single layer of GAT, which updates the embedding of task based on its neighbours.

$$h'_i = \Phi \sum_{j \in N_{(i)}} g_{ij} \, W \, h_j, \tag{2.20}$$

where, $h'i$ is the updated embedding of task i after aggregating information from its neighbors, $N_{(i)} \subseteq [1...|X|]$ is the neighborhood of tasks immediately dependent on task $x_i \in X$, including x_i. h_j is the current embedding of neighboring task j, W is a learnable weight matrix that transforms the neighbor's embedding into a suitable feature space. g_{ij} is a weight that specifies the "importance" of task x_j's features to x_i and Φ is the Leaky Rectified Linear Unit (ReLU) activation function. For the implementation, multiple such layers are stacked and each of these layers allows a task to aggregate information from its immediate neighbors; stacking them allows

the information to flow in from distant tasks. The task x_j's features $h_j \in H$ are multiplied by W to achieve the higher-level representation, a shared task-wise feature transformation that serves as an input to the graph convolutional layer. What is particularly interesting here, however, is the calculation of g_{ij}; this is where the GAT's attention mechanism comes into play. Let e_{ij} be the unnormalized *attention coefficient* across pairs of tasks $x_i, x_j \in X$ based on their features:

$$e_{ij} = \Phi(\delta^\top [W\, h_i \| W\, h_j]), \tag{2.21}$$

where, δ is the learnable attention weight vector and $\|$ is the vector concatenation operator. We incorporate the application's structure in the mechanism by only allowing task x_i to attend over its immediately dependent tasks $x_j \in X, \forall j \in N_{(i)}$. To make them comparable across different neighbourhoods, the attention coefficients are typically normalised using the softmax function:

$$g_{ij} = \frac{\exp(e_{ij})}{\sum_{k \in \mathbb{N}(i)} \exp(e_{ik})}. \tag{2.22}$$

A large g_{ij} suggests a close connection between the two tasks. The GAT training process consists of an encoder and a decoder. The model learns the optimal parameters' values by minimizing the feature and structural loss function that measures the difference between predicted embeddings and true dependencies. The feature loss $\mathcal{L}_{\text{feature}}$, ensures that the decoder efficiently generates the original node features and is given as:

$$\mathcal{L}_{\text{feature}} = \frac{1}{N} \sum_{i=1}^{N} \left\| h_i - \hat{h}_i \right\|_2^2, \tag{2.23}$$

where N is number of nodes, $h^{(n)}$ is the original feature vector of node n and $\hat{h}^{(n)}$ is the reconstructed node feature. The structure loss $\mathcal{L}_{\text{structure}}$ which is responsible for encouraging the node embedding to keep the original structure of the graph is given as:

$$\mathcal{L}_{\text{structure}} = -\frac{1}{|\mathcal{E}|} \sum_{(i,j) \in \mathcal{E}} \log \sigma(e_{i,j}), \tag{2.24}$$

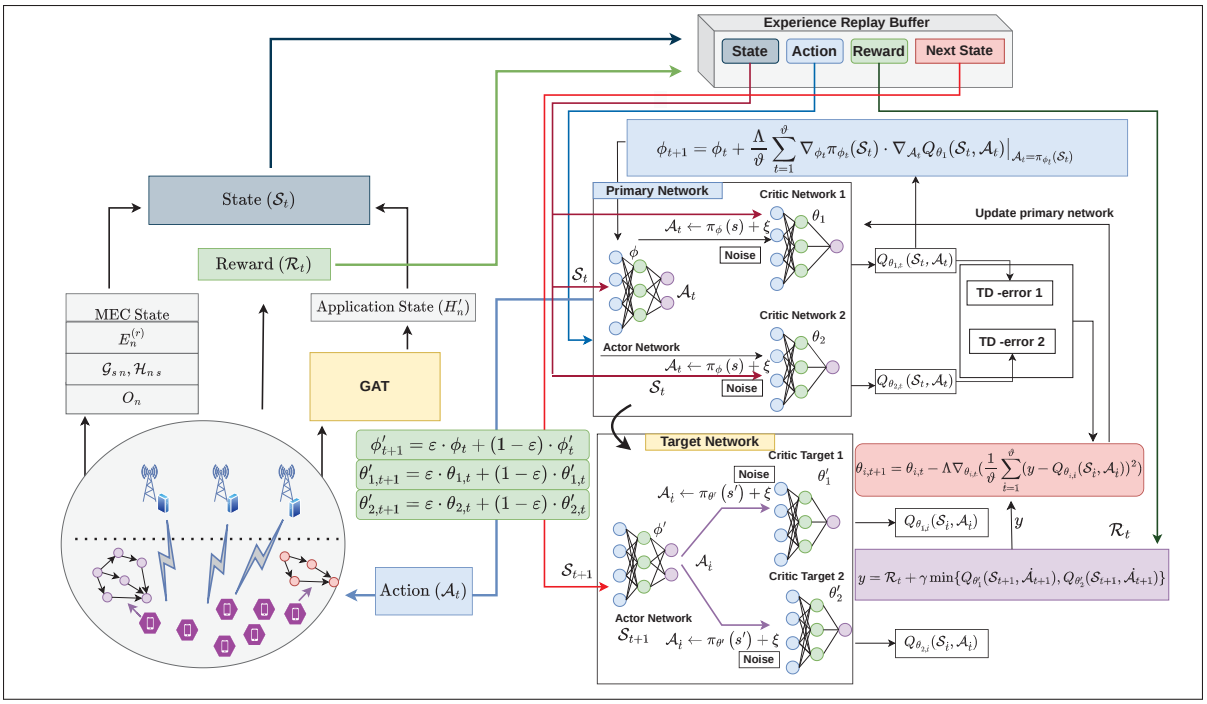

Figure 2.4 EMDTORA information and control flow

where $|\mathcal{E}|$ denotes the total number of edges, (i, j) represent the edges between node i and j. $\sigma(\cdot)$ is the sigmoid function and $e_{i,j}$ is the attention. The total loss combines both the feature and structure loss: $\mathcal{L}_{GAT} = \mathcal{L}_{\text{feature}} + \mathcal{L}_{\text{structure}}$. After computing the higher-level embedding H'_n for each application $n \in \mathbb{N}$ the MDP system state defined in Eq. (2.17) becomes:

$$\mathcal{S}_t = \{E_n^{(r)}, O_n, H'_n, \mathcal{G}_{s\,n}, \mathcal{H}_{n\,s} \,|\, \forall n \in \mathbb{N}, \forall s \in \mathbb{S}\}. \tag{2.25}$$

2.4.3 TD3-based task offloading and resource allocation

To solve the online MDP representation of the joint dependent task offloading and resource allocation problem we leverage the capabilities of DRL. In this field, DQN-based algorithms specifically emerge as a prominent solution, especially when dealing with high-dimensional state spaces like the one in this work. DQNs distinguish themselves with two novel features: a fixed

Q-target and a replay buffer. These elements aim to decrease sample correlation and stabilize the training process (Mnih *et al.*, 2015). However, DQNs are constrained by their inability to handle continuous action spaces and operate solely within discrete domains. Addressing this limitation, the Deep Deterministic Policy Gradient (DDPG) algorithm, which belongs to the actor-critic class of DRL algorithms, has been introduced (Lillicrap *et al.*, 2015b). DDPG seamlessly merges the characteristics of DQNs into the actor-critic structure. It efficiently learns both the *Q-value* and a deterministic policy by using a target network and replay experience. This integration accelerates convergence, making DDPG suitable for environments with continuous action spaces. Further advances in DRL have led to the development of Twin Delayed DDPG (TD3) (Fujimoto *et al.*, 2018b), an enhancement over DDPG. TD3 addresses the prevalent issues of overestimation bias and high variance found in DDPG. It introduces key improvements such as the clipped double Q-learning method, a delayed policy update mechanism, and a smoothing technique for target policy updates. These innovations collectively optimize TD3 for better performance and speed in training processes involving continuous actions.

Since we are dealing with a hybrid action space where the offloading decision part d is discrete and the resource allocation part f is continuous, we propose a TD3-based algorithm to solve the given problem. This algorithm consists of three main components: *(i) a primary network*, *(ii) a target network*, and *(iii) an experience replay buffer*. The primary network is responsible for making the offloading and resource allocation decisions by mapping the current states to actions according to the policy. It is an evaluation network that consists of three DNNs; one *actor network*, π, that generates the actions, and two DNNs called *critic networks*, Q_1, Q_2, that are used to evaluate the Q-values of state-action pairs. The target network follows a similar three-DNN structure, with one actor, π', and two target critic networks, Q'_1, Q'_2, used to define targets for training the primary networks. The weights of the primary actor network are denoted by ϕ, while the weights of the two primary critic networks are θ_1 and θ_2. Similarly, the weights of the target actor network are denoted by ϕ' and the weights of the two target critic networks are θ'_1 and θ'_2, respectively. The last component, the experience replay buffer, denoted by $\mathbb{B}$, is used to store the current state $\mathcal{S}_t$, action $\mathcal{A}_t$, and reward $\mathcal{R}_t$, together with the next state $\mathcal{S}_{t+1}$. Then,

during the training, a random mini-batch ϑ of experience tuples are fetched for the training to avoid the potential correlation among the tuples of the batch. The objective of the joint offloading and resource allocation action is to maximize the infinite horizon expected cumulative reward:

$$\dot{\mathcal{R}}_t = \max_{\pi_\phi} \mathbb{E}[\textstyle\sum_{k=0}^{\infty} \gamma^k \mathcal{R}_{t+k+1} \mid \mathcal{S}_t, \mathcal{A}_t], \tag{2.26}$$

where π_ϕ is the employed policy, which in our case is the output of the primary actor network. $\mathcal{R}_{t+k+1}$ is the future reward received at time $t+k+1$ given the current state $\mathcal{S}_t$ and action $\mathcal{A}_t$ and is calculated using Eq. (2.19). The variable k is used to index the future time steps in relation to the current time step t. A discount factor $\gamma \in (0, 1)$ weighs the importance of future rewards relative to the immediate ones. Next, we describe the networks' training procedure.

2.4.3.1 Primary Network Training

The primary actor network π is parameterized by its weights ϕ and is responsible for the offloading and resource allocation decisions. Let t denote an episode of the training process. The explored policy is used to map the current state $\mathcal{S}_t$ to an action $\mathcal{A}_t$. The action after adding exploration noise is:

$$\mathcal{A}_t = \text{clip}\left(\pi_\phi(\mathcal{S}_t) + \xi, \mathcal{A}_{\min}, \mathcal{A}_{\max}\right), \quad \xi \sim \mathcal{N}(0, \mu), \tag{2.27}$$

where $\mathcal{A}_{\min}$ and $\mathcal{A}_{\max}$ represent the lower and upper bounds of the valid action space, respectively. ξ is Gaussian noise with mean 0 and variance μ used for encouraging the exploration of the environment. In the primary actor network, the policy gradient theorem is used iteratively in training episodes to update the weights of the networks in the direction that maximizes the discounted cumulative reward (i.e., Eq. (2.26)) and is given by:

$$\phi_{t+1} = \phi_t + \tfrac{\Lambda}{\vartheta} \textstyle\sum_{t=1}^{\vartheta} \nabla_{\phi_t} \pi_{\phi_t}(\mathcal{S}_t) \cdot \nabla_{\mathcal{A}_t} Q_{\theta_1}(\mathcal{S}_t, \mathcal{A}_t)\big|_{\mathcal{A}_t = \pi_{\phi_t}(\mathcal{S}_t)}, \tag{2.28}$$

where π_ϕ is the task placement and resource allocation policy of the actor primary network, Λ is the learning rate that determines the rate at which the model learns by adjusting the scale of the updates to the weights of the network during training, and ϑ is the size of the mini-batch sample. $\nabla_\phi \pi_\phi$ and $\nabla_{\mathcal{A}} Q_{\theta_1}$ are the gradient of the policy with respect to weights ϕ and the gradient of the Q-value with respect to action $\mathcal{A}$ respectively. The Q-value, also referred as the action value, is typically calculated from the Bellman optimality equation:

$$Q(\mathcal{S}_t, \mathcal{A}_t) = \mathbb{E}[\ \mathcal{R}_t\big|_{\mathcal{S}_t, \mathcal{A}_t} + \gamma \max Q(\mathcal{S}_{t+1}, \mathcal{A}_{t+1})], \tag{2.29}$$

where $\mathcal{S}_{t+1}$, $\mathcal{A}_{t+1}$ are the next state and next action respectively. The primary critic networks use Eq. 2.29 to evaluate the current action. The weights of the critic networks are updated as follows:

$$\theta_{1,t+1} = \theta_{1,t} - \Lambda \nabla_{\theta_{1,t}} (\tfrac{1}{\vartheta} \textstyle\sum_{i=1}^{\vartheta} (y - Q_{\theta_{1,i}}(\mathcal{S}_i, \mathcal{A}_i))^2), \tag{2.30}$$

$$\theta_{2,t+1} = \theta_{2,t} - \Lambda \nabla_{\theta_{2,t}} (\tfrac{1}{\vartheta} \textstyle\sum_{i=1}^{\vartheta} (y - Q_{\theta_{2,i}}(\mathcal{S}_i, \mathcal{A}_i))^2), \tag{2.31}$$

where $Q_{\theta_{1,i}}(\mathcal{S}_i, \mathcal{A}_i)$ and $Q_{\theta_{2,i}}(\mathcal{S}_i, \mathcal{A}_i)$ are the current Q-values and y is the target Q-value calculated by the target network, as explained in the following.

2.4.3.2 Target Network Training

Generally in Q-learning-based algorithms, the function approximation error leads to optimistic values, a phenomenon called *overestimation bias*. Using TD3, we estimate two Q-values coming from the two critic networks and use the minimum between them to reduce this bias. The target value is thus calculated as:

$$y = \mathcal{R}_t + \gamma \min\{Q_{\theta'_1}(\mathcal{S}_{t+1}, \dot{\mathcal{A}}_{t+1}), Q_{\theta'_2}(\mathcal{S}_{t+1}, \dot{\mathcal{A}}_{t+1})\}, \tag{2.32}$$

where $\dot{\mathcal{A}}_{t+1}$ is the suggested best action from the corresponding target network. To achieve a stable training procedure, the weights of the target actor and critic networks are updated by

Algorithm 2.2 EMDTORA Offline Networks' Training Phase

Input: Node set V_n, feature matrix $H_n^{'}$ $\forall n \in \mathbb{N}$, buffer capacity ϑ, replay buffer $\mathbb{B}$, learning rate Λ, update frequency φ

Initialize: Actor network π and target actor network π' with weights ϕ and ϕ'; Critic networks Q_1, Q_2 and target critics Q_1', Q_2' with weights $\theta_1, \theta_2, \theta_1', \theta_2'$

while *not converged* **do**
  Sample initial state $\mathcal{S}_0$ randomly
  while $t < \max_{n \in \mathbb{N}}(|X_n|)$ ***AND*** *Eq.* (2.16b) ***AND*** *Eq.* (2.16c) **do**
    if $|\mathbb{B}| < \vartheta$ **then**
      `// Experience replay`
      Select action $\mathcal{A}_t$ randomly
    end if
    else
      Select action $\mathcal{A}_t$ using policy π_ϕ (Eq. (2.27))
    end if
    Execute action $\mathcal{A}_t$, calculate reward $\mathcal{R}_{t+1}$ (Eq. (2.19)), and determine next state $\mathcal{S}_{t+1}$
    Store transition $(\mathcal{S}_t, \mathcal{A}_t, \mathcal{R}_t, \mathcal{S}_{t+1})$ in $\mathbb{B}$
    if $|\mathbb{B}| \geq \vartheta$ **then** `// Training step`
      Sample mini-batch of ϑ experiences from $\mathbb{B}$
      Compute target value y (Eq. (2.32))
      Update critic weights θ_1 and θ_2 (Eqs. (2.30), (2.31))
      if t ***mod*** $\varphi == 0$ **then**
        Update actor weights ϕ (Eq. (2.28))
        Update target weights ϕ', θ_1', and θ_2' (Eqs. (2.33), (2.34), (2.35))
      end if
    end if
    $t \leftarrow t + 1$
  end while
end while

Output: Trained primary actor network π with weights ϕ

slowly mixing the weights of the primary and target networks using Polyak averaging (Haarnoja,

Zhou, Abbeel & Levine, 2018):

$$\phi'_{t+1} = \varepsilon \cdot \phi_t + (1 - \varepsilon) \cdot \phi'_t, \tag{2.33}$$

$$\theta'_{1,t+1} = \varepsilon \cdot \theta_{1,t} + (1 - \varepsilon) \cdot \theta'_{1,t}, \tag{2.34}$$

$$\theta'_{2,t+1} = \varepsilon \cdot \theta_{2,t} + (1 - \varepsilon) \cdot \theta'_{2,t} \tag{2.35}$$

where $\varepsilon \in (0, 1)$ is the Polyak averaging coefficient. The update is typically performed every φ steps to avoid an overestimation bias in the Q-values. In this case φ is called the *delay factor*. To ensure that the final actions $\mathcal{A}$ are in the valid range, we use the squashing function such that $\mathcal{A} = \mathcal{A}_{\text{min}} + (\mathcal{A}_{\text{max}} - \mathcal{A}_{\text{min}}) \times \sigma(\mathcal{A}_{\text{raw}})$, where $\sigma(\mathcal{A}_{\text{raw}}) = \frac{1}{1+e^{-\mathcal{A}_{\text{raw}}}}$ is the sigmoid function, which maps the output to the range $[0, 1]$. The training process of the primary and target networks is formally outlined in Algorithm 2.2. The training goes on until it reaches a convergence point, which is marked by the lack of substantial improvement in the average rewards yielded.

2.4.4 Online Phase

After having the actor network's weights trained, the online phase of the proposed framework initiates with acquiring information about the devices, the environment state and the long-term in-depth task dependencies from the GAT task embedding. Based on this set and the state of the network, it calculates the current state $\mathcal{S}_t$ using Eq. (2.25). With the current state available, EMDTORA then selects an action $\mathcal{A}_t$ using the trained actor network π_ϕ. This action reflects the offloading and resource allocation decisions for the t^{th} task of all devices. This online phase of EMDTORA is outlined in Algorithm

Algorithm 2.3 EMDTORA Online Task Placement and Resource Allocation Phase

Input: Node set V_n, feature matrix $H_n^{'}$ $\forall n \in \mathbb{N}$, trained actor network π, and its weights ϕ
while $t < \max_{n \in \mathbb{N}}(|X_n|)$ **do**
  Calculate state $\mathcal{S}_t$
  Select action $\mathcal{A}_t$ using policy π_ϕ (Eq. (2.27) with $\xi = 0$)
  Execute action $\mathcal{A}_t$
  $t \leftarrow t + 1$
end while
Output: Joint placement and resource allocation decision ρ for all tasks

2.4.5 Convergence Analysis

To prove the convergence of EMDTORA, we define the Bellman operator $\mathcal{B}_{\pi_\phi}$ for the Q-function $Q(\mathcal{S}_t, \mathcal{A}_t)$ as follows:

$$\mathcal{B}_{\pi_\phi} Q(\mathcal{S}_t, \mathcal{A}_t) = \mathcal{R}_t + \gamma \min \Big\{ Q_{\theta_1'}(\mathcal{S}_{t+1}, \dot{\mathcal{A}}_{t+1}), \\ Q_{\theta_2'}(\mathcal{S}_{t+1}, \dot{\mathcal{A}}_{t+1}) \Big\}, \tag{2.36}$$

where $\gamma \in [0, 1)$ is the discount factor. Let Q_1 and Q_2 be two Q-value functions. Applying the Bellman operator leads to the contraction of the difference between the two Q-values, which ensures that the Bellman operator $\mathcal{B}_{\pi_\phi}$ is a contraction property and is derived as follows:

$$\begin{aligned} &\left| \mathcal{B}_{\pi_\phi} Q_1(\mathcal{S}_t, \mathcal{A}_t) - \mathcal{B}_{\pi_\phi} Q_2(\mathcal{S}_t, \mathcal{A}_t) \right| \\ &= \gamma \Big| \min \Big(Q_{\theta_1',1}(\mathcal{S}_{t+1}, \dot{\mathcal{A}}_{t+1}), \;\; Q_{\theta_2',1}(\mathcal{S}_{t+1}, \dot{\mathcal{A}}_{t+1}) \Big) \\ &\quad - \min \Big(Q_{\theta_1',2}(\mathcal{S}_{t+1}, \dot{\mathcal{A}}_{t+1})), Q_{\theta_2',2}(\mathcal{S}_{t+1}, \dot{\mathcal{A}}_{t+1})) \Big) \Big|, \end{aligned} \tag{2.37}$$

$$\left| \mathcal{B}_{\pi_\phi} Q_1(\mathcal{S}_t, \mathcal{A}_t) - \mathcal{B}_{\pi_\phi} Q_2(\mathcal{S}_t, \mathcal{A}_t) \right| \leq \\ \gamma \left| Q_1(\mathcal{S}_{t+1}, \dot{\mathcal{A}}_{t+1}) - Q_2(\mathcal{S}_{t+1}, \dot{\mathcal{A}}_{t+1}) \right|, \tag{2.38}$$

$$||\mathcal{B}_{\pi_\phi} Q_1 - \mathcal{B}_{\pi_\phi} Q_2||_\infty \leq \gamma ||Q_1 - Q_2||_\infty. \tag{2.39}$$

Since the Bellman operator is a contraction, the sequence of Q-values generated by the algorithm, $\{Q^{(t)}\}$, converges as to a fixed point Q^* as $t \rightarrow \infty$. The difference between successive Q-value iterates decreases as

$$||\mathcal{B}_{\pi_\phi}^{(t+1)}Q - \mathcal{B}_{\pi_\phi}^{(t)}Q||_\infty \leq \gamma||\mathcal{B}_{\pi_\phi}^{(t)}Q - \mathcal{B}_{\pi_\phi}^{(t-1)}Q||_\infty \leq \\ \cdots \leq \gamma^t||\mathcal{B}_{\pi_\phi}Q_1 - Q_2||_\infty. \tag{2.40}$$

The difference between successive iterates of the Q-values diminishes geometrically, ensuring the convergence of EMDTORA.

2.4.6 Computational Complexity Analysis

The total computational complexity of the EMDTORA depends on the complexity of the DNN training process and the experience replay buffer. The computational complexity of each critic network is $O(\Upsilon W_c \alpha_c + \sum_{l=1}^{\alpha_c - 1} W_{c,l} W_{c,l+1})$, where Υ is the input layer, W_c denotes the number of neurons, α is the number of training layers while l shows the indexing of layers. Similarly, the complexity of the actor network is equal to $O(\Upsilon W_a \alpha_a + \sum_{l=1}^{\alpha_a - 1} W_{a,l} W_{a,l+1})$. We consider that the actor network is updated every E steps and thus, the total training complexity over total steps T_{tot} is $O(T_{tot}(2\Upsilon W_c \alpha_c + 2\sum_{l=1}^{\alpha_c - 1} W_{c,l} W_{c,l+1}) + \frac{T_{tot}}{E}(\Upsilon W_a \alpha_a + \sum_{l=1}^{\alpha_a - 1} W_{a,l} W_{a,l+1}))$. The complexity of the experience replay buffer is given as $O(2|S| \times |A| + \log_2 \vartheta)$ where $|S|$ is the state size, $|A|$ is action size and ϑ is the batch size. Thus, the total computational complexity of EMDTORA is given as:

$$O\Bigg(T_{tot}\Big(2\Upsilon W_c \alpha_c + 2\sum_{l=1}^{\alpha_c - 1} W_{c,l} W_{c,l+1}\Big) + \\ \frac{T_{tot}}{E}\Big(\Upsilon W_a \alpha_a + \sum_{l=1}^{\alpha_a - 1} W_{a,l} W_{a,l+1}\Big) \\ + 2|S| \times |A| + \log_2 \vartheta\Bigg). \tag{2.41}$$

The complexity of EMDTORA online phase is approximately $O(V \times X \times \mathbb{N})$ where V is the number of application per user, X is the number of tasks per application, and $\mathbb{N}$ is total number of users. The complexity is linear with respect to the number of applications, total tasks in the application and total number of users.

2.5 Results and Discussion

In this section, we evaluate the efficiency and efficacy of the proposed mechanism. First, the experimental setup and parameter configuration is detailed, followed by the presentation of the baseline solutions used for benchmarking. Then, extensive simulations and their results are presented in order to verify EMDTORA's ability of minimizing the IoT device's energy consumption and delay under dynamic conditions, while respecting the system constraints.

2.5.1 Simulation setup

We simulate a MEC environment with $|\mathbb{S}| = 3$ edge servers and $|\mathbb{N}| \in [2, \ldots, 10]$ IoT devices. The devices are wirelessly connected to the servers through the respective base stations. Regarding the tasks, we generated a diverse range of DAGs to ensure the broad applicability of our approach. The number of tasks $|X_n|$ in each DAG is varied in the range of 5 to 30 tasks per application. Furthermore, the in- and out-degrees of the vertices are randomly varied between 1 and 4. The data sizes for each task in an application w_{in} are randomly assigned within a range of 5 to 25 KB. The computational workload c_{in} per task for all DAGs is assumed to be between 10^6 and 10^8 Hz. For the network setup, the transmission power for the user devices $P_{ns}^{(u)}$ is fixed at $100mW$, and for the edge servers $P_{sn}^{(d)}$ at $1W$ (Lin *et al.*). The CPU clock frequency for the user devices F_n^0 is fixed to $1GHz$, while each edge server is equipped with a total resource capacity F^s of $2.4GHz$ (Chen *et al.*, 2021a). The channel gains between the user devices and the edge servers $\mathcal{G}_{sn}$, $\mathcal{H}_{ns}$ range from -5 to $-40dB$. To simulate the dynamic network conditions, at each simulation time step, the channel gain changes, and the next value is randomly selected from within this range, unless stated otherwise.

The TD3 model comprises six neural networks and is the core component of our simulations; in a rigorous offline phase, we fine-tuned the model to achieve optimal performance, leading to the following configuration: each neural network comprises three hidden layers, structured with 512, 256, and 256 neurons respectively. This distribution of neurons is used for both the actor and critic components of the model. The learning rate Λ is fixed for the actor-critic network at 0.00001 as with this value we observe a stable performance. The experience replay buffer size $|\mathbb{B}|$ is 50000 while a batch size ϑ of 256 is used; the soft update parameter φ is set to 0.005. The Adam optimizer is used to update the weights of the actor and critic networks. We train the agent for 4000 episodes. During testing, we conduct experiments with 40 randomly generated DAG applications and then the results are averaged. The simulations were carried out on an Intel Core i7 workstation that is equipped with 8 CPU cores, 16 threads, and $32GB$ of RAM. We used CUDA 12.1 on an NVIDIA GTX T600 with $6GB$ of memory for the training.

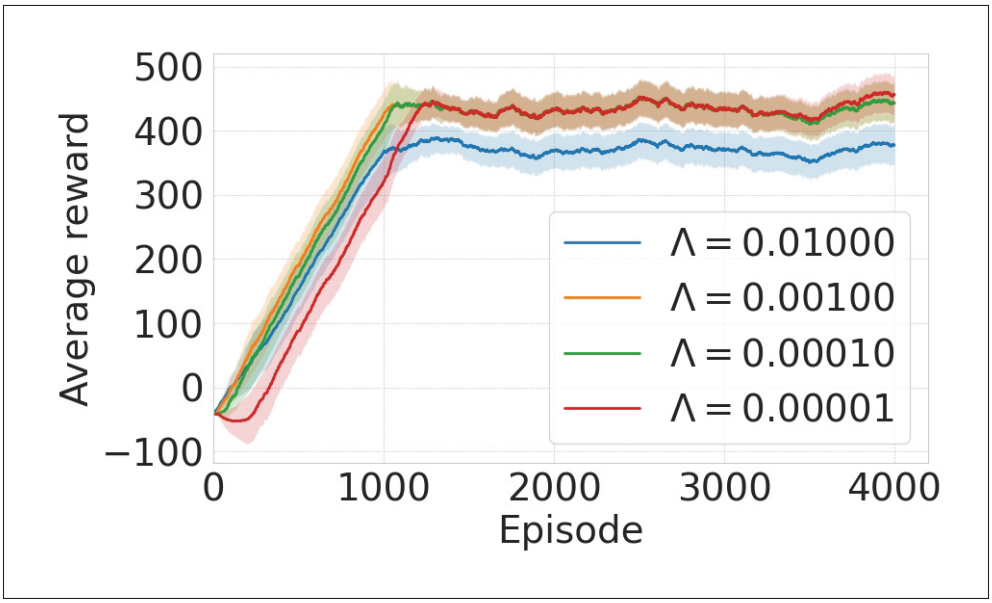

Figure 2.5 Λ impact on convergence

In Fig. 2.5, we demonstrate the effect of the learning rate value on the convergence pattern of the TD3 agent training by plotting the average reward $\mathcal{R}_t$ for 4000 episodes. The figure shows a moving average calculated with a window size of 100. The upper part of the shaded area shows the maximum value, while the lower part shows the minimum value of the reward

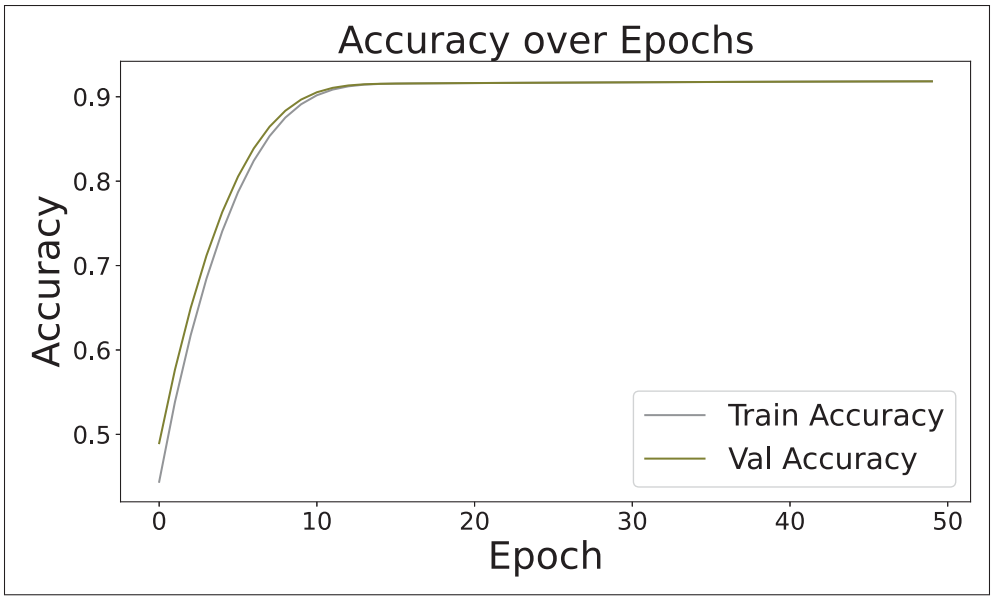

Figure 2.6 GAT model accuracy analysis

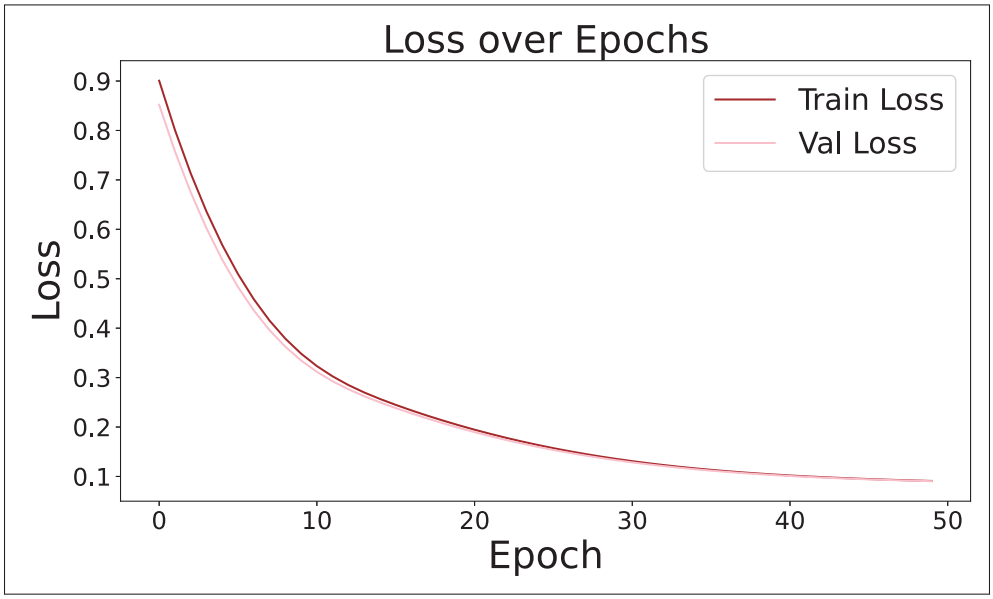

Figure 2.7 GAT model loss analysis

during the window. The observed fluctuations occur due to the exploration tendency of the TD3 enhanced by the additional noise introduced during the training phase. We see that a lower learning rate, $\Lambda = 0.00001$, triggers a gradual and stable improvement in rewards, indicating

a steady but potentially slower learning process. On the contrary, with higher learning rates as $\Lambda = 0.01$, the reward values demonstrate rapid fluctuations, suggesting faster but less stable learning trajectories. Moderate learning rates, such as $\Lambda = 0.001$ and $\Lambda = 0.0001$, offer a balance between stability and convergence speed, providing the optimal setting for efficient learning. Fig. 2.6 and 2.7 show the training and validation accuracy and loss of the GAT model. It is observed that both the training and validation accuracy reached a plateau close to 0.9, while the continual decrease in the loss suggests that convergence occurred. In other words, smooth, continuous calculation of training and validation loss curves with a rise in accuracy can only prove that the gradient descent algorithm does minimize the loss function. Closely aligned training and validation curves suggest good generalization without great overfitting. This convergence pattern suggests that GAT model has efficiently learned task features, as well as task dependencies.

2.5.2 Comparison with other schemes

We benchmark EMDTORA with the following algorithms:

- *All Local Execution (ALE)*: In this scheme, all tasks of an application are executed on the local device, utilizing its resources to the fullest extent without being offloaded to an edge server. ALE is used in various studies in the literature, such as in (Wang *et al.*, 2021a), (Nieto *et al.*, 2023), and (Xiao *et al.*, 2022), for comparison purposes. This scheme is suitable when the network conditions are not favorable and/or energy efficiency is not the primary objective.
- *All Edge Execution (AEE)*: In the AEE scheme, all the tasks from the local devices are offloaded to the edge servers for execution. AEE is suitable for a scenario where energy is the main concern, and the network conditions are favorable. The servers' resources in AEE are distributed equally among the tasks. In our implementation, the AEE follows a greedy approach based on resource availability for the selection of the edge server. Both the works in (Wang *et al.*, 2021a) and (Chen *et al.*, 2021a) have employed this scheme in their evaluations.
- *All Random Execution (ARE)*: In this kind of offloading scheme, the offloading decision for a task is randomly selected between locally and at the edge server. In this scheme, the

resources are also allocated randomly to the tasks. ARE is used in (Xiao *et al.*, 2022) and (Chen *et al.*, 2021a).

- *DVRMO-MM*: The DVRMO-MM scheme is introduced in (Liu *et al.*, 2023b), and uses the DRL-based DDPG algorithm to make offloading and resource allocation decisions. In the original work, DVRMO-MM accounts only for minimizing the delay of the system; however, to make the comparison fair, we modified it to include the energy consumption of the IoT devices as well. This modification aims to provide a more comprehensive optimization of both the reliability and energy efficiency of the system.

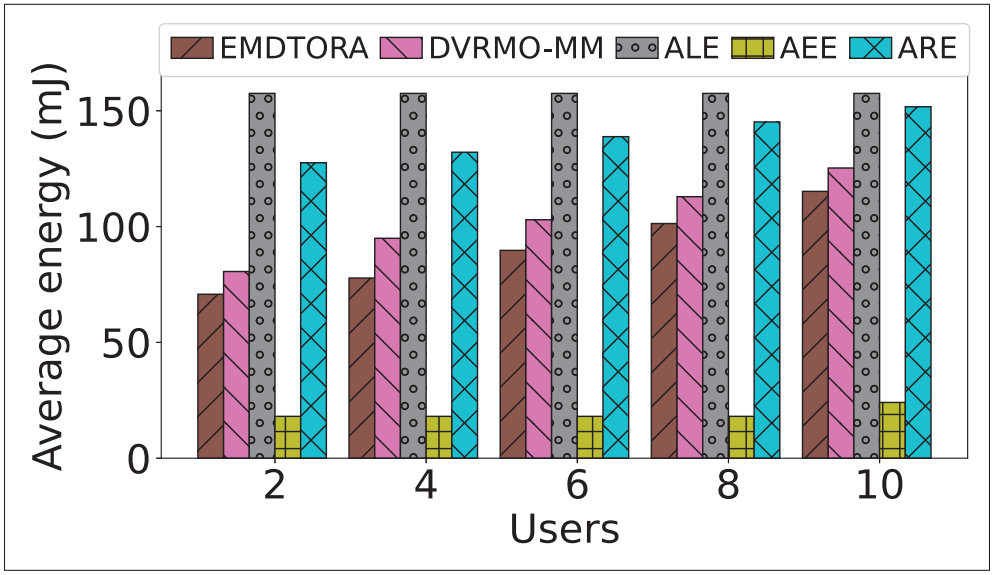

Figure 2.8 Energy consumption analysis

In the first experiment, we perform a comparative analysis against a growing number of users/IoT devices. Each device $n \in \mathbb{N}$ executes an application with random structure that consists of $|X_n| = 10$ tasks with varying features. In Fig. 2.8 the results of the average energy consumption per IoT device for all five mechanisms are depicted. Since we only account for the energy consumption of the IoT devices, AEE demonstrates the highest efficiency, as the only energy consumed is the one during data transmission. In contrast, the ALE strategy yields a higher energy consumption, as the on-device execution is power hungry. In the ARE scheme, the CPU resources allocated for the task as well as the placement decision between local and edge servers are random; thus, this inefficient resource allocation and scheduling results in the highest

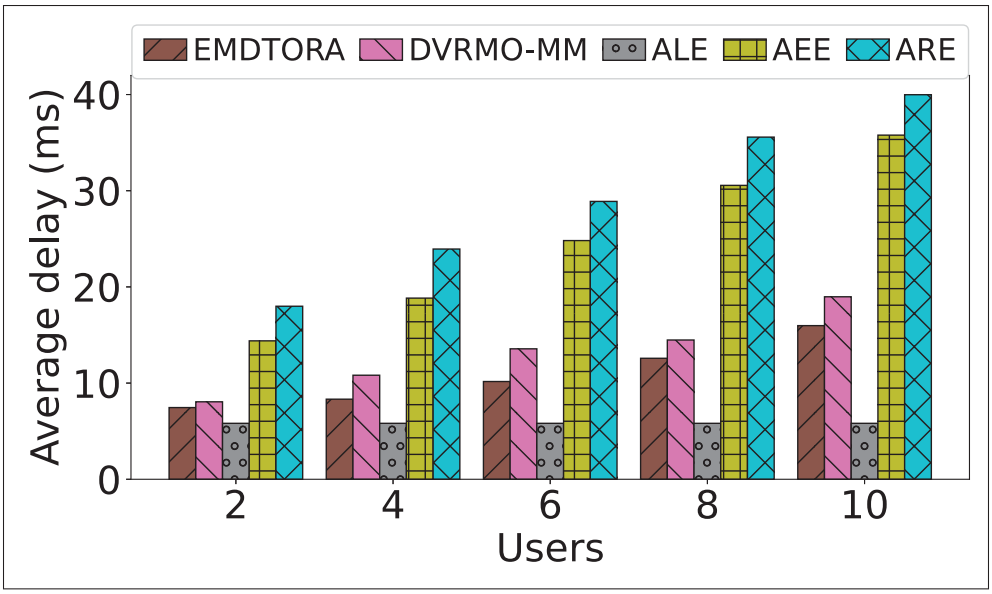

Figure 2.9 End-to-end delay analysis

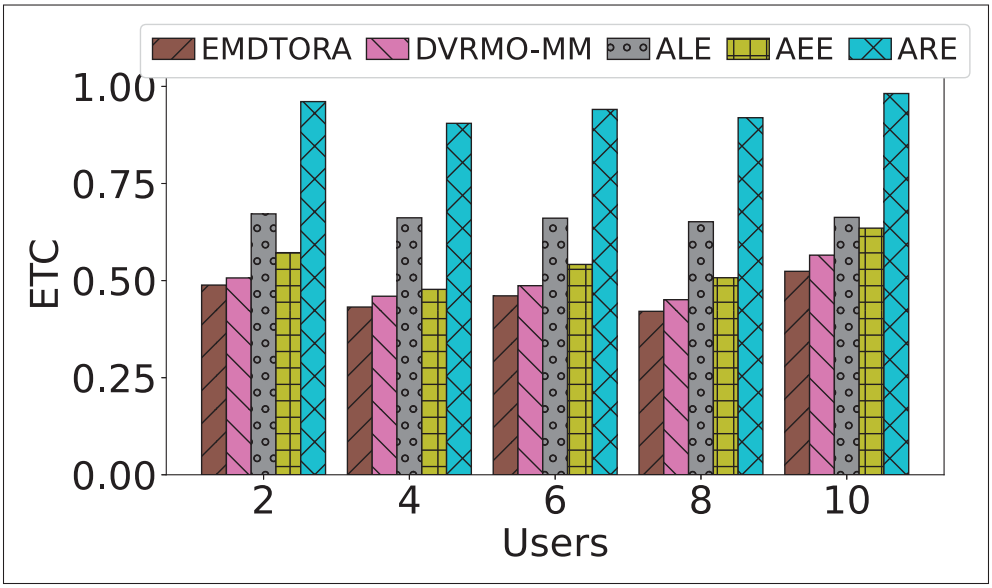

Figure 2.10 ETC analysis

energy consumption between the compared algorithms. Between the DRL-based alternatives, i.e., EMDTORA and DVRMO-MM, our proposed solution prevails due to the clipped double Q-learning employed by TD3. This helps reduce the overestimation bias, resulting in a more reliable learning. We also observe that as the number of users increases, an increase in energy

consumption occurs for both mechanisms. This happens due to the fact that the edge servers' capacity does not change. As a result, the number of tasks executed at the local devices gradually increases since the availability at the edge is not enough to accommodate all the users simultaneously. This ultimately leads in a gradually increase of the energy consumption.

The results presented in Fig. 2.9 show the average delay per application per device, i.e., after the completion of all their tasks, for all five compared mechanisms. Naturally, the ALE scheme performs the best since when tasks are executed on-device the delay is lower as there is no transmission involved. On the contrary, the AEE strategy showcases the highest average delay due to constant transmissions to/from the edge servers. Although the execution time is lower at the edge, the transmission delay seems to play a dominant role in this setting. Because of that, ARE, where tasks are randomly executed locally or at the edge, demonstrates a lower delay than AEE but higher than the proposed EMDTORA. The performance of DVRMO-MM is better than AEE and ARE but worse than that of EMDTORA, again because of the overestimation bias. Overall, the proposed solution consistently maintains a low average delay as it intelligently executes the tasks either at the local devices or edge servers, depending on the state of the environment. However, the true value of the proposed algorithm emerges when the energy consumption and the end-to-end delay minimization are studied jointly in terms of *Energy-Time-Cost (ETC)* (Yan, Bi, Zhang & Tao, 2019), (Guo, Liu, Yang, Xiao & Li, 2018), (Yuan, Xu, Sun & Zhang, 2022). The ETC is calculated with the following formula: $\frac{1}{2}(\frac{\mathbb{E}}{\mathbb{E}_M} + \frac{\mathbb{D}}{\mathbb{D}_M})$, where $\mathbb{E}$ is the mean energy consumed and $\mathbb{E}_M$ is the peak value of the energy consumption; $\mathbb{D}$ is average delay and $\mathbb{D}_M$ is the highest delay value. In Fig. 2.10 we see that when the gains between the two targets are combined, EMDTORA outperforms all the competitors by a considerable margin by demonstrating a superior balance between energy consumption and delay.

Following, we evaluate the performance of EMDTORA against the other schemes under varying wireless channel conditions. For this experimentation, we fix the number of users to 10 and we deterministically vary the channel gains from $-40db$ to $-5db$. In Fig. 2.11 we show the analysis of energy consumption. Initially, under the worst channel conditions, the average energy consumption for the ARE scheme is higher than all the other schemes, as the tasks are

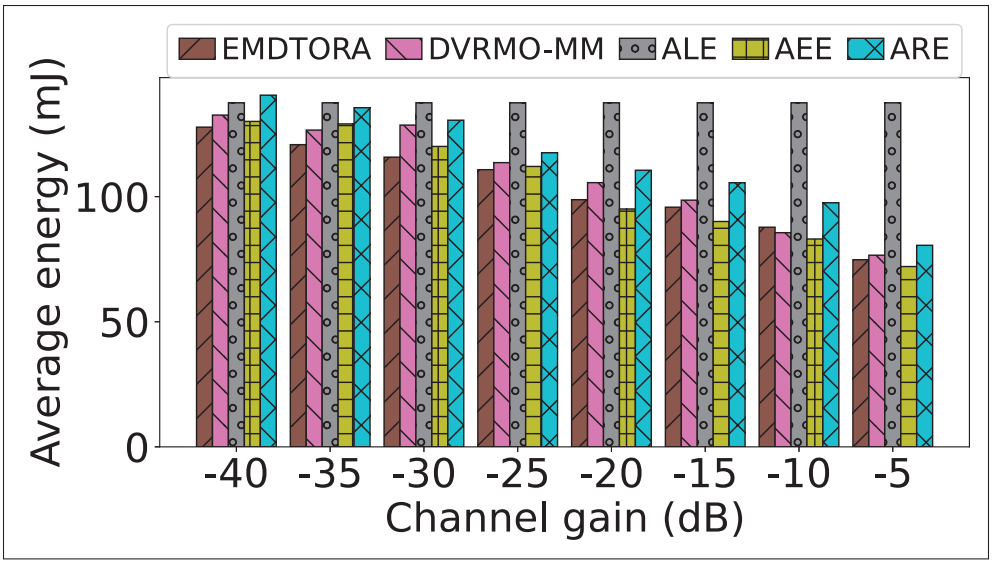

Figure 2.11 Energy consumption analysis

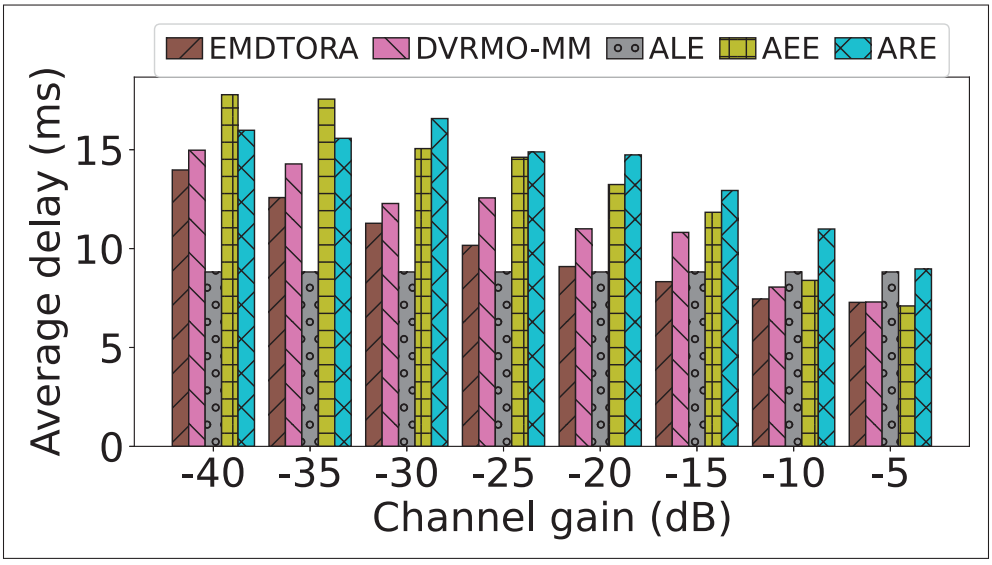

Figure 2.12 End-to-end delay analysis

executed randomly either locally or at the edge, while having random resource allocation. As expected, the varying channel conditions have no effect under the ALE strategy as there is no transmission energy involved. In the case of AEE, the energy consumption is high at the start; however, as the channel conditions improve, the energy consumption is reduced. Under the

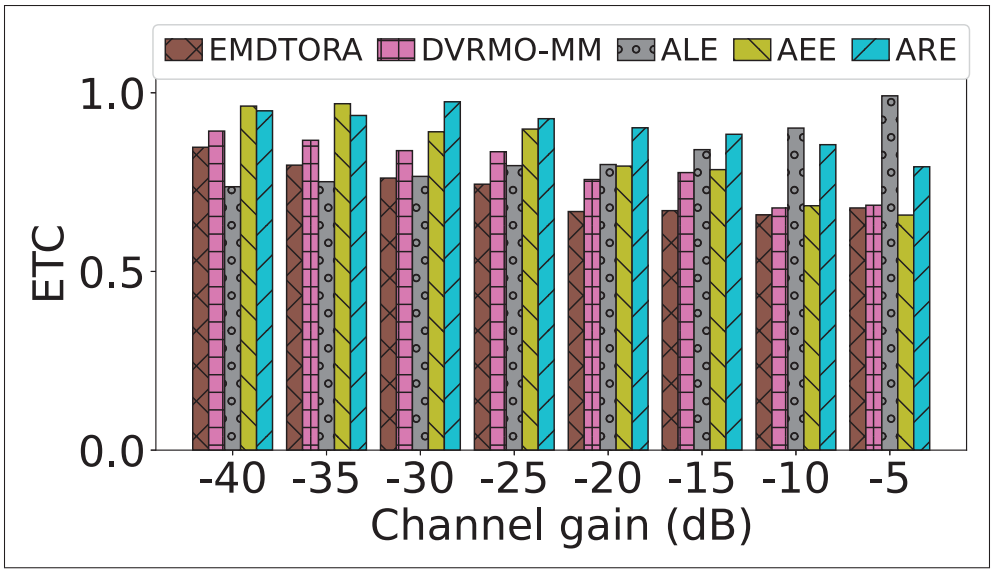

Figure 2.13 ETC analysis

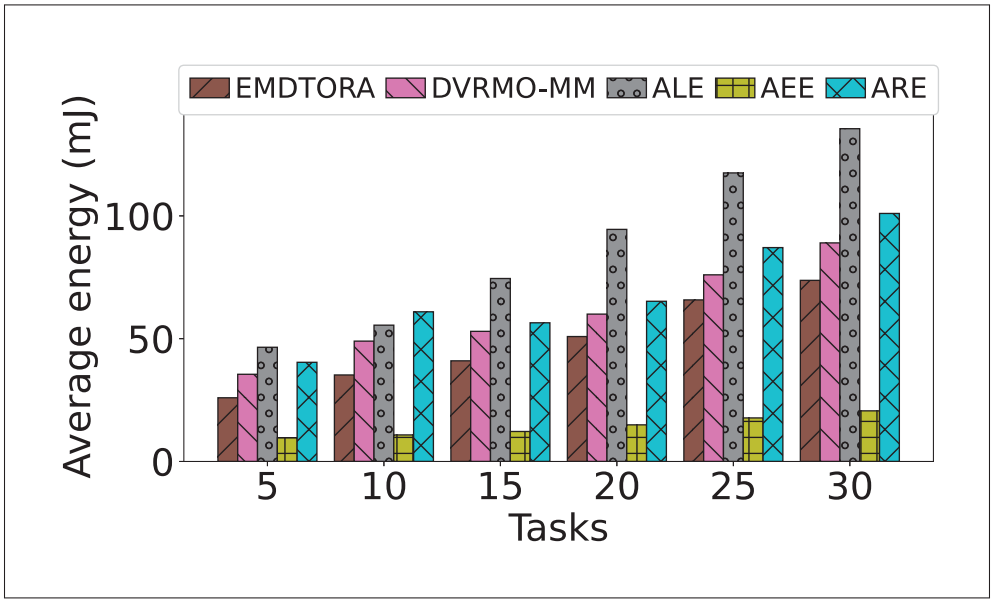

Figure 2.14 Energy consumption analysis against a growing number of tasks

worst channel conditions, EMDTORA is outperforming all the other schemes; it efficiently analyzes channel conditions and performs the task on local devices with efficient resource allocation to reduce energy consumption. DVRMO-MM also achieves low energy consumption; however, the placement and resource allocation of DVRMO-MM is not that efficient and thus has

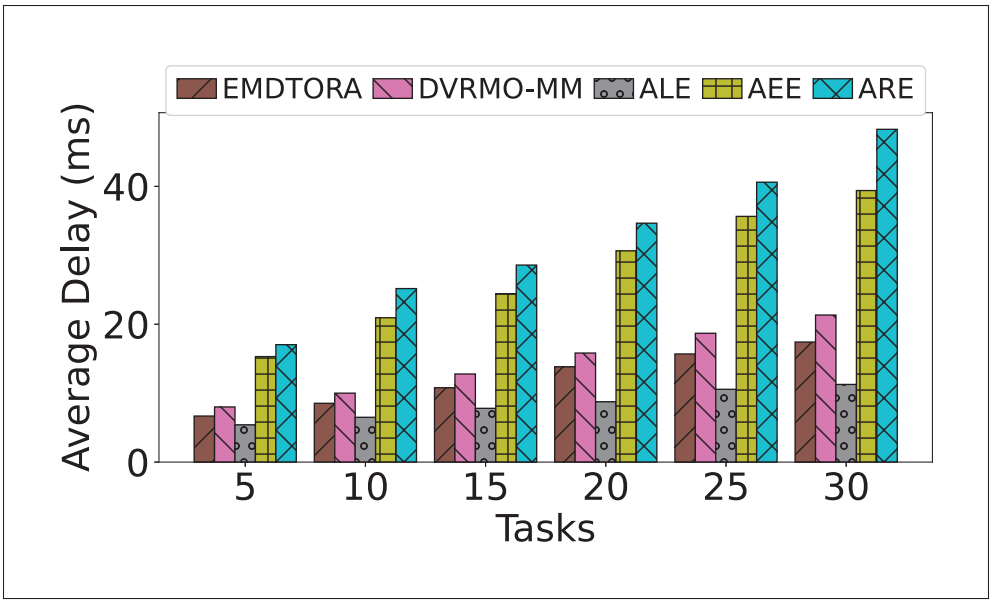

Figure 2.15 End-to-end delay analysis against a growing number of tasks

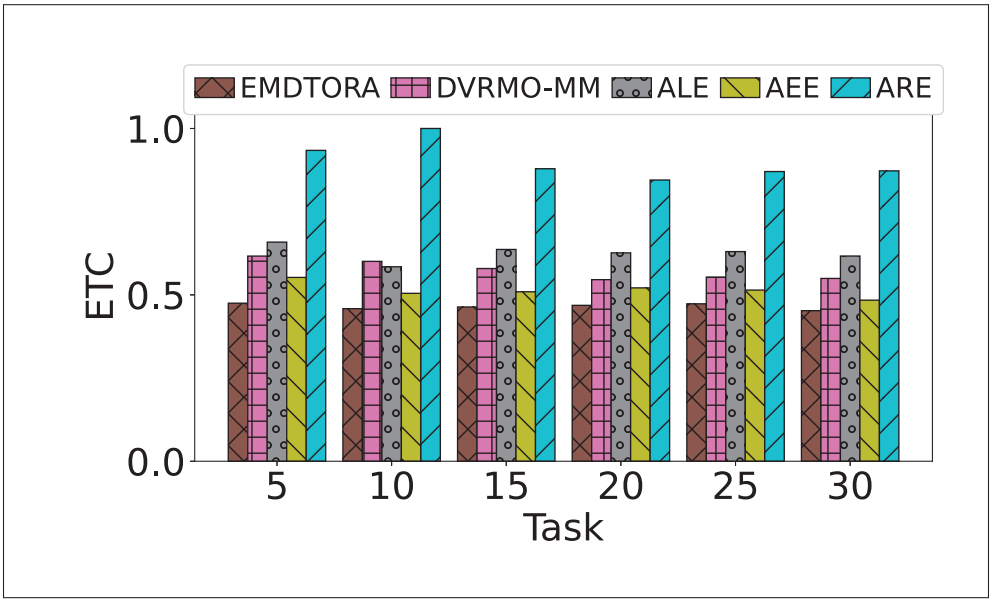

Figure 2.16 ETC analysis against a growing number of tasks

higher energy consumption than EMDTORA. Additionally, as the channel conditions improve, there is a decrease in the energy consumption of EMDTORA because the transmission energy consumption is reduced.

When it comes to the average end-to-end delay achieved under various channel gain values, as shown in 2.12, the results demonstrate that under poor channel conditions, EMDTORA naturally yields higher delays compared to the ALE approach, which outperforms every alternative. However, as the channel conditions improve, EMDTORA effectively reduces the average delay experienced. This reduction is achieved by intelligently increasing the offloaded tasks to edge servers due to more favorable conditions. In particular, close to a channel gain of $-5dB$, EMDTORA's average delay is lower than that of ALE and on par with AEE, primarily due to significantly reduced transmission delays. The DVRMO-MM based offloading scheme also maintains a lower delay than ALE; however, on average, the learned policy is still not effective enough to compete with EMDTORA. As expected, the AEE scheme yields the lowest delay under favorable network conditions, as all the tasks are offloaded to the edge without any consideration.

The ETC analysis corroborates our findings from this experimentation family in Fig. 2.13. It is evident from the results that nearly for each channel gain value the ETC of EMDTORA is lower than that of all others, striking a balance between energy consumption and delay minimization. However, specifically at the value of $-5db$ the ALE scheme performs better than EMDTORA since when channel conditions are that favorable, slight miscalculations of DRL-based agents can be proven costly against an all-out edge offloading strategy.

Next we depicts the results of a performance analysis under a growing number of tasks per application. As shown in Fig. 2.14 when the number of tasks increases, the average energy consumption also increases for all competitors. However, naturally, the increase in energy consumption for AEE is the lowest compared to the other schemes and the highest for ALE because for AEE only transmission energy is consumed. EMDTORA maintains a lower energy consumption compared to all other schemes, except AEE, due to intelligent offloading decisions and resource allocation under the joint goal of optimizing energy consumption together with delay. The delay effect is shown in Fig. 2.15. Here, we observe that as the number of tasks increases, the longer it takes to execute the whole application; thus, there is an increase in the average delay for all five mechanisms. However, the increase yielded by the ALE strategy is the

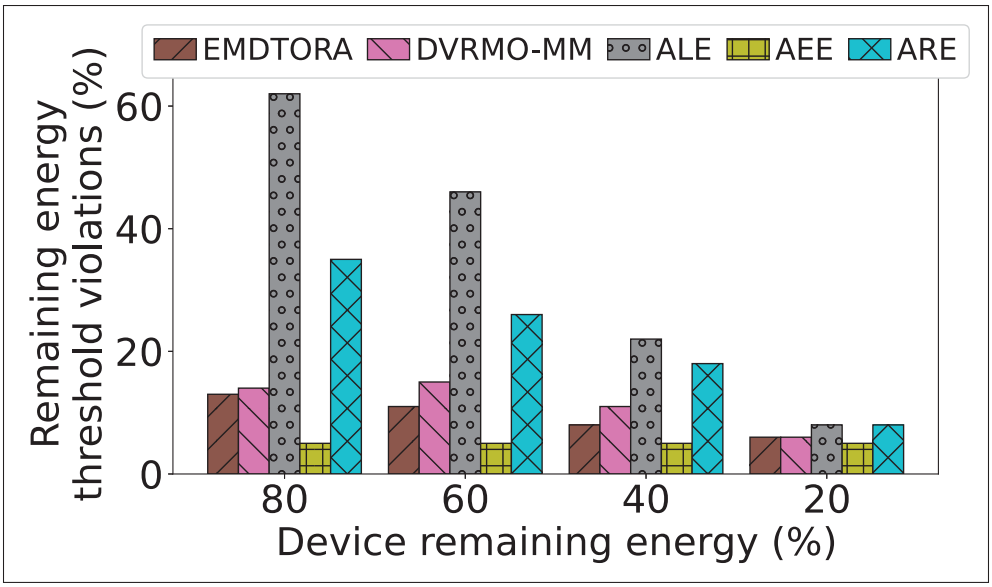

Figure 2.17 Energy consumption analysis against relaxing constraints

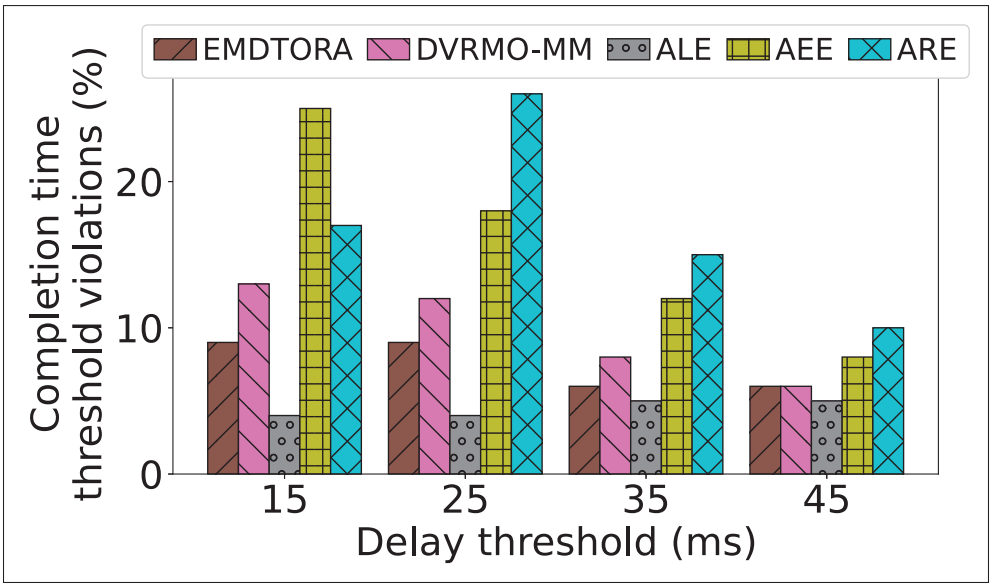

Figure 2.18 End-to-end delay analysis against relaxing constraints

lowest since only the local execution delay is taken into account, while the impact is the worst on the ARE achieved delay due to the inefficient placement and resource allocation decisions. As expected, in terms of ETC, the proposed EMDTORA solution shows a significant performance lead compared to all the other schemes. This is partly due to the GAT mechanism which helps

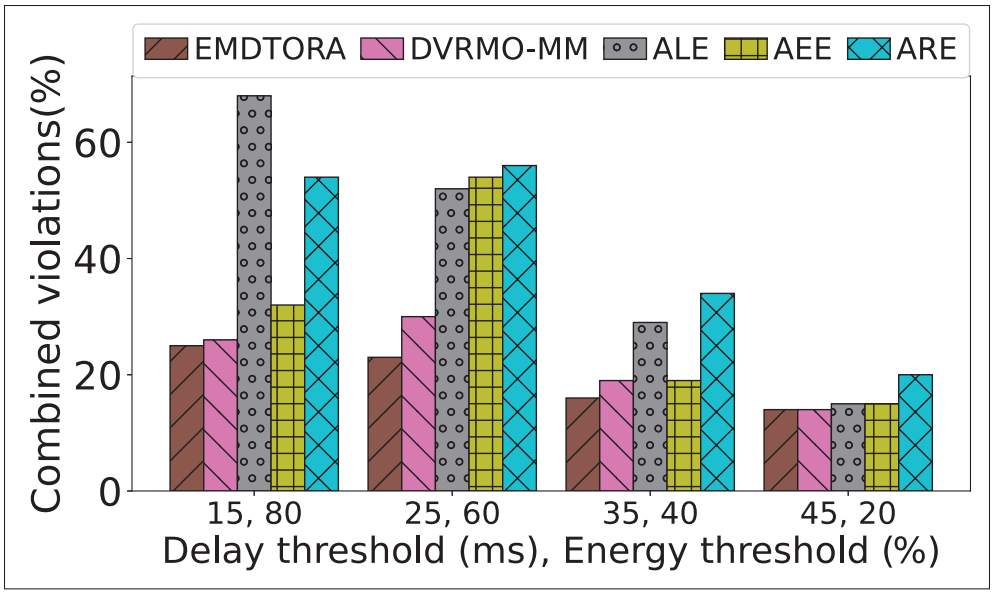

Figure 2.19 Combined violations analysis against relaxing constraints

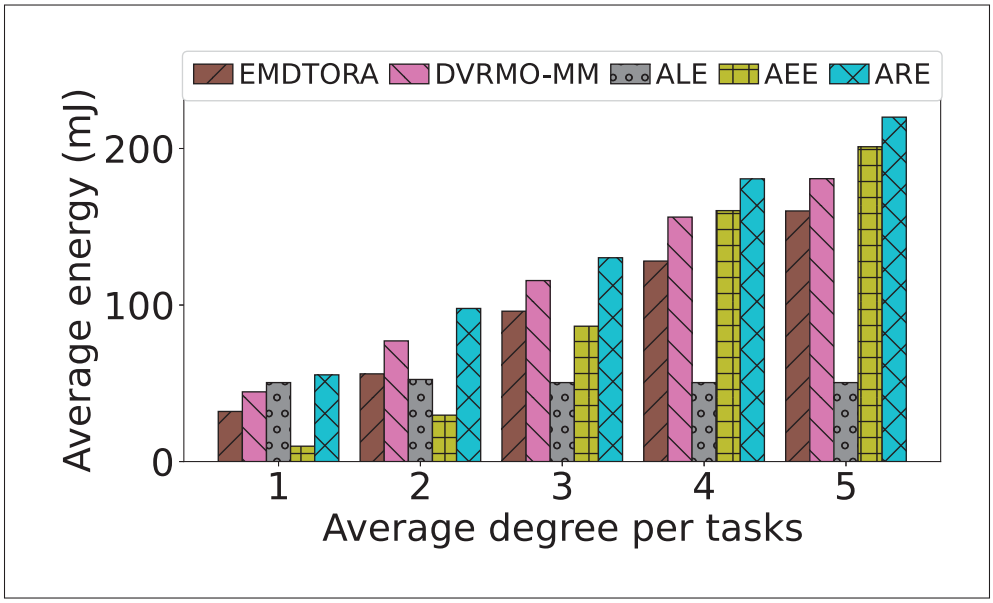

Figure 2.20 Energy consumption analysis against growing average degree per task

effectively learn the inter task dependencies and features of the applications and make informed task placement and resource allocation decisions. As shown in Fig. 2.16, EMDTORA, led by the tendency to improve the acquired long-term reward, keeps the ETC of the system low and

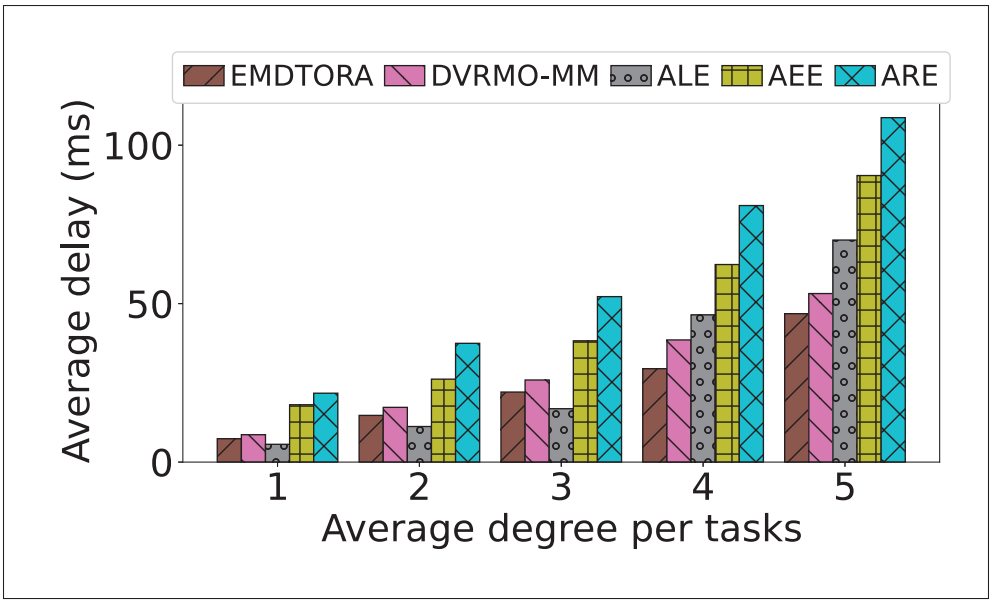

Figure 2.21 End-to-end delay analysis against growing average degree per task

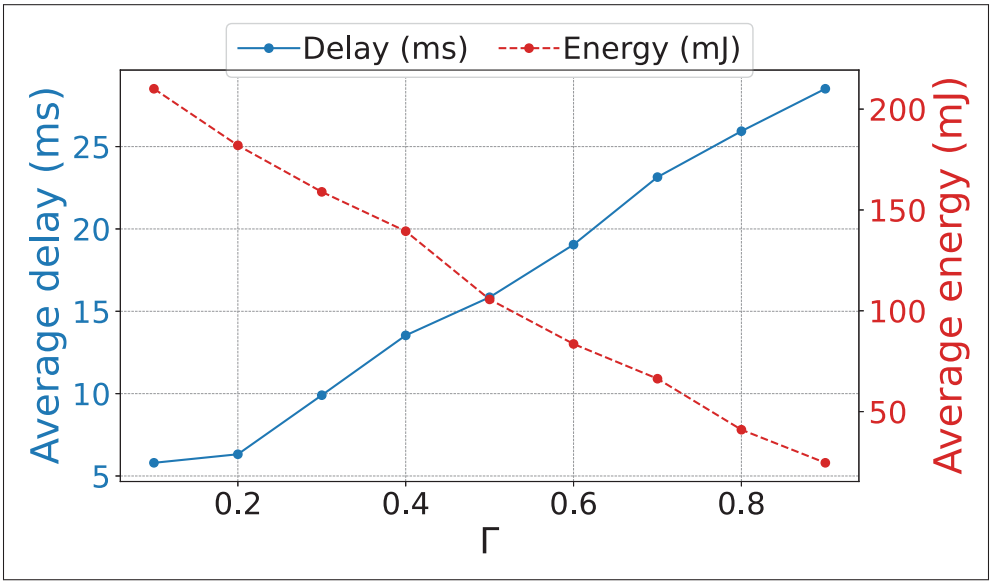

Figure 2.22 Effect of hyperparameter Γ on system delay and energy

manages to maintain the balance between energy consumption and delay minimization. A similar behavior is observed by DVRMO-MM; however, the negative impact of the overestimation bias

and the lack of the in-depth knowledge of the inherit structure and features of each application is evident.

In all of the above experiments, most of the compared schemes frequently violated the energy and delay constraints. To provide more insights on this behavior, we evaluated the degree of constraint violation for the different schemes. Fig. 2.17 shows the percentage of violations for a varying threshold of the remaining energy constraint. Specifically, the x-axis represents the threshold, which is the minimal residual energy that we want to have at each device after executing an application in full. Initially, this threshold is stringent, but it becomes more lenient as we progress. The results represent the average percentage of violations after executing 40 randomly structured applications. ALE has the highest number of violations while AEE has the lowest. With regards to AEE, the violations occur mainly due to transmission energy consumption. EMDTORA has the lowest number of energy consumption violations compared to all except for AEE. Moving on, in Fig. 2.18 the application completion time threshold violation is evaluated against different delay threshold values. When the completion threshold is extremely low, the ALE scheme showcases the lowest number of violations, while AEE the highest one. As the threshold value is relaxed, the completion time threshold violations for all schemes are reduced. As always, the goal of EMDTORA is to maintain the balance between energy consumption and delay satisfaction. Fig. 2.19 describes on the x-axis different configurations of delay thresholds (in ms) and remaining energy thresholds in (%); such as, "$15, 80$" describes the configuration for a $15ms$ delay threshold together with an 80% remaining energy threshold. As we move on the right of the x-axis the thresholds are relaxed with $45ms$ of delay and 20% of remaining energy threshold at the end. The y-axis represents the percentage of total violations. The performance of the proposed EMDTORA method is significantly better than that of any other scheme for all threshold combinations where the condition exhibits a lower rate of violation.

Our evaluations reveal that the two extremes, i.e., the ALE and AEE schemes, can outperform EMDTORA under specific scenarios. However, these cases are rare and happen due to the inherent non-optimality of the DRL-based solvers. Additionally, when considering the joint

optimization in terms of ETC, EMDTORA stood out as superior in all benchmarks. This showcases EMTDORA's capability in finding a good tradeoff between the two conflicting goals. Finally, compared to the non-GAT DDPG implementation in DVRMO-MM, EMDTORA capturing the inherit structure and feature of the application's tasks results in significant improvements. What is more, the usage of double Q-networks and delay policy updates to overcome the overestimation bias found in DVRMO-MM enhances its performance further.

The average degree of a DAG is calculated by summing the in-degree or out-degree for each node and then dividing it by the number of nodes in the graph. This is an essential metric since a higher average degree implies more dependencies among tasks. Fig. 2.20 showcases the impact of the degree per task in a DAG against the average energy consumption. Specifically, the energy consumption of all schemes increases with the task dependency degree. Nonetheless, EMDTORA demonstrates a relatively low energy consumption throughout all the testing degrees except for ALE. ALE shows a lower energy consumption, as all the tasks are executed on the local device and no transmission energy is involved. It is observable that AEE and ARE suffer from higher energy consumption compared to DVRMO-MM and EMDTORA, which can handle such complexity more efficiently. Fig. 2.21 reflects the relationship between the average degree of tasks in a DAG and the corresponding average delay. It is evident that when increasing the node degree, the delay in all schemes increases as well. Among all, the delays in AEE and ARE are the largest and surpass $100ms$ at an average degree per task of 5. The scheme that possesses less sharpness in delay increase is EMDTORA and DVRMO-MM, which have kept the delay to less than $50ms$. This shows that these schemes are more resistant to task complexity. ALE acts moderately by showing an intermediate performance. Finally, Fig. 2.22 illustrates the effect of Γ on the average energy consumption and completion time. The energy consumption reaches a minimum value of about $25mJ$ when the value of Γ reaches to 0.9. However, this is at the expense of the delay, which increases to about $27ms$. On the other hand, when the Γ values are lower, the minimization of the completion time is prioritized. For example, the average delay is reduced to about $5ms$, whereas the energy consumption increases to about $200mJ$. At $\Gamma = 0.5$, a balance is established between the energy consumption and delay.

2.6 Conclusion & Future Work

In conclusion, this paper presented EMDTORA, an Energy-Aware Multi-user Dependent Task Offloading and Resource Allocation mechanism for IoT devices operating in MEC infrastructures. EMDTORA integrates a Graph Attention Network-based mechanism to capture the complex application task interdependencies with an actor-critic, off-policy DRL algorithm for improved decision-making. Extensive simulations show that EMDTORA outperforms existing approaches, effectively balancing the trade-off between energy consumption and delay minimization, while adhering to system constraints. The results underscore the potential of EMDTORA to improve the energy efficiency and performance longevity, thus contributing to the advancement of next-generation network paradigms. In the future, we plan to extend this work for a greater number of users in a distributed learning fashion by incorporating multi-agent RL, allowing us to observe the efficiency of the system on a larger scale. We aim to also examine the impact of device mobility during task offloading, adding another level of dynamicity during the evaluation.

CHAPTER 3

FEDORA: FEDERATED ENSEMBLE REINFORCEMENT LEARNING FOR DAG-BASED TASK OFFLOADING AND RESOURCE ALLOCATION IN MEC

Sangrez Khan[1], Amir Ali-Pour[1] Marios Avgeris[2], Julien Gascon-Samson[1], and Aris Leivadeas[1]

[1] Department of Software and IT Engineering, École de Technologie Supérieure, 1100 Notre-Dame Ouest, Montréal, Québec, Canada H3C 1K3
[2] Informatics Institute, Faculty of Science, University of Amsterdam (UvA), Amsterdam, The Netherlands

Paper published in *IEEE Internet of Things Journal*, August 2025

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.

3.1 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 (Zhou, Zhang, Wu, Dong & Leung, 2022b). 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 (Mach & Becvar, 2017). This proximity reduces data transmission delays, eases pressure on centralized cloud infrastructures, and enhances energy efficiency through localized processing (Mao *et al.*, 2017a). 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 (Saeik *et al.*, 2021b).

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) (Cao *et al.*, 2024b). In these scenarios, the execution of one task may depend on the completion of others, introducing significant challenges in determining optimal offloading strategies (Chen & Liu, 2021).

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 (Jiang, Dai, Xiao & Iyengar, 2022). 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 (Wang *et al.*, 2020a).

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 (Wang, Wang, Li, Leung & Taleb, 2020c). 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 (Liu *et al.*, 2021a). 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 (Feriani & Hossain, 2021). Techniques such as deep Q-networks (DQN) and actor-critic methods excel in navigating the high-dimensional state and action spaces common in MEC task offloading scenarios (Wu, Dinh, Fu, Lin & Quek, 2021). 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) (Konečnỳ *et al.*, 2016) 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 (Zhao, Li & He, 2023). 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 (Fan *et al.*, 2022). 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) (Veličković, 2017), 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 (Wang *et al.*, 2021a), (Li *et al.*, 2023).

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 (Khan, Avgeris, Gascon-Samson & Leivadeas, 2024), 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 (Mach & Becvar, 2017). 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 3.2 provides a state-of-the-art overview of the related literature. Section 2.3 introduces the system model and formulates the corresponding multi-objective optimization problem. In Section 3.4, 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 2.5 discusses the simulation results and performance evaluation. Finally, the conclusion of the paper is provided in Section 3.6.

3.2 Related Work

3.2.1 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, Uma & Subbalakshmi (2016) 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.* (2020) 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.* (2023) 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.* (2023a) 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.* (2022) 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.* (2023c) 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.

3.2.2 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, Tian, Wang & Lin (2024a) 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.* (2024) 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.

3.2.3 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.* (2024a) 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.* (2024a) combined FL with multi-agent reinforcement learning for VEC, demonstrating reductions in latency and energy consumption while improving task completion rates. Zhao *et al.* (2024a) extended federated DRL to vehicular networks, effectively managing both task offloading and resource allocation under privacy constraints. Zhou *et al.* (2024a) 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.* (2023) 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.* (2024) 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, Tong *et al.* (2024) proposed a privacy-preserving DAG task offloading framework in MEC environments using a federated deep Q-network

Table 3.1 Comparison of Related Works on Dependent MEC Task Offloading and Resource Allocation

Reference	**Solution Type**	**Energy Min.**	**Delay Min.**	**Res. Alloc.**	**DAG Mod.**	**Dyn. Net.**	**Privacy (FL)**
(Mahmoodi *et al.*, 2016)	Trad.	✓	✓		✓		
(Liu *et al.*, 2020)	Trad.		✓		✓		
(Xu *et al.*, 2023)	Trad.	✓	✓	✓	✓		
(Liu *et al.*, 2023a)	Trad.		✓		✓	✓	
(An *et al.*, 2022)	Trad.	✓	✓	✓	✓		
(Liu *et al.*, 2023c)	Trad.		✓	✓	✓	✓	
(Liu *et al.*, 2024a)	DRL	✓	✓	✓	✓	✓	
(Feng *et al.*, 2024)	DRL			✓	✓	✓	
(Xiao *et al.*, 2024a)	FL-DRL	✓	✓				✓
(Wu *et al.*, 2024a)	FL-DRL	✓	✓				✓
(Zhao *et al.*, 2024a)	FL-DRL		✓	✓			✓
(Zhou *et al.*, 2024a)	FL-DRL		✓				✓
(Zhao *et al.*, 2023)	FL-DRL	✓	✓				✓
(Shen *et al.*, 2024)	FL-DRL	✓	✓		✓		✓
(Tong *et al.*, 2024)	FL-DRL	✓	✓		✓		✓
FEDORA (Ours)	**FL-DRL**	✓	✓	✓	✓	✓	✓

(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 3.1 while various recent works have explored dependent task offloading using 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(Chen *et al.*, 2021a). 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.

3.3 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. 3.1. 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 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, $C^{i,s}_{m,n}$, where s indicates the designated local device or edge server. We explicitly assume the use of Orthogonal

Table 3.2 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

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 (Chen, Gu & Li,

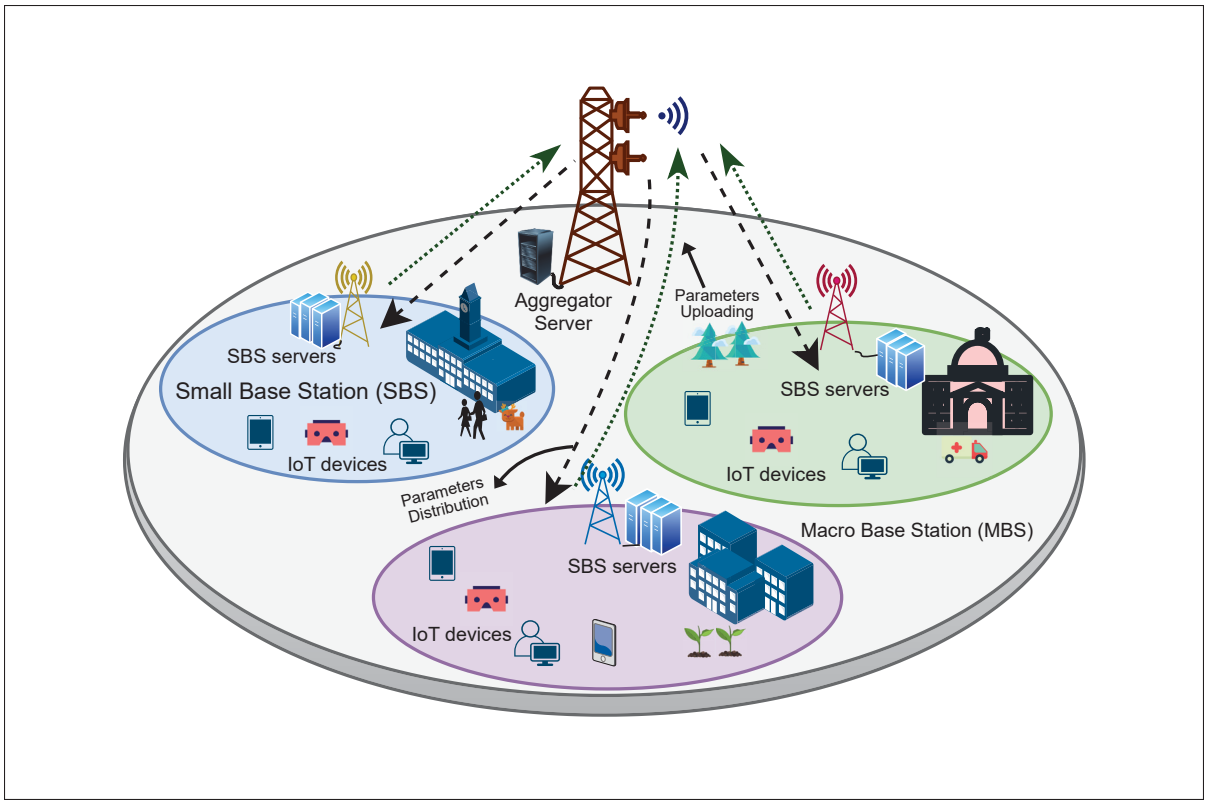

Figure 3.1 Detailed system model illustrating a Multi-tier IoT architecture integrated with MEC & FL

2022). 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 $\delta_{m,n}^{i,s}$, a binary task placement decision variable that

determines the execution location of task $x^i_{m,n} \in X_{m,n}$:

$$\delta^{i,s}_{m,n} = \begin{cases} 1, & \text{if task } x^i_{m,n} \text{ is executed} \\ & \text{on MEC server s at SBS } m, \\ 0, & \text{otherwise,} \end{cases} \tag{3.1}$$

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:

$$\sum_{s=0}^{|\mathbb{S}_m|} \delta^{i,s}_{m,n} = 1, \forall x^i_{m,n} \in X_{m,n}, \forall n \in \mathbb{N}_m, \forall m \in \mathbb{M}. \tag{3.2}$$

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:

$$0 \leq f^{i,s}_{m,n} \leq F^{m,s}_{\max} \delta^{i,s}_{m,n}, \forall s \in \mathbb{S}_m, \forall n \in \mathbb{N}_m, \forall m \in \mathbb{M}. \tag{3.3}$$

$F^{m,s}_{\max}$ 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.

3.3.1 Computation and Communication Model

3.3.1.1 Local Execution Model

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

$$T^{i,0}_{m,n} = \frac{c^i_{m,n}}{f^{i,0}_{m,n}}, \quad \forall m \in \mathbb{M}, \forall n \in \mathbb{N}_m, \forall i \in X_{m,n}, \tag{3.4}$$

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:

$$\Pi_{m,n}^{i,0} = \max_{j \in \text{pre}(i)} \{\delta_{m,n}^{j,0} C_{m,n}^{j,0} + \sum_{s=1}^{|\mathbb{S}_m|} \delta_{m,n}^{j,s} (C_{m,n}^{j,s} + \mathcal{T}_{m,n}^{j,d})\}, \tag{3.5}$$

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

$$\mathcal{T}_{m,n}^{i,d} = \frac{o_{m,n}^{i}}{R_{m,n}^{d}}, \quad \forall m \in \mathbb{M}, \forall n \in \mathbb{N}_m, \forall i \in X_{m,n}, \tag{3.6}$$

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

$$R_{m,n}^{d} = B_{m,n} \log_2(1 + \frac{P_{m,s}^{d} \mathcal{G}_{m,s}}{\sigma^2}). \tag{3.7}$$

The overall task completion time when executed locally is:

$$C_{m,n}^{i,0} = \Pi_{m,n}^{i,0} + T_{m,n}^{i,0}. \tag{3.8}$$

The local execution energy consumption is modeled as:

$$E_{m,n}^{i,0} = \kappa_{m,n} c_{m,n}^{i} (f_{m,n}^{i,0})^2, \tag{3.9}$$

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

3.3.1.2 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:

$$\mathcal{T}_{m,n}^{i,u} = \frac{o_{m,n}^{i}}{R_{m,n}^{u}}, \quad \forall m \in \mathbb{M}, \forall n \in \mathbb{N}_m, \forall i \in X_{m,n} \tag{3.10}$$

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

$$R^{u}_{m,n} = B_{m,n} \log_2(1 + \frac{P^{u}_{m,n} \mathcal{H}_{m,n}}{\sigma^2}). \tag{3.11}$$

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

$$T^{i,s}_{m,n} = \frac{c^{i}_{m,n}}{f^{i,s}_{m,n}}, \tag{3.12}$$

The task start time at the edge server is:

$$\Pi^{i,s}_{m,n} = \max_{j \in \text{pre}(i)} \{\delta^{j,0}_{m,n}(C^{j,0}_{m,n} + \mathcal{T}^{j,u}_{m,n}) + \sum_{s=1}^{|\mathbb{S}_m|} \delta^{j,s}_{m,n} C^{j,s}_{m,n}\}. \tag{3.13}$$

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

$$C^{i,s}_{m,n} = \Pi^{i,s}_{m,n} + T^{i,s}_{m,n}. \tag{3.14}$$

The corresponding uplink energy consumption is:

$$E^{i,u}_{m,n} = P^{u}_{m,n} \mathcal{T}^{i,u}_{m,n}. \tag{3.15}$$

3.3.2 Overall System Delay and Energy Consumption

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

$$T_{m,n} = \sum_{i=1}^{|X_{m,n}|} [\delta^{i,0}_{m,n} T^{i,0}_{m,n} \sum_{s=1}^{|\mathbb{S}_m|} \delta^{i,s}_{m,n} (\mathcal{T}^{i,u}_{m,n} + T^{i,s}_{m,n} + \mathcal{T}^{i,d}_{m,n})]. \tag{3.16}$$

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

$$E_{m,n} = \sum_{i=1}^{|X_{m,n}|} [\delta^{i,0}_{m,n} E^{i,0}_{m,n} + \sum_{s=1}^{|\mathbb{S}_m|} \delta^{i,s}_{m,n} E^{i,u}_{m,n}]. \tag{3.17}$$

3.3.3 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:

$$E_{\text{total}} = \sum_{m \in \mathcal{M}} \sum_{n \in \mathcal{N}_m} E_{m,n}. \tag{3.18}$$

Similarly, the total system completion delay is:

$$T_{\text{total}} = \sum_{m \in \mathcal{M}} \sum_{n \in \mathcal{N}_m} T_{m,n}. \tag{3.19}$$

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

$$\underset{\delta^{i,s}_{m,n},\, f^{i,s}_{m,n}}{\text{minimize}} \quad \sum_{m \in \mathbb{M}} \sum_{n \in \mathbb{N}_m} \left(T_{m,n} + \alpha E_{m,n} \right), \tag{3.20a}$$

$$\text{subject to} \quad E_{m,n} \leq E^{\max}_{m,n}, \quad \forall m \in \mathbb{M}, \forall n \in \mathbb{N}_m, \tag{3.20b}$$

$$T_{m,n} \leq T^{\max}_{m,n}, \quad \forall m \in \mathbb{M}, \forall n \in \mathbb{N}_m, \tag{3.20c}$$

$$\textstyle\sum_{s=0}^{|\mathbb{S}_m|} \delta^{i,s}_{m,n} = 1, \quad \forall m \in \mathbb{M}, \forall n \in \mathbb{N}_m, \forall i \in X_{m,n}, \tag{3.20d}$$

$$0 \leq f^{i,s}_{m,n} \leq F^{m,s}_{\max}\, \delta^{i,s}_{m,n}, \quad \forall m \in \mathbb{M}, \forall s \in \mathbb{S}_m, \forall n \in \mathbb{N}_m, \forall i \in X_{m,n}, \tag{3.20e}$$

$$\textstyle\sum_{n \in \mathbb{N}_m} \sum_{i \in X_{m,n}} \frac{D_{m,n,i}\, \delta^{i,s}_{m,n}}{R^{s}_{m,n}} \leq C_m, \quad \forall m \in \mathbb{M}, \forall s \in \mathbb{S}_m. \tag{3.20f}$$

where $0 \leq \alpha \leq 1$ in the objective function (Eq. 3.20a) is a weighting factor to balance energy and delay. Constraint (3.20b) ensures that the energy consumed by device n under SBS m does not exceed its maximum allowed energy budget $E^{\max}_{m,n}$. Constraint (3.20c) ensures that the

completion time of each task does not exceed the device's delay tolerance $T_{m,n}^{\max}$. Constraint (3.20d) guarantees that each task is executed at exactly one location (locally or MEC server), while (3.20e) limits CPU frequency allocation to the capacity of the selected execution location. Constraint (3.20f) ensures that the total uplink communication demand at each SBS does not exceed its capacity C_m. The problem formulated in Eq. 3.20a 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 (Li *et al.*, 2023). Moreover, these traditional methods fail to meet the strict real-time execution requirements critical for IoT applications (Mao *et al.*, 2017a). 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.

3.4 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- (Li *et al.*, 2020) 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

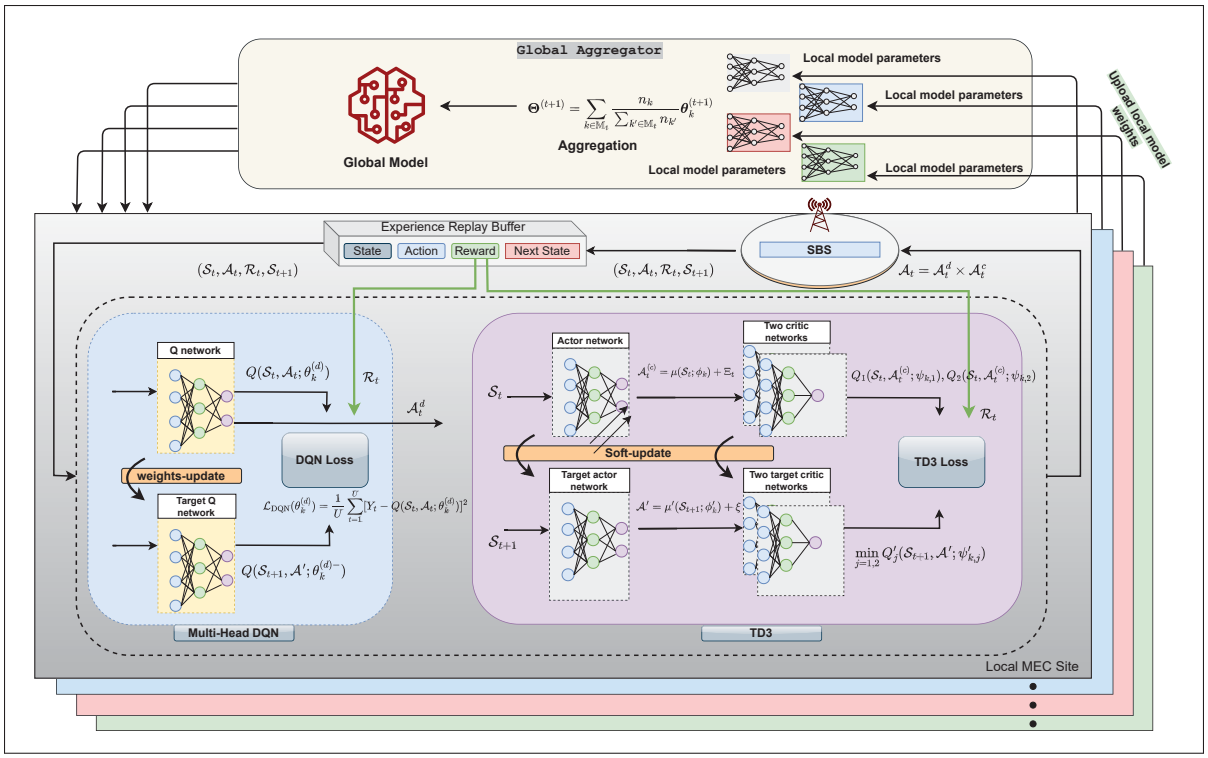

Figure 3.2 FEDORA Architecture

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, 3.2, 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.

3.4.1 Markov Decision Process Formulation

To effectively model our joint offloading and resource allocation problem within the proposed MEC environment, 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 $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma\}$ (Sutton & Barto, 2018b), 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.

3.4.1.1 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:

$$\begin{aligned}\mathcal{S}_t = \{&\widetilde{E}_n(t), O_n(t), H'_n(t), \mathcal{G}_{s\,n}(t), \\ &\mathcal{H}_{n\,s}(t) \mid \forall n \in \mathbb{N}, s \in \mathbb{S}\}.\end{aligned} \tag{3.21}$$

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

3.4.1.2 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:

$$\mathcal{A}_t = \{(\delta^{i,s}_{m,n}(t), f^{i,s}_{m,n}(t)) \mid \forall m \in \mathbb{M}, n \in \mathbb{N}_m, \\ i \in X_{m,n}, s \in \mathbb{S}_m \cup \{0\}\}, \tag{3.22}$$

3.4.1.3 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:

$$\mathcal{R}_t(\mathcal{S}_t, \mathcal{A}_t) = -(w_d \frac{T_{\text{avg}}}{T_{\max} + \epsilon} + w_e \frac{E_{\text{avg}}}{E_{\max} + \epsilon}) + \widetilde{R} \\ - \sum_{i \in \mathcal{S}} \max(0, \sum_j A_{i,j} - 1) - V_e \frac{\Delta E}{E^{\max}_{m,n}} - V_d \frac{\Delta T}{T^{\max}_{m,n}}, \tag{3.23}$$

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, $-V_e \frac{\Delta E}{E^{\max}_{m,n}}$ and $-V_d \frac{\Delta T}{T^{\max}_{m,n}}$, 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 $E^{\max}_{m,n}, T^{\max}_{m,n}$, respectively. These terms guide the agent towards consistently feasible and efficient decisions.

3.4.2 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 2.4.1), $\mathcal{A}_k = \mathcal{A}^d_k \times \mathcal{A}^c_k$ is a hybrid action space comprising discrete offloading decisions ($\mathcal{A}^d_k$) and continuous CPU resource allocations ($\mathcal{A}^c_k$), $\mathcal{R}_k$ 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^d_k, to approximate the optimal action-value function $Q^*(\mathcal{S}_t, \mathcal{A}_t)$, where $\mathcal{A}_t \in \mathcal{A}^d_k$. 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 $\boldsymbol{\Theta} = (\Theta^d, \Psi_1, \Psi_2, \Phi)$ are the aggregated versions of the local models, with $\boldsymbol{\Theta}' = (\Theta^{d-}, \Psi_1', \Psi_2', \Phi')$ denoting their corresponding target networks.

3.4.2.1 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:

$$Q(\mathcal{S}_t, \mathcal{A}_t; \theta_k^d) \approx Q^*(\mathcal{S}_t, \mathcal{A}_t), \tag{3.24}$$

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:

$$\theta_k^{d-} \leftarrow \tau\theta_k^d + (1-\tau_d)\theta_k^{d-}, \quad 0 < \tau_d \ll 1, \tag{3.25}$$

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:

$$Y_t = \mathcal{R}_t + \gamma \max_{\mathcal{A}'} Q(\mathcal{S}_{t+1}, \mathcal{A}'; \theta_k^{d-}), \tag{3.26}$$

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:

$$\mathcal{L}_{\text{DQN}}(\theta_k^d) = \frac{1}{U}\sum_{t=1}^{U}[Y_t - Q(\mathcal{S}_t, \mathcal{A}_t; \theta_k^d)]^2. \tag{3.27}$$

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:

$$\mathcal{A}_t^d = \begin{cases} \text{random action}, & \text{if } \zeta < \Xi^d, \\ \arg\max_{\mathcal{A}} Q(\mathcal{S}_t, \mathcal{A}; \theta_k^d), & \text{otherwise.} \end{cases} \tag{3.28}$$

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:

$$\mathcal{A}_t^d = \arg\max_{\mathcal{A}} Q(\mathcal{S}_t, \mathcal{A}; \theta_k^d). \tag{3.29}$$

3.4.2.2 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:

$$\mathcal{A}_t^c = \mu(\mathcal{S}_t; \phi_k) + \Xi_t, \quad \Xi_t \sim \mathcal{N}(0, \sigma^2), \tag{3.30}$$

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

$$\mathcal{A}_t^c = \text{clip}(\mathcal{A}_t^c, \mathcal{A}_{\min}^c, \mathcal{A}_{\max}^c). \tag{3.31}$$

Algorithm 3.1 FEDORA Training Phase

Input: $\mathbb{M}, \mathbb{N}_k, \mathbb{S}_k, \mathbb{D}_{k,n}, \mathbb{B}, \mathcal{B}, h, R, \gamma, \tau, \Xi, \Xi_t, \sigma^2, \bar{\sigma}^2, \mu, \mathcal{E}_k$
Initialize: $\boldsymbol{\Theta}^{(0)} = (\Theta^d, \Psi_1, \Psi_2, \Phi)$, $\boldsymbol{\Theta}'^{(0)} = (\Theta^{d-}, \Psi'_1, \Psi'_2, \Phi')$ $\boldsymbol{\theta}_k^{(0)} = (\theta_k^d, \psi_{k,1}, \psi_{k,2}, \phi_k)$, $\mathbb{B}_k$ $\forall k \in \mathbb{M}$
for $r = 1$ *to* R **do**
  for *each* $k \in \mathbb{M}$ *in parallel* **do**
    Synchronize: $\boldsymbol{\theta}_k^{(r)} \leftarrow \boldsymbol{\Theta}^{(r-1)}$
    for $e = 1$ *to* $\mathcal{E}_k$ **do**
      Sample initial state $\mathcal{S}_0$ randomly from $\mathcal{S}_k$
      for $t = 0$ *to* $\max_{n \in \mathbb{N}_k}(|X_{k,n}|) - 1$ **do**
        if $|\mathbb{B}_k| < U$ ***OR*** *system constraints not satisfied* **then**
          `// Random exploration`
          Select $\mathcal{A}_t^d \in \mathcal{A}_k^d$ randomly
          Select $\mathcal{A}_t^c \in \mathcal{A}_k^c$ randomly within $(\mathcal{A}_{\min}^c, \mathcal{A}_{\max}^c)$
        end if
        else
          `// Exploration-exploitation strategy`
          Select $\mathcal{A}_t^d$ using Eq. (3.29)
          Select $\mathcal{A}_t^c$ using Eq. (3.31)
        end if
        Execute action $\mathcal{A}_t = (\mathcal{A}_t^d, \mathcal{A}_t^c)$, observe reward $\mathcal{R}_{t+1}$ using Eq. (3.23), next state $\mathcal{S}_{t+1}$
        Store $(\mathcal{S}_t, \mathcal{A}_t, \mathcal{R}_{t+1}, \mathcal{S}_{t+1})$ in $\mathbb{B}_k$
        if $|\mathbb{B}_k| \geq U$ ***AND*** *system constraints satisfied* **then**
          `// Training step with FedProx regularization`
          Sample mini-batch of B transitions $\{(\mathcal{S}_i, \mathcal{A}_i, \mathcal{R}_{i+1}, \mathcal{S}_{i+1})\}_{i=1}^B$ from $\mathbb{B}_k$
          Compute DQN target using Eq. (3.26)
          Compute DQN loss using Eq. (3.27)
          Compute TD3 target using Eq. (3.33)
          Compute TD3 critic loss using Eq. (3.32)
          Compute total loss with FedProx using Eq. (3.37), Eq. (3.38)
          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
          if $t \mod \varphi = 0$ **then**
            Update actor using Eq. (3.34)
            Update target networks using Eq. (3.25), Eq. (3.36)
          end if
        end if
      end for
    end for
  end for
  Select participating agents $\mathbb{M}_r \subseteq \mathbb{M}$
  Aggregate at $\mathbb{A}$: $\boldsymbol{\Theta}^{(r)} = \sum_{k \in \mathbb{M}_r} \frac{n_k}{\sum_{k' \in \mathbb{M}_r} n_{k'}} \boldsymbol{\theta}_k^{(r)}$
  Broadcast $\boldsymbol{\Theta}^{(r)}$ to all agents $k \in \mathbb{M}$
end for
Output: $\boldsymbol{\theta}_k = (\theta_k^d, \psi_{k,1}, \psi_{k,2}, \phi_k)$ $\forall k \in \mathbb{M}$

The critic networks are trained by minimizing the MSE loss:

$$\mathcal{L}_{\text{TD3}}(\psi_{k,j}) = \frac{1}{U}\sum_{t=1}^{U}[y_t - Q_j(\mathcal{S}_t, \mathcal{A}_t^c; \psi_{k,j})]^2, \quad j \in \{1,2\}, \tag{3.32}$$

where the target y_t is computed using the target networks:

$$y_t = \mathcal{R}_t + \gamma \min_{j=1,2} Q'_j(\mathcal{S}_{t+1}, \mathcal{A}'; \psi'_{k,j}), \tag{3.33}$$

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:

$$\begin{aligned}\nabla_{\phi_k} J(\phi_k) = \frac{1}{U}\sum_{t=1}^{U} &\nabla_{\mathcal{A}} Q_1(\mathcal{S}_t, \mathcal{A}; \psi_{k,1})\big|_{\mathcal{A}=\mu(\mathcal{S}_t;\phi_k)} \\ &\cdot \nabla_{\phi_k}\mu(\mathcal{S}_t; \phi_k).\end{aligned} \tag{3.34}$$

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

$$\psi'_{k,j} \leftarrow \tau_c \psi_{k,j} + (1-\tau_c)\psi'_{k,j}, \quad j \in \{1,2\}, \tag{3.35}$$

$$\phi'_k \leftarrow \tau_c \phi_k + (1-\tau_c)\phi'_k, \tag{3.36}$$

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:

$$\mathcal{L}_k(\boldsymbol{\theta}_k) = \mathcal{L}_{\text{DQN}}(\theta_k^d) + \mathcal{L}_{\text{TD3}}(\psi_{k,1}, \psi_{k,2}, \phi_k). \tag{3.37}$$

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

$$\boldsymbol{\theta}_k^{(r)} = \arg\min_{\boldsymbol{\theta}_k} \left[\mathcal{L}_k(\boldsymbol{\theta}_k) + \frac{\mu}{2} \|\boldsymbol{\theta}_k - \boldsymbol{\Theta}^{(r-1)}\|_2^2 \right], \tag{3.38}$$

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.

3.4.3 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:

$$\min_{\boldsymbol{\theta}_k} \left[\mathcal{L}_k(\boldsymbol{\theta}_k) + \frac{\mu}{2} \|\boldsymbol{\theta}_k - \boldsymbol{\Theta}^{(r)}\|_2^2 \right], \tag{3.39}$$

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 $\boldsymbol{\Theta}^{(t+1)}$:

$$\boldsymbol{\Theta}^{(r+1)} = \sum_{k \in \mathbb{M}_t} \frac{n_k}{\sum_{k' \in \mathbb{M}_r} n_{k'}} \boldsymbol{\theta}_k^{(r+1)}, \tag{3.40}$$

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 (McMahan *et al.*, 2017a) but is applied to the heterogeneous updates produced by the proximal term modified local training. The aggregator broadcasts the updated global model $\mathbf{\Theta}^{(r+1)}$ back to all agents, which update their local parameters:

$$\boldsymbol{\theta}_k \leftarrow \mathbf{\Theta}^{(r+1)}, \quad \forall k \in \mathbb{M}. \tag{3.41}$$

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 3.1.

3.4.4 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 (Sutton & Barto, 2018b), 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.

3.5 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 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. 3.3 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.

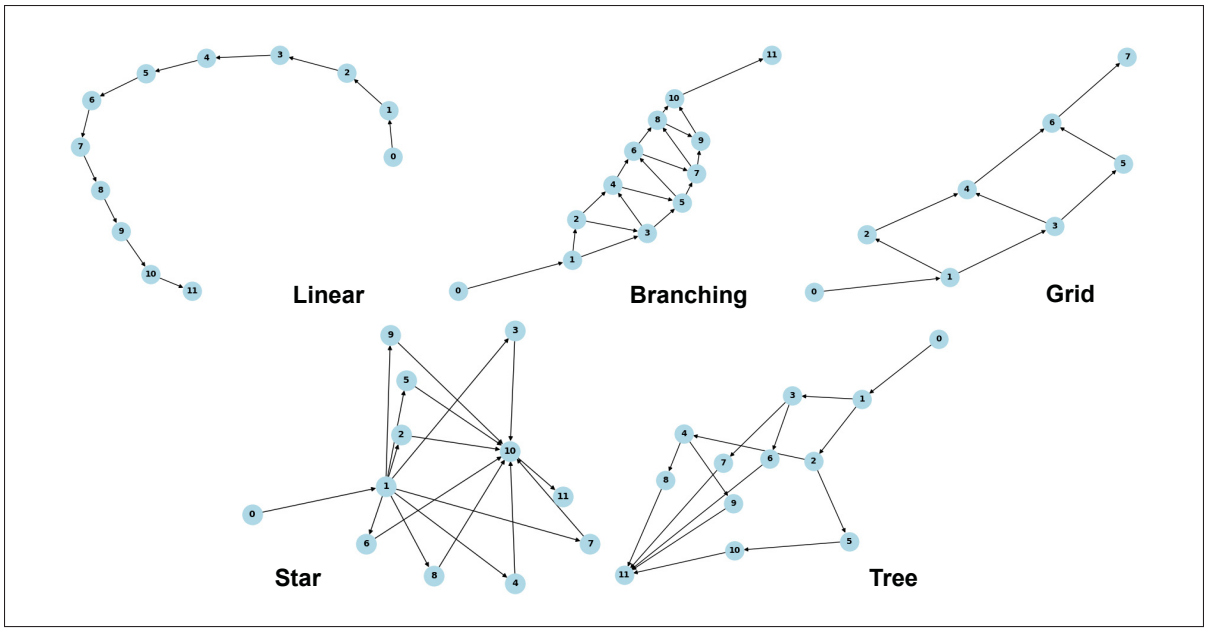

Figure 3.3 Type of DAGs used for evaluation

- Mixed: A hybrid of the above structures to ensure generalization.

Table 3.3 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 ($\lvert\mathbb{B}\rvert$)	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 3.3. 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 (Mao *et al.*, 2017a; Li *et al.*, 2023).

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:

3.5.1 Federated Optimization Baselines

- **FedAvg** (McMahan *et al.*, 2017a): Performs simple averaging of local model weights. It assumes IID data across clients and does not correct for drift in heterogeneous settings.
- **FedNova** (Wang *et al.*, 2020b): Normalizes local updates based on the number of local training steps, mitigating objective inconsistency in non-IID data distributions.
- **SCAFFOLD** (Karimireddy *et al.*, 2020): Incorporates control variates to address client drift and improve convergence under statistical heterogeneity.

3.5.2 FRL Baselines

- **FL-DQN** (Al-Naday *et al.*, 2024): A federated version of DQN, where clients train Q-value functions locally and synchronize periodically using federated averaging.

- **FL-DDPG** (Ouyang, Li & Chen, 2023): 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.

3.5.3 Heuristic Baselines

- **ALE** (Wang *et al.*, 2021a), (Nieto *et al.*, 2023), (Xiao *et al.*, 2022): All local execution, where all the task are executed at the local device.
- **AEE** (Wang *et al.*, 2021a),(Chen *et al.*, 2021a): All edge execution, where all tasks are offloaded to edge servers, regardless of system state.
- **ARE** (Xiao *et al.*, 2022), (Chen *et al.*, 2021a): All random execution, where offloading and resource decisions are made randomly, without learning or task awareness.

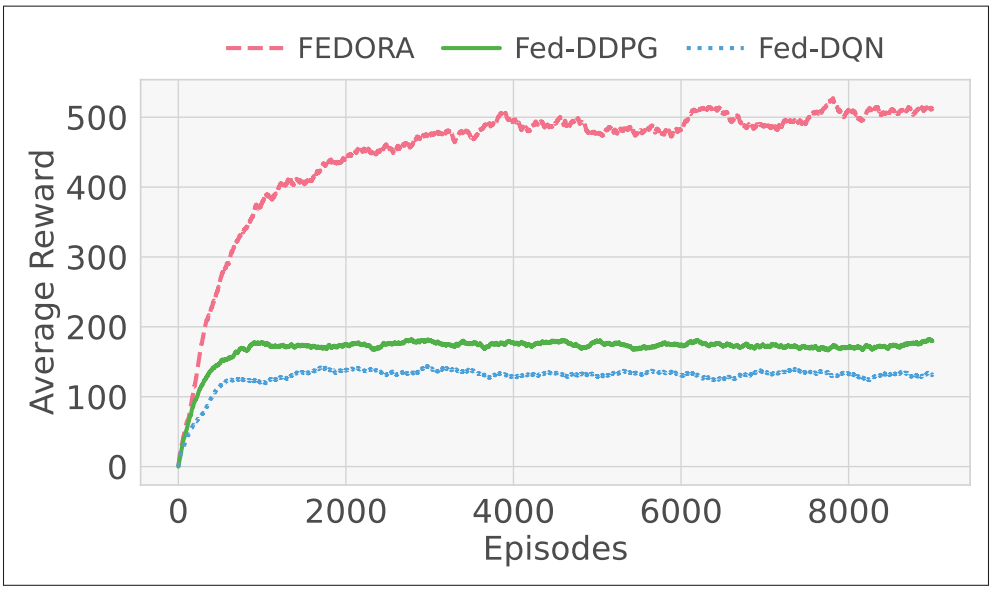

Figure 3.4 Average reward analysis for DRL training

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 illustrates the training performance of FEDORA in comparison to existing DRL, federated optimization baselines, and centralized

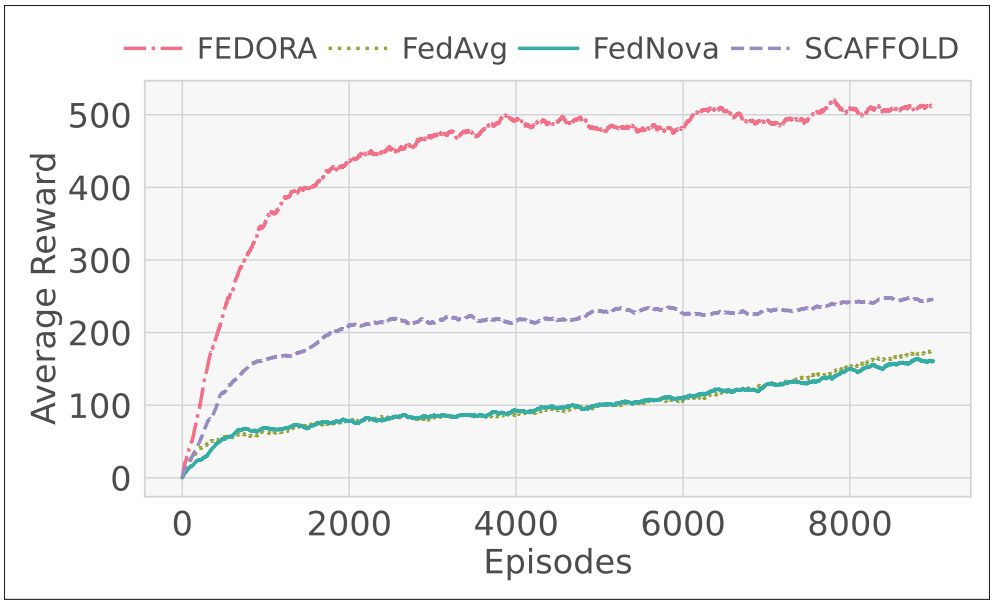

Figure 3.5 Average reward analysis for Federated training

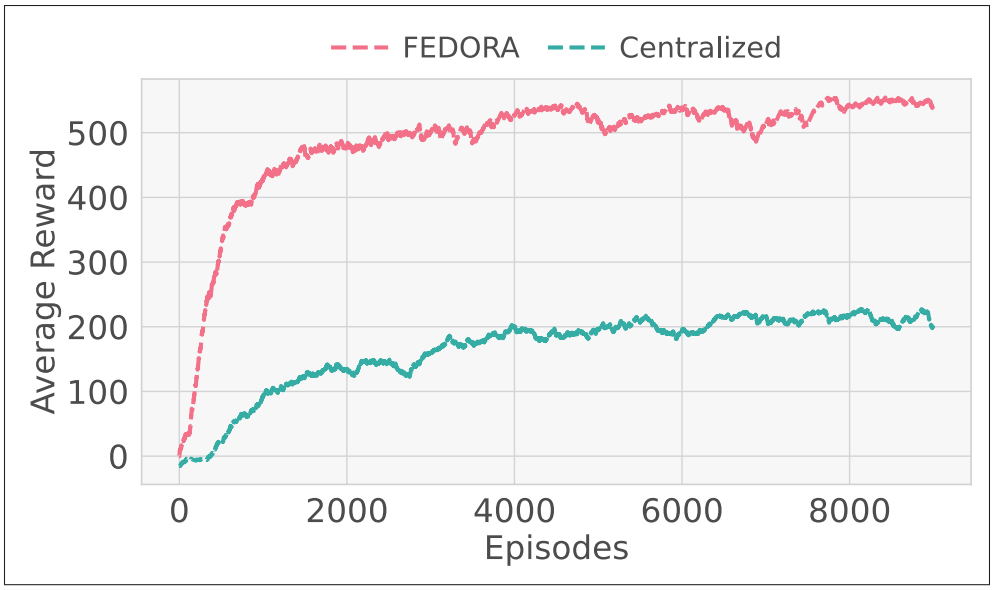

Figure 3.6 Training performance comparison: Federated versus Centralized Learning

methods, measured by the average reward obtained during training episodes. The centralized methods concern training without the distributed framework of FL. Fig. 3.4 highlights the performance comparison among the proposed FEDORA, Fed-DQN, and Fed-DDPG methods.

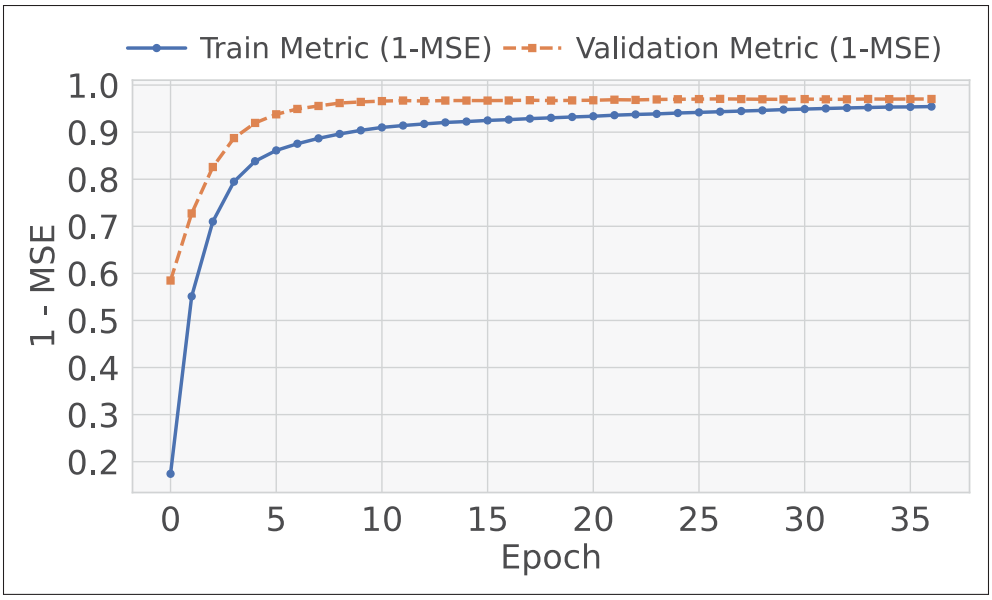

Figure 3.7 GAT model accuracy analysis for task dependency learning

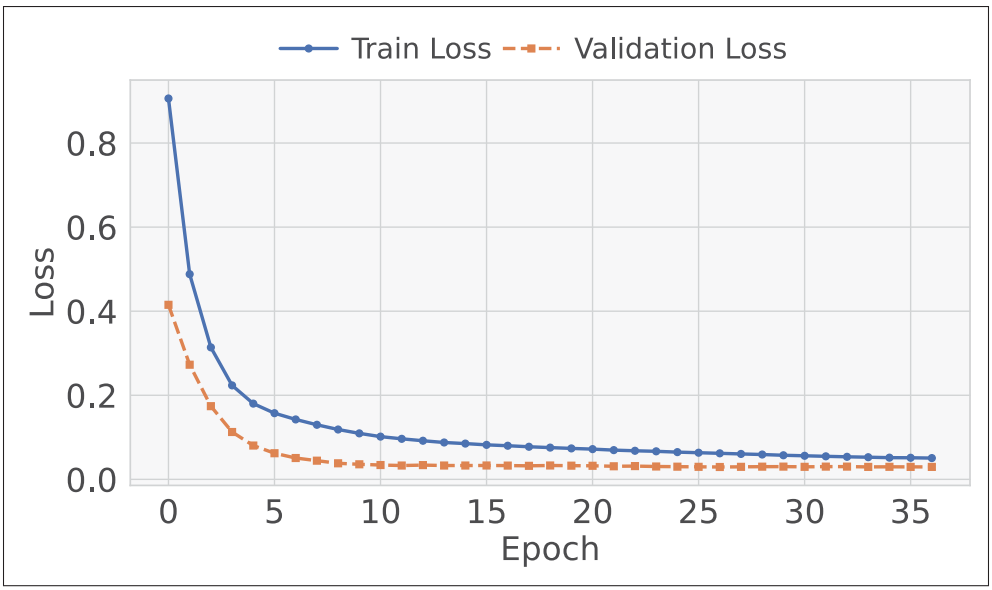

Figure 3.8 GAT model loss analysis for task dependency learning

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

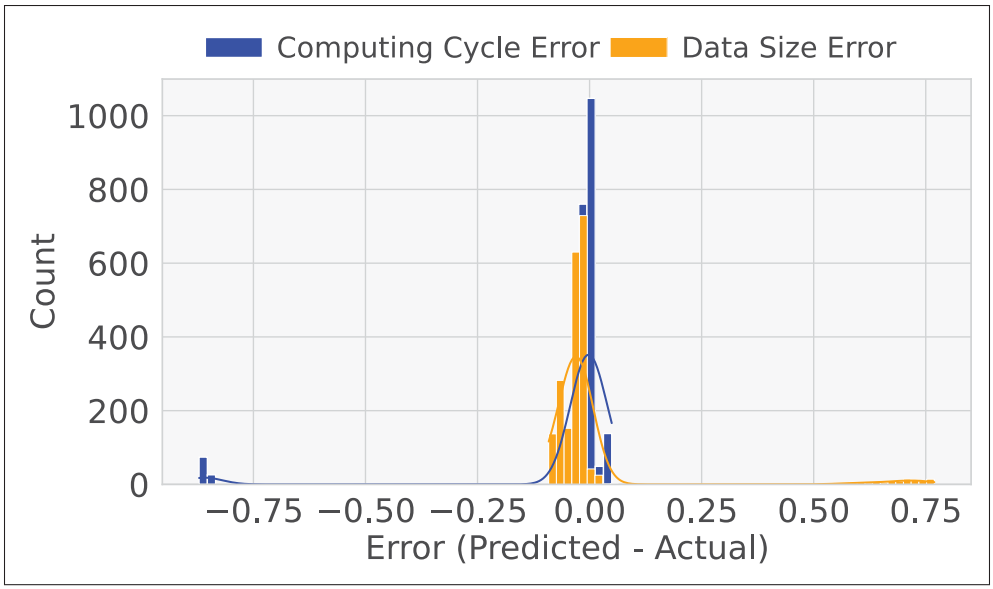

Figure 3.9 GAT model feature prediction and error distribution analysis

significantly lower performance levels, stabilizing at average 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. 3.5, 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. 3.6 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.

Next we 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. 3.7 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, continuous rise in accuracy, coupled with the decreasing loss and confirms the GAT model's effective learning of task

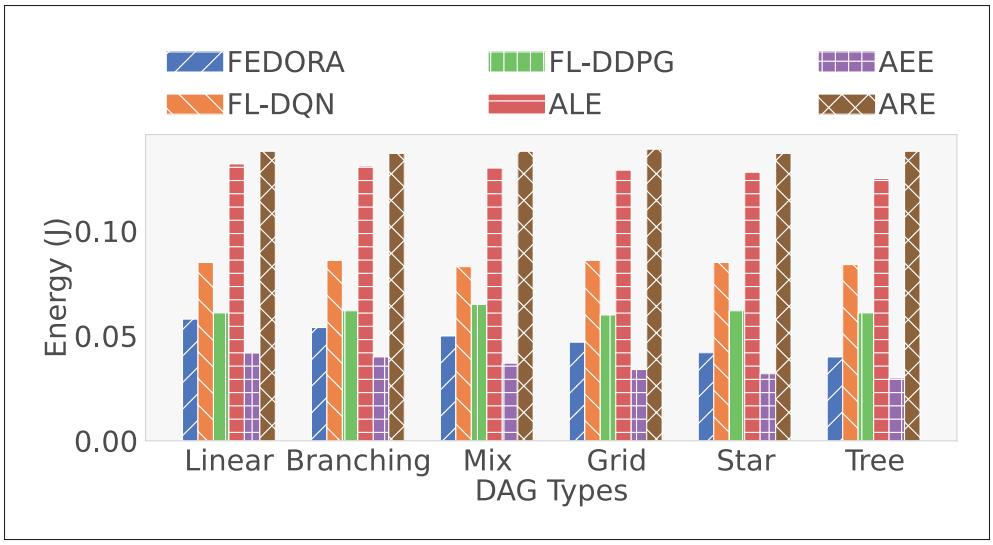

Figure 3.10 Energy consumption analysis across various DAG topologies

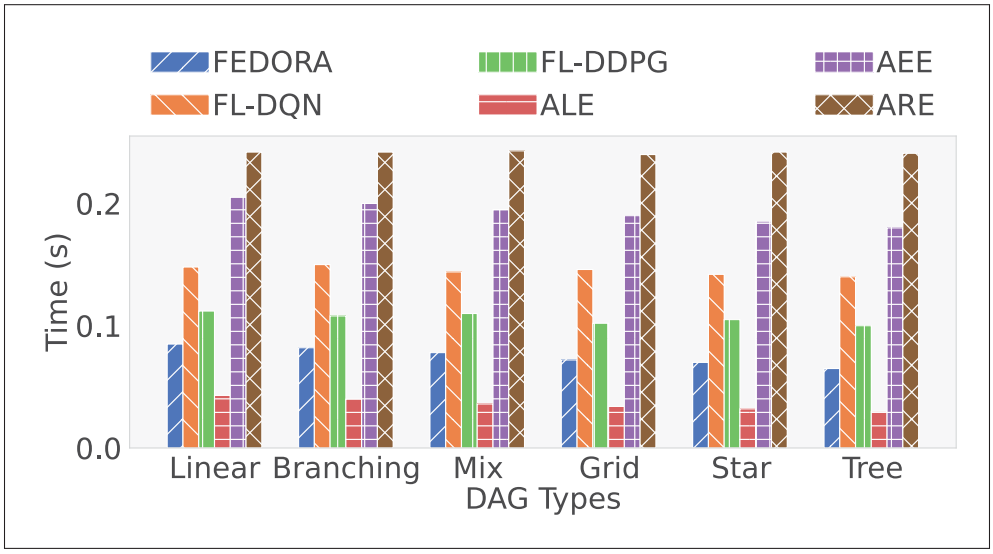

Figure 3.11 Task completion time analysis across various DAG topologies

features and dependencies. The small gap between the training and validation accuracy further indicates robust generalization, a critical factor for practical deployment.

Fig. 3.8 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

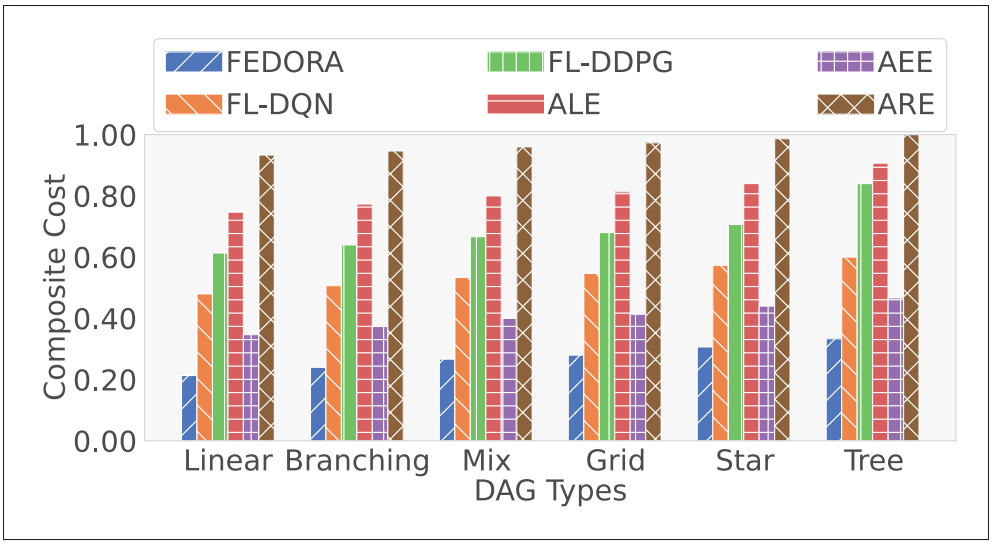

Figure 3.12 Overall system cost analysis across various DAG topologies

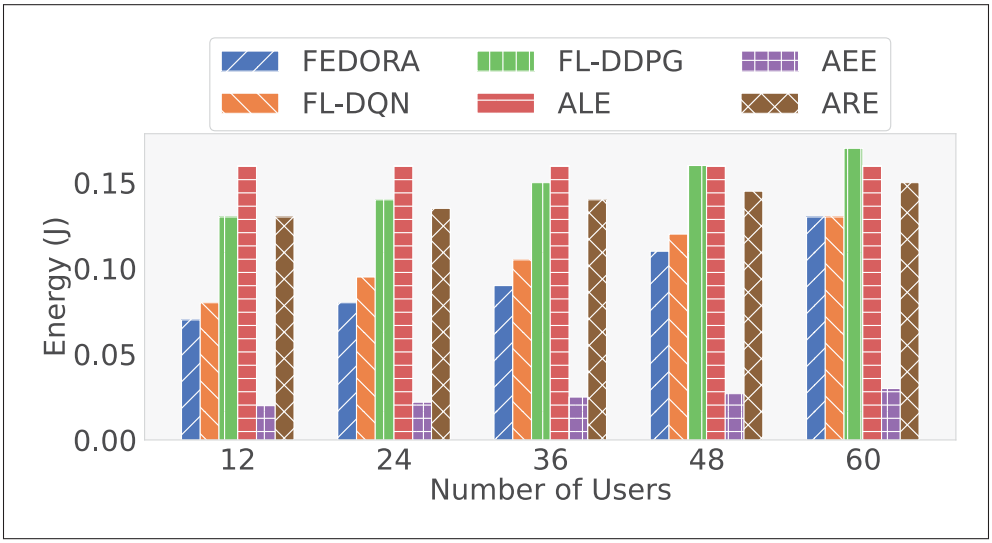

Figure 3.13 Energy consumption analysis for a varying number of users

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.

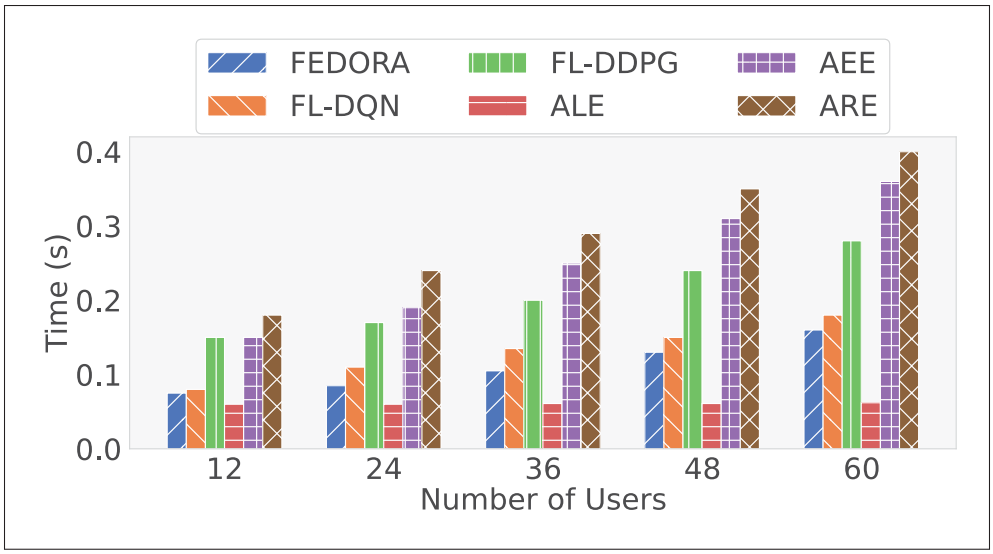

Figure 3.14 Task completion time analysis for a varying number of users

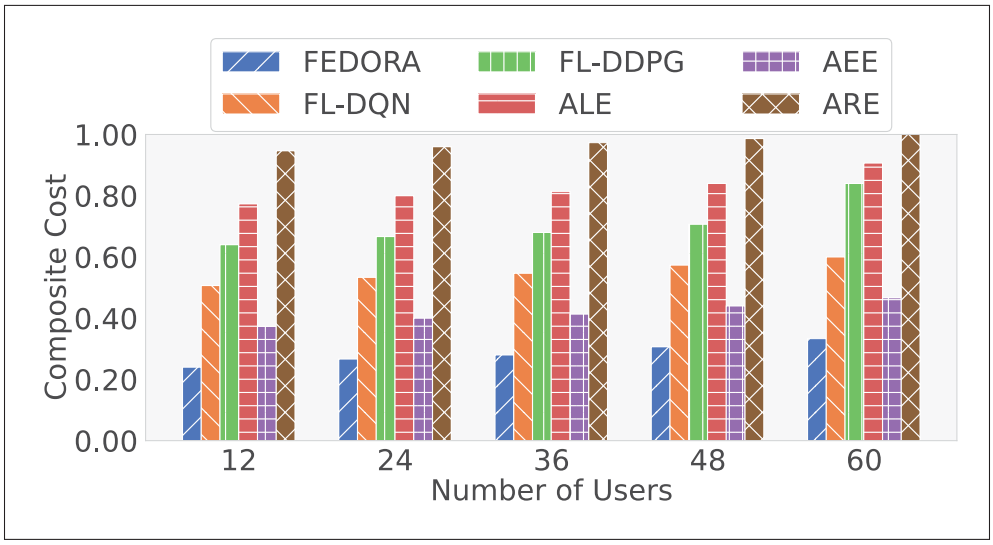

Figure 3.15 Overall system cost analysis for a varying number of users

Figure 3.9 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

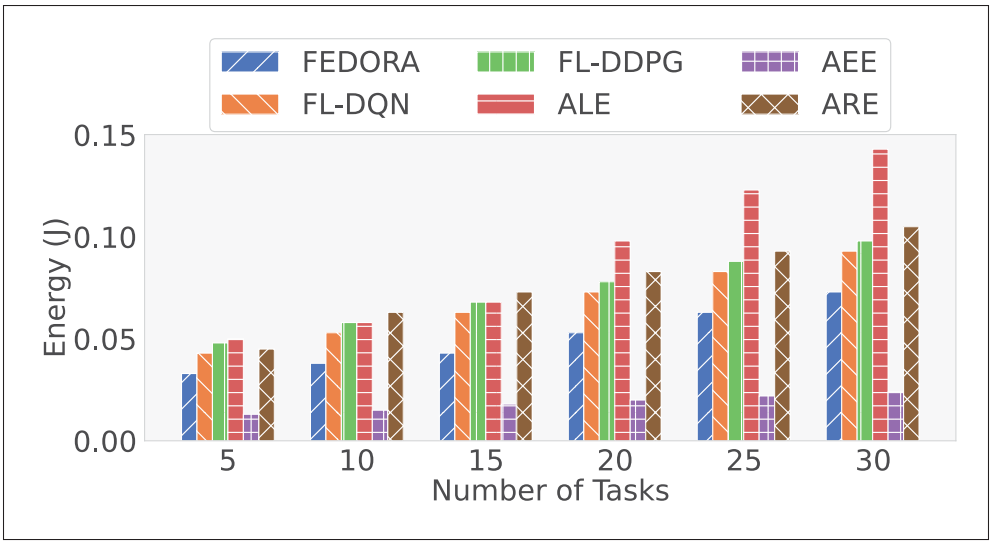

Figure 3.16 Energy performance analysis for a varying number of tasks

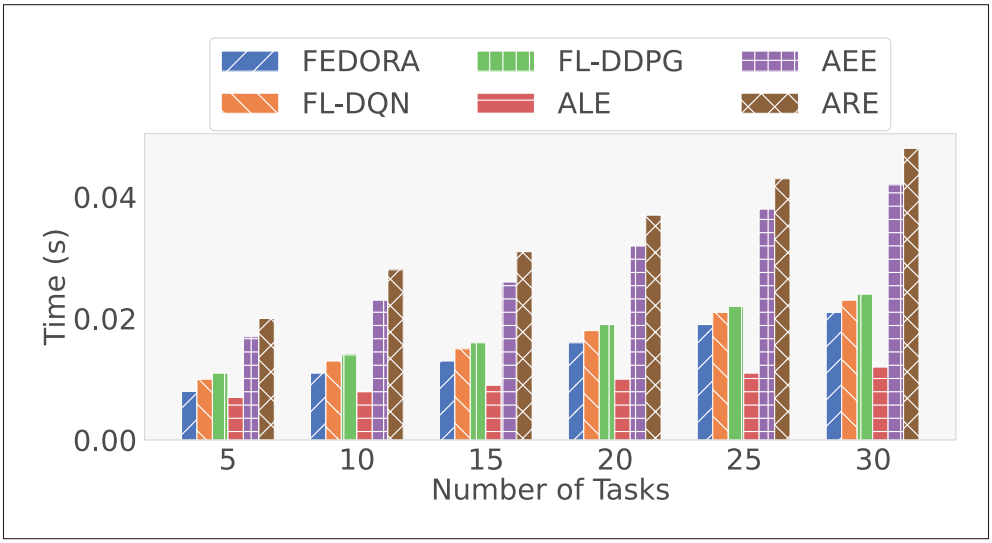

Figure 3.17 Task completion time performance analysis for a varying number of tasks

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.

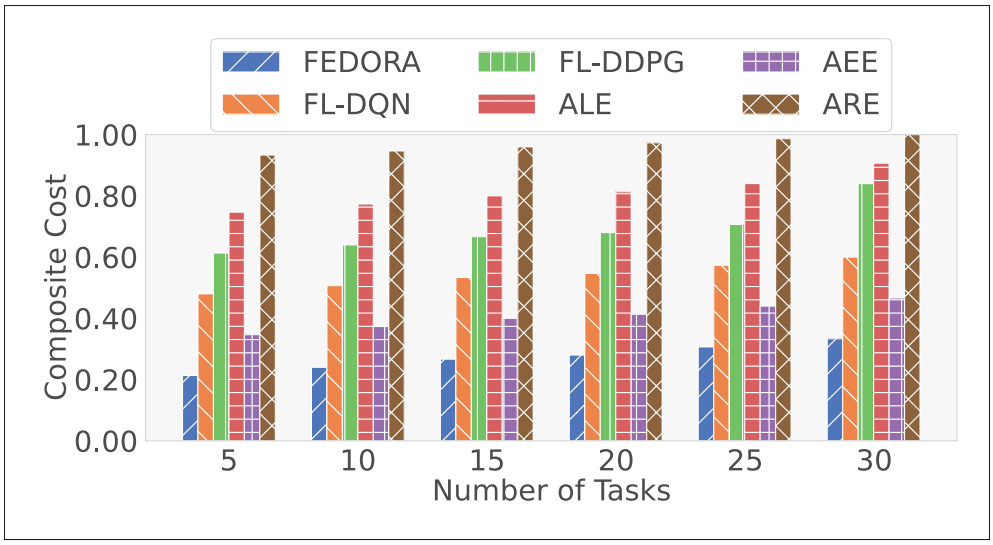

Figure 3.18 Overall system cost analysis for a varying number of tasks

To investigate FEDORA's adaptability across diverse workflow structures, we evaluate its performance on the various DAG types. Fig. 3.10 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. 3.11, 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 (3.23). Here, they represent counts of constraint violations for resource, energy and delay rather than penalty magnitudes. The composite cost results presented in Fig. 3.12 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, we evaluated the considered methods when varying the number of users from 12 to 60. Fig. 3.13 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. 3.14, 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. 3.15 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.

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. 3.16, 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. 3.17, 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. 3.18, 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.

3.6 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.

CHAPTER 4

DEPENDENT TASK OFFLOADING IN VEHICULAR EDGE COMPUTING USING TRAJECTORY-AWARE DEEP REINFORCEMENT LEARNING

Sangrez Khan[1], Marios Avgeris[2], Amir Ali-Pour[1], Julien Gascon-Samson[1], and Aris Leivadeas[1]

[1] Department of Software and IT Engineering, École de Technologie Supérieure, 1100 Notre-Dame Ouest, Montréal, Québec, Canada H3C 1K3
[2] Informatics Institute, Faculty of Science, University of Amsterdam (UvA), Amsterdam, The Netherlands

Paper accepted in *IEEE International Conference on Communications*, May 2026

Abstract

Vehicular Edge Computing (VEC) enables latency sensitive applications by offloading computation from moving vehicles to roadside units (RSUs). Scheduling dependent tasks in such environments is challenging because vehicles move quickly through RSU coverage zones and connectivity windows are short. We propose a trajectory aware task offloading framework that unifies short-term mobility forecasting with deep reinforcement learning. A mobility-aware transformer predicts the vehicle's future trajectory and derives RSU coverage intervals, while a Graph Attention Network (GAT) based actor–critic, trained via Proximal Policy Optimization (PPO), decides where each task in a directed acyclic graph (DAG) should execute. Experiments with real highway trajectories and a large synthetic DAG library show that the proposed approach consistently outperforms baselines in terms of delay, and energy.

4.1 Introduction

The rapid evolution of connected and autonomous vehicles has intensified the demand for low-latency, high-reliability computation to support applications such as real-time navigation, cooperative perception and augmented reality (Wang *et al.*, 2017). Vehicular Edge Computing (VEC) addresses this need by enabling vehicles to offload computational tasks to nearby RSUs

equipped with edge servers. By processing data close to the source, VEC reduces end-to-end latency and alleviates the load on remote cloud infrastructures (Liu *et al.*, 2021c).

However, task offloading in vehicular environments faces distinct challenges. Vehicles move at high speeds, causing rapidly fluctuating wireless channels, brief RSU coverage periods, and frequent handoffs (Mao, You, Zhang, Huang & Letaief, 2017b). These conditions make scheduling decisions more complex, particularly for applications with dependent subtasks represented as Directed Acyclic Graphs (DAGs), where the order of execution and data dependencies must be strictly followed (Chen, Jiao, Li & Fu, 2016; Mao, Zhang & Letaief, 2016). Poor scheduling in such scenarios can result in longer execution times, higher energy usage, and unnecessary handoffs.

Traditional offloading strategies, including heuristic rules and optimization based approaches, struggle to adapt to the fast changing topology of vehicular networks. While Deep Reinforcement Learning (DRL) has shown promise in enabling adaptive and data driven decision making, most existing works consider independent tasks and ignore future vehicle locations (Dai, Liu, Mo, Xu & Huang, 2022a). Consequently, these solutions overlook the link between predicted RSU coverage windows and the structure of DAG based applications. To address these limitations, we propose a trajectory aware DAG based task offloading framework that jointly considers predicted mobility patterns and task dependencies. Our contribution is threefold:

- We develop an enhanced Transformer model to predict short horizon vehicle trajectories from real highway data. The predicted positions are utilized to define RSU coverage intervals, giving the agent foresight into future connectivity.
- We design a Graph Attention Network (GAT) based actor–critic policy, trained with Proximal Policy Optimization (PPO), that makes DAG level offloading decisions by jointly encoding task features, RSU states and predicted coverage information.
- We conduct extensive experiments using real mobility traces and a large library of synthetic DAG workloads. Our method consistently outperforms all-local, all-remote, random and compute-aware strategies in terms of delay, and energy.

The rest of the paper is organized as follows, Section II reviews related work. Section III presents the system model and problem formulation. Section IV describes the proposed framework, Section V the experimental setup and results, and Section VI concludes.

4.2 Related Work

Mobility prediction is a fundamental enabler for efficient VEC offloading. Early works have relied on simple models such as constant velocity or Markov chains to estimate future vehicle positions (Liu, Mao, Fang, Zhu & Meng, 2021b). While computationally inexpensive, these models fail to capture the non-linear dynamics of real traffic scenarios, particularly under varying acceleration and lane-change behaviors.

More recent approaches employ machine learning-based predictors, including Recurrent Neural Networks (RNNs) and Convolutional Neural Networks (CNNs) (Park, Kim, Kang, Chung & Choi, 2018; Nikhil & Morris, 2018). These methods improve prediction accuracy but often struggle with long-term dependencies and irregular sampling intervals in vehicular data. Existing DAG scheduling methods in edge computing include list scheduling (Arabnejad & Barbosa, 2014) and critical-path heuristics (Kwok & Ahmad, 1996), but these often require precise a priori knowledge of task execution times and network conditions, which is unrealistic in vehicular contexts. Some studies have proposed integer linear programming (ILP) formulations for DAG offloading (de Queiroz, de Rezende & Barbosa, 2024), achieving optimality at the expense of scalability. More adaptive methods have incorporated metaheuristics such as particle swarm optimization (PSO) (You & Tang, 2021), though convergence speed can be an issue under fast changing topologies.

DRL has recently been applied to VEC to enable adaptive decision making under uncertainty (Hortelano *et al.*, 2023). Works employing Deep Q-Networks (DQN) (Dai, Liu, Mo, Xu & Huang, 2022b) or Deep Deterministic Policy Gradient (DDPG) (Chen & Wang, 2020) focus mainly on independent tasks and ignore inter task dependencies. Multi-agent DRL frameworks (Moon & Lim, 2022) have explored coordination among RSUs but typically assume perfect

knowledge of vehicle positions. Proximal Policy Optimization (PPO) has gained attention for its stability and sample efficiency (Schulman, Wolski, Dhariwal, Radford & Klimov, 2017b), yet its integration with DAG-based scheduling in vehicular environments remains unexplored. Few studies have jointly optimized mobility prediction and task offloading. A representative example is (Lv, Liu, Chen & Qin, 2021), where an LSTM-based predictor informs a heuristic offloading scheme. However, this approach suffers from error propagation and lacks a unified learning paradigm. In (Yan, Gu, Zhang & Jiao, 2023), predicted positions are used for coarse-grained RSU selection, without integrating predictions into the DRL state space.

From the discussion above, three key gaps can be identified: (i) mobility prediction models are often separated from task offloading decisions, resulting in inefficient scheduling; (ii) dependent task structures are frequently simplified or overlooked, reducing performance for complex applications; and (iii) current DRL methods lacks leverage graph-based neural networks to capture task dependencies in vehicular edge computing (VEC). Our work addresses these gaps by proposing a unified trajectory-aware DAG offloading framework that combines an enhanced transformer predictor with a GAT-PPO policy, tested in a vehicular edge scenario.

4.3 System Model

The system model, depicted in Fig. 3.1, describes a VEC environment along a multi-lane highway segment, comprising vehicles, RSUs, and a core network. The highway is a linear segment along the y-axis with vehicles moving in multiple lanes, divided into contiguous coverage zones of length L_{zone} meters. The set of RSUs is $\mathcal{R} = \{1, 2, \ldots, N_{RSU}\}$, where $N_{RSU} \in \mathbb{N}$ is the number of RSUs, each positioned at intervals of L_{zone} meters with coverage zone centered at $y_r = (r-1) \cdot L_{\text{zone}}$ for RSU $r \in \mathcal{R}$. Each RSU has a computing capacity of f_r in GHz, a wireless bandwidth of B_r in Mbps, and an instantaneous load of $u_r(t)$ representing resource utilization, connected to the core network via high-speed backhaul links with bandwidth B_{bh} in Gbps. The set of vehicles is $\mathcal{U} = \{u_1, u_2, \ldots, u_{|\mathcal{U}|}\}$, where $|\mathcal{U}| \in \mathbb{N}$ is the number of vehicles. Each vehicle $u \in \mathcal{U}$ at time t has a position $(x_u(t), y_u(t)) \in \mathbb{R}^2$, with $x_u(t)$ being the lane position and $y_u(t)$ being the longitudinal position, a local CPU frequency f_u, and a dynamic trajectory

defined by speed $s_u(t)$, acceleration $a_u(t)$, and lane changes. A vehicle u at $(x_u(t), y_u(t))$ is associated with RSU $r \in \mathcal{R}$ if its longitudinal position $y_u(t)$ lies within the scalar range $(r-1)L_{\text{zone}} \leq y_u(t) < rL_{\text{zone}}$, which defines the coverage interval of RSU r along the highway. Applications on vehicle u are modeled as DAGs $G_u = (V_u, E_u)$, where $V_u = \{v_1, v_2, \ldots, v_{|V_u|}\}$ is the set of tasks with $|V_u| \in \mathbb{N}$, and $E_u \subseteq V_u \times V_u$ represents precedence constraints, with $(v_i, v_j) \in E_u$ indicating task v_j depends on v_i's completion and output. Each task $v_i \in V_u$ is defined by a tuple (c_i, d_i, t_i^{dl}), where c_i is the required computation cycles, d_i is the output data size, and t_i^{dl} is the strict per-task deadline in seconds, necessitated by the highly dynamic topology and limited vehicle communication windows in the VEC environment.

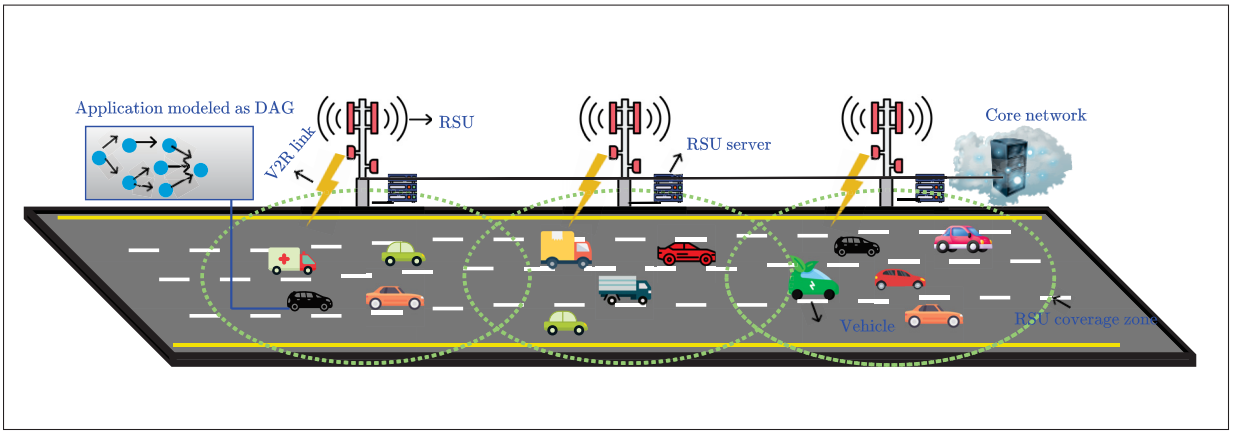

Figure 4.1 Trajectory aware vehicular edge computing system model with DAG based task offloading

The data rate $R_{u,r}(t)$ between vehicle $u \in \mathcal{U}$ and RSU $r \in \mathcal{R}$ at time t is given as:

$$R_{u,r}(t) = B_r \cdot \log_2\left(1 + \frac{P_t G_{u,r}(t)}{N_0 + I_r(t)}\right). \tag{4.1}$$

where P_t is the transmit power in W, $G_{u,r}(t)$ is the channel gain, N_0 is the noise power in W, and $I_r(t)$ is the inter-cell interference in W. The rate $R_{u,r}(t)$ in Mbps degrades with distance and load. Inter-RSU communication is supported by high-speed fiber backhaul links with bandwidth

B_{bh} in Gbps. The transmission delay for data of size d over a V2R link is:

$$T_{\text{tx}}(d) = \frac{d}{R_{u,r}(t)}. \tag{4.2}$$

and over a backhaul link between two RSUs is:

$$T_{\text{bh}}(d) = \frac{d}{B_{\text{bh}}}. \tag{4.3}$$

Connectivity between vehicle u and RSU r is guaranteed at the time of data transfer by the action masking mechanism, which restricts offloading decisions to RSUs within the vehicle's predicted coverage window; inter-RSU output forwarding is handled via the backhaul links and whose latency is negligible relative to the V2R wireless link. For each task $v_i \in V_u$, the local and remote execution times are given as:

$$T_i^k = \begin{cases} \frac{c_i}{f_u}, & k = 0, \\ \frac{c_i}{f_k}, & k \in \mathcal{R}. \end{cases} \tag{4.4}$$

The start and finish times S_i and F_i are then defined as

$$S_i = \max_{v_j \in \text{pred}(i)} \left\{ F_j + T_{tx}(d_j, \text{loc}(j), \text{loc}(i)) \right\}, \tag{4.5}$$

$$F_i = S_i + \sum_{k \in \{0\} \cup \mathcal{R}} a_i^k \, T_i^k, \tag{4.6}$$

where $\text{pred}(i) \subseteq V_u$ is the set of predecessor tasks, and $\text{loc}(l)$ is the execution node of task v_l. The energy consumption for local execution of task $v_i \in V_u$ is $E^{\text{u}}(v_i) = c_i \cdot \kappa_{\text{cpu}}$, where κ_{cpu} is the CPU energy per cycle. For offloaded tasks, energy consumption is mainly due to the

transmission of output data to dependent tasks at remote locations. The total energy E_{total} is:

$$\begin{aligned} E_{\text{total}} = & \sum_{u \in \mathcal{U}} \sum_{v_i \in V_u} a_i^0 \cdot c_i \cdot \kappa_{\text{cpu}} \\ & + \sum_{u \in \mathcal{U}} \sum_{(v_j, v_i) \in E_u} \mathbb{1}_{\{a_j^0 = 1, a_i^k \neq 0\}} \cdot d_j \cdot \kappa_{\text{tx}}, \end{aligned} \tag{4.7}$$

where κ_{tx} is the energy consumed per transmitted bit, and $\mathbb{1}$ is the indicator function.

4.3.1 Problem Formulation

For each vehicle $u \in \mathcal{U}$ and tasks v_i the binary decision variable is:

$$a_i^k = \begin{cases} 1, & \text{if task } v_i \text{ is executed at RSU } k, \\ 0, & \text{if task } v_i \text{ is executed locally on the vehicle} \end{cases} \tag{4.8}$$

where $k \in \{0\} \cup \mathcal{R}$, with $k = 0$ for local execution on u. The objective is to minimize the weighted sum of total completion time and energy consumption:

$$\underset{\{a_i^k\}}{\text{minimize}} \quad \lambda_t \sum_{u \in \mathcal{U}} T_{\text{tot},u} + \lambda_e \sum_{u \in \mathcal{U}} E_{\text{total},u}, \tag{4.9a}$$

$$\text{subject to} \quad F_i \leq t_i^{\text{dl}}, \quad \forall v_i \in V_u, \forall u \in \mathcal{U}, \tag{4.9b}$$

$$\textstyle\sum_{k \in \{0\} \cup \mathcal{R}} a_i^k = 1, \quad \forall v_i \in V_u, \forall u \in \mathcal{U}, \tag{4.9c}$$

$$a_i^k \in \{0, 1\}, \quad \forall v_i \in V_u, \forall k \in \{0\} \cup \mathcal{R}. \tag{4.9d}$$

where $T_{\text{tot},u} = \max_{v_i \in V_u} F_{i,u}$ is the completion time for vehicle u, $E_{\text{total},u}$ is the energy consumption (Eq. 4.7), and λ_t, λ_e are weighting coefficients that satisfy $\lambda_t + \lambda_e = 1$, with typical values between (0,1). This problem is NP-hard because it involves discrete offloading decisions

under precedence and deadline constraints which can be solved using a reinforcement learning framework to sequentially determine $\{a_i^k\}$.

4.4 Trajectory Aware DAG Offloading Framework

Our framework integrates a Transformer-based mobility prediction module and a GAT-based DAG aware task offloading policy trained with PPO. By incorporating predicted mobility into offloading decisions, our framework operates in two sequential stages: (i) a Transformer-based mobility prediction stage that forecasts short-horizon trajectories and RSU coverage windows, and (ii) a GAT-PPO offloading stage that uses these predictions to make task-assignment decisions. These stages are executed sequentially for each scheduling interval, with the prediction stage feeding updated mobility information to the offloading stage.

4.4.1 Trajectory Prediction

This stage forecasts short horizon vehicular mobility to estimate RSU coverage windows, critical for avoiding misaligned task scheduling that potentially increases latency and energy consumption. For vehicle $u \in \mathcal{U}$ at time t, the state is defined as

$$\mathbf{s}_u(t) = [x_u(t), y_u(t), s_u(t), \theta_u(t), a_u(t)], \tag{4.10}$$

where $x_u(t), y_u(t)$ denote the vehicle's position coordinates, $s_u(t)$ is its speed, $\theta_u(t)$ is the heading angle that is direction of motion relative to the x-axis, measured in radians, and $a_u(t)$ is the acceleration. The input to the Transformer-based mobility predictor is a sequence of w historical states:

$$X_{u,t} = [\mathbf{s}_u(t - w + 1), \ldots, \mathbf{s}_u(t)] \in \mathbb{R}^{w \times 5}. \tag{4.11}$$

The goal is to predict the next H positions:

$$\hat{Y}_{u,t} = \{(\hat{x}_{u,t+1}, \hat{y}_{u,t+1}), \ldots, (\hat{x}_{u,t+H}, \hat{y}_{u,t+H})\}, \tag{4.12}$$

mapped to RSU $r \in \mathcal{R}$ if $\hat{y}_{u,t+h} \in [(r-1) \cdot L_{\text{zone}}, r \cdot L_{\text{zone}})$. The sequence $X_{u,t}$ is projected into a d-dimensional embedding space:

$$X'_{u,t} = W_p X_{u,t} + b_p, \tag{4.13}$$

where $W_p \in \mathbb{R}^{d \times 5}$, $b_p \in \mathbb{R}^d$ are learnable parameters. Sinusoidal positional encodings with learnable scaling and bias preserve temporal ordering. The encoded sequence is processed through N Transformer encoder blocks, each with multi-head self-attention and a two-layer feedforward network (FFN) with residual connections and layer normalization. For input $X'_{u,t} \in \mathbb{R}^{w \times d}$, attention computes queries $Q \in \mathbb{R}^{w \times d_k}$, keys $K \in \mathbb{R}^{w \times d_k}$ and values $V \in \mathbb{R}^{w \times d_v}$, with output:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V. \tag{4.14}$$

The FFN uses the Gaussian Error Linear Unit (GELU):

$$\text{FFN}(x) = \text{GELU}(xW_1 + b_1)W_2 + b_2, \tag{4.15}$$

where W_1, W_2, b_1, b_2 are learnable parameters. Outputs are aggregated via adaptive average pooling, and a two-layer FFN maps to H coordinate pairs $(\hat{x}_{u,t+h}, \hat{y}_{u,t+h})$. The predicted $\hat{Y}_{u,t}$ yields RSU coverage intervals for Stage 2.

4.4.2 Task Offloading via Reinforcement Learning

The task offloading problem is formulated as a sequential DRL process for each vehicle $u \in \mathcal{U}$ with DAG G_u. The DRL agent sequentially assigns each task $v_i \in V_u$ to either local execution on the vehicle or to an RSU, respecting the DAG's dependencies and the system model's constraints

($F_i \leq t_i^{\mathrm{dl}}$, $\sum_k a_i^k = 1$). The agent leverages mobility predictions from Stage 1 to anticipate RSU availability, optimizing for the system model's objective. In the next section we will define the problem as a Markov Decision Process (MDP) and then solve it using DRL.

4.4.2.1 State Space

At decision step t for task v_i, the reinforcement learning state $\mathbf{s}_{u,t}^{\mathrm{RL}} \in \mathcal{S}$ for vehicle u encapsulates task, mobility, and infrastructure information:

$$\mathbf{s}_{u,t}^{\mathrm{RL}} = (\mathbf{h}_i, \mathbf{c}_r, \mathbf{z}_u, u_r(t)), \tag{4.16}$$

where $\mathbf{h}_i \in \mathbb{R}^d$ is the embedding of task v_i from a GAT applied to G_u, capturing dependencies, $\mathbf{c}_r = [r, r_{\mathrm{next}}, t_{\mathrm{cov}}, \bar{u}_r]$ is the RSU context, with $r \in \mathcal{R}$ as the current RSU, $r_{\mathrm{next}} \in \mathcal{R}$ as the predicted next RSU based on $\hat{Y}_{u,t}$, t_{cov} as the estimated coverage duration for RSU r, and $\bar{u}_r$ as the time-averaged load of RSU r; $\mathbf{z}_u$ is the trajectory embedding derived from $\hat{Y}_{u,t}$ via Stage 1's pooling layer; and $u_r(t)$ is the instantaneous load of RSU r, reflecting current resource utilization.

4.4.2.2 Action Space

For task v_i, the action $a_i \in \mathcal{A}$ selects the execution node:

$$a_i \in \{0\} \cup \mathcal{R}, \tag{4.17}$$

where $a_i = 0$ denotes local execution on vehicle u with execution time $T_i^{\mathrm{loc}} = \frac{c_i}{f_u}$, and $a_i = r \in \mathcal{R}$ denotes offloading to RSU r with execution time $T_i^r = \frac{c_i}{f_r}$. This corresponds to setting $a_i^k = 1$ for the chosen $k \in \{0\} \cup \mathcal{R}$ in the system model, ensuring $\sum_k a_i^k = 1$. The agent respects precedence constraints by only selecting a_i for task v_i after all predecessor tasks $v_j \in \mathrm{pred}(i)$ have been assigned and completed. After selecting an action a_i for task v_i, the decision is stored in the vector $\mathbf{a}$, which records the execution node of every task in the DAG and is later used to evaluate delay and energy metrics.

4.4.2.3 Reward Function

After executing the DAG G_u under action sequence $\{a_i\}$, the reward for vehicle u is:

$$r_u = \lambda_t \cdot \frac{1}{1 + T_{\text{tot},u}} + \lambda_e \cdot \frac{1}{1 + E_{\text{total},u}}, \tag{4.18}$$

where $T_{\text{tot},u} = \max_{v_i \in V_u} F_{i,u}$ is the total completion time of vehicle u, $E_{\text{total},u}$ is the corresponding energy consumption, and λ_t and λ_e are weighting coefficients, as defined in the System Model (Eq. (4)–(5) for S_i, F_i and Eq. (3) for $E_{\text{total},u}$). The weights λ_t, λ_e balance latency and energy, aligning with the system model's objective. The inverse form promotes minimization of $T_{\text{tot},u}$ and $E_{\text{total},u}$.

4.4.2.4 Policy Network

The agent employs an actor-critic policy $\pi_\theta(a_i|s_{u,t})$ parameterized by θ, implemented using a GAT to model the DAG G_u. For each task v_i, the GAT computes $\mathbf{h}_i$ by aggregating features from neighboring tasks $v_j \in \text{pred}(i)$ and successors, incorporating (c_i, d_i, t_i^{dl}) and edge weights based on d_j for $(v_j, v_i) \in E_u$. The state $s_{u,t}$ is formed by concatenating $\mathbf{h}_i$, $\mathbf{c}_r$, $\mathbf{z}_u$, and $u_r(t)$, enabling the agent to account for task dependencies, RSU availability, and predicted mobility. The actor head outputs logits $\boldsymbol{\ell}_i$ over the action space $\{0\} \cup \mathcal{R}$ for a_i, which are masked to ensure feasibility based on RSU availability in C and completion of predecessors $v_j \in \text{pred}(i)$ ($F_j \leq t$). The critic head estimates the value function $V(s_{u,t})$ to reduce variance in policy updates. PPO maximizes the clipped surrogate objective with entropy regularization:

$$\begin{aligned} L^{\text{PPO}}(\theta) = \mathbb{E}_t \Big[& \min\left(\rho_t(\theta) A_t,\ \text{clip}(\rho_t(\theta),\ 1-\epsilon,\ 1+\epsilon)\, A_t\right) \\ & + \beta\, \mathcal{H}[\pi_\theta(\cdot \mid s_t)] \Big]. \end{aligned} \tag{4.19}$$

where $\rho_t(\theta) = \frac{\pi_\theta(a_i|s_{u,t})}{\pi_{\theta_{\text{old}}}(a_i|s_{u,t})}$ is the probability ratio, A_t is the advantage computed via generalized advantage estimation using reward r_u, $H[\pi_\theta]$ is the policy entropy, and ϵ, β are hyperparameters. This architecture enables the agent to make informed offloading decisions by integrating

Algorithm 4.1 Trajectory Aware DAG Offloading with PPO

Input: H, DAG $G_u = (V_u, E_u)$, RSU set $\mathcal{R}$
Output: Offloading decisions $\{a_i^k\}$ for all tasks $v_i \in V_u$
// Stage 1: Trajectory Prediction
Predict future positions $\{(\hat{x}_{u,t+h}, \hat{y}_{u,t+h})\}_{h=1}^{H}$ using Transformer $\Rightarrow \hat{Y}_{u,t}$
Compute trajectory embedding $\mathbf{z}_u \leftarrow \text{PoolingLayer}(\hat{Y}_{u,t})$
Estimate RSU coverage intervals $C \leftarrow \text{RSUCoverageWindows}(\hat{Y}_{u,t}, \mathcal{R})$
// Stage 2: GAT-PPO Policy Decision
Normalize task features $\mathbf{X} \leftarrow \text{NormalizeFeatures}(G_u)$
Obtain RSU states $\mathbf{S} \leftarrow \text{GetRSUStates}(\mathcal{R})$
Extract RSU context $\mathbf{c}_r \leftarrow \text{ExtractContext}(C, \mathbf{S})$
Initialize empty decision vector $\mathbf{a}$
for *each task* $v_i \in V_u$ *in topological order* **do**
    Compute task embedding $\mathbf{h}_i \leftarrow \text{GAT}(\mathbf{X}, E_u)[i]$
    Form complete state $\mathbf{s}_{u,t} \leftarrow \text{Concat}(\mathbf{h}_i, \mathbf{c}_r, \mathbf{z}_u, u_r(t))$
    Obtain action logits $\boldsymbol{\ell} \leftarrow \text{Actor}(\mathbf{s}_{u,t})$
    Mask infeasible actions $\boldsymbol{\ell}^m \leftarrow \text{ActionMask}(\boldsymbol{\ell}, C, \mathbf{S}, \text{pred}(i))$
    Sample action $a_i \sim \text{Categorical}(\text{Softmax}(\boldsymbol{\ell}^m))$
    Store decision $\mathbf{a}[i] \leftarrow a_i$
end for
if *training mode* **then**
    // Training update using PPO
    Execute decisions $\mathbf{a}$ in environment
    Compute total delay $T_{\text{tot},u}$ and total energy $E_{\text{total},u}$ using ComputeMetrics$(\mathbf{a}, G_u)$
    Compute reward $r_u \leftarrow \lambda_t \cdot \frac{1}{1+T_{\text{tot},u}} + \lambda_e \cdot \frac{1}{1+E_{\text{total},u}}$
    Store experience $(\mathbf{s}_{u,t}, \mathbf{a}, r_u)$ in replay buffer
    if *replay buffer full* **then**
        Estimate advantage $\hat{A}_t \leftarrow \text{ComputeGAE(buffer)}$
        Update policy parameters θ using PPO objective with clipping parameter ϵ
    end if
end if
Output: $\mathbf{a}$: final offloading decisions for tasks $v_i \in V_u$

task dependencies via $\mathbf{h}_i$, RSU availability via $\mathbf{c}_r$ and $u_r(t)$, and mobility forecasts via $\mathbf{z}_u$, ensuring robust scheduling that aligns with the system model's objective of minimizing $\lambda_t \sum_u T_{\text{tot},u} + \lambda_e \sum_u E_{\text{total},u}$. The overall procedure of the proposed trajectory aware DAG offloading framework is summarized in Algorithm 4.1.

4.5 Experimental Setup

For the experimentation we utilize the US101 highway dataset from the Next Generation Simulation (NGSIM) program (of Transportation Federal Highway Administration, 2016), containing high-resolution vehicle trajectory data at 10 Hz sampling frequency over a 2000-meter highway segment. After preprocessing, the dataset yields 49 unique vehicle trajectories. The evaluation considers $|\mathcal{U}| = 5$ vehicles with sufficiently long trajectories and $|\mathcal{R}| = 10$ RSUs providing 200-meter coverage zones with 20% overlap. Each RSU i has compute capacity $4.0 + 0.4i$ GHz and bandwidth $100 + 10i$ Mbps, modelling heterogeneous infrastructure where RSUs are provisioned with varying hardware capabilities. The channel model incorporates free-space path loss with shadowing, co-channel interference, and SINR-based data rates following Shannon capacity. The core network connects the RSUs via 10 Gbps fiber backhaul links. The pre processing of the dataset includes: (i) filtering stationary vehicles (max speed < 5 m/s) and short trajectories (< 60 time steps) and (ii) applying 4-sigma clipping to remove outliers in speed, acceleration, and heading, and z-score normalization per feature. The mobility predictor uses a 50-step history window to predict 50 future (x, y) coordinates.

Task workloads are synthetically generated using the *DAGGEN* tool (Suter & Hunold, 2013), which produces realistic DAGs modeling three vehicular computing scenarios: *Light tasks* (8–15 nodes, 0.5–3.0 MB, 0.1–1.0 GCycles) representing mobile applications like object detection; *Medium tasks* (10–20 nodes, 1.0–5.0 MB, 0.3–2.0 GCycles) for IoT sensor processing; and *Heavy tasks* (15–25 nodes, 2.0–8.0 MB, 0.5–3.0 GCycles) for AR/VR applications. The complete dataset comprises 50,000 DAGs with realistic precedence structures: 15,000 light, 15,000 medium, and 20,000 heavy tasks.

We compare against four representative baseline strategies: (i) “all local" executes all tasks on the vehicle’s processor; (ii) “all remote" offloads all tasks to the first available RSU; (iii) “random" assigns each task uniformly between local and remote execution; and (iv) “compute-aware" uses a threshold-based heuristic that offloads tasks exceeding 1.0 GCycles to the best available RSU, while smaller tasks execute locally. The Transformer mobility predictor employs 4 encoder

blocks with 8 attention heads and 128-dimensional embeddings. Training uses the Adam optimizer with learning rate 1×10^{-4}, weight decay 1×10^{-5}, and early stopping based on validation MSE. The GAT-based PPO policy features 2 GAT layers with 4 attention heads each, processing task features, RSU states, and predicted trajectories. Key hyperparameters are summarized in Table 4.1.

Table 4.1 Training Hyperparameters

Parameter	**Transformer**	**PPO Policy**
Learning Rate	1×10^{-4}	3×10^{-4} (Actor) 1×10^{-3} (Critic)
Batch Size	1024	64 episodes
Hidden Dimension	128	128
Attention Heads	8	4
Layers/Blocks	4	2 (GAT)
Dropout	0.2	–
Weight Decay	1×10^{-5}	–
PPO Epochs	–	100
Clip Parameter	–	0.2
GAE Lambda	–	0.95
Discount Factor	–	0.99

4.6 Results and Discussion

We evaluated the proposed trajectory-aware DAG offloading framework through experiments on real vehicular trajectories and synthetic workloads. The experimental assessment focuses on three aspects: the accuracy of trajectory prediction, the performance of task offloading, and scalability.

Fig. 4.2 shows the step-wise deterioration of prediction accuracy over the 100-step prediction interval. The MAE gradually rises from 5.0 meters at the initial steps to 20.0 meters at the final step, following a quadratic growth trend typical of position prediction tasks. Meanwhile, the R^2 value decreases from 0.998 to 0.980, indicating that the model still maintains strong correlation with the actual positions even over extended prediction horizons.

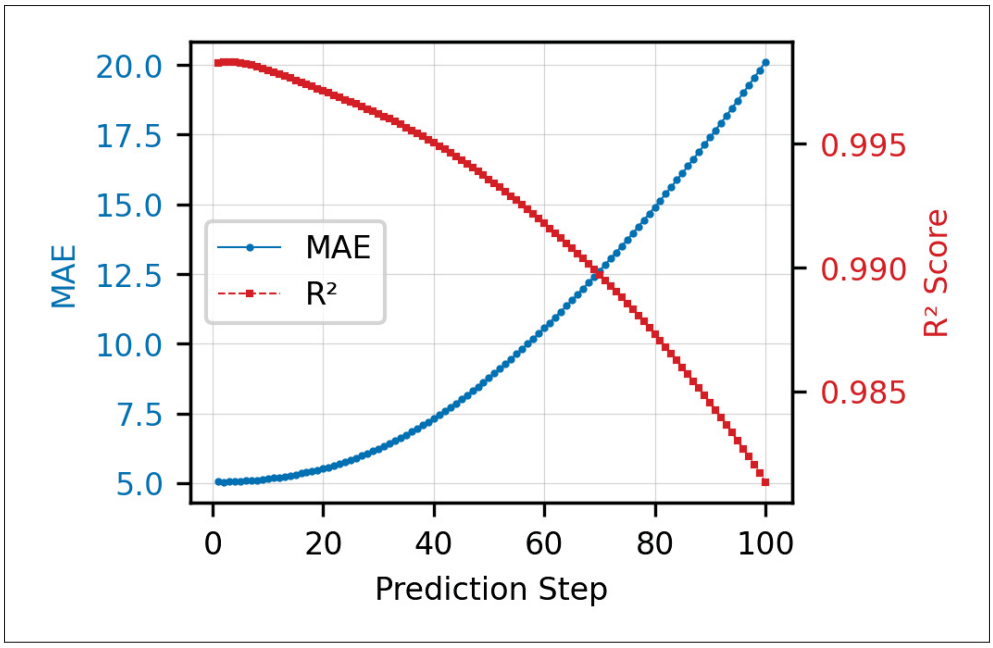

Figure 4.2 Step-wise MAE and R^2 over the prediction horizon

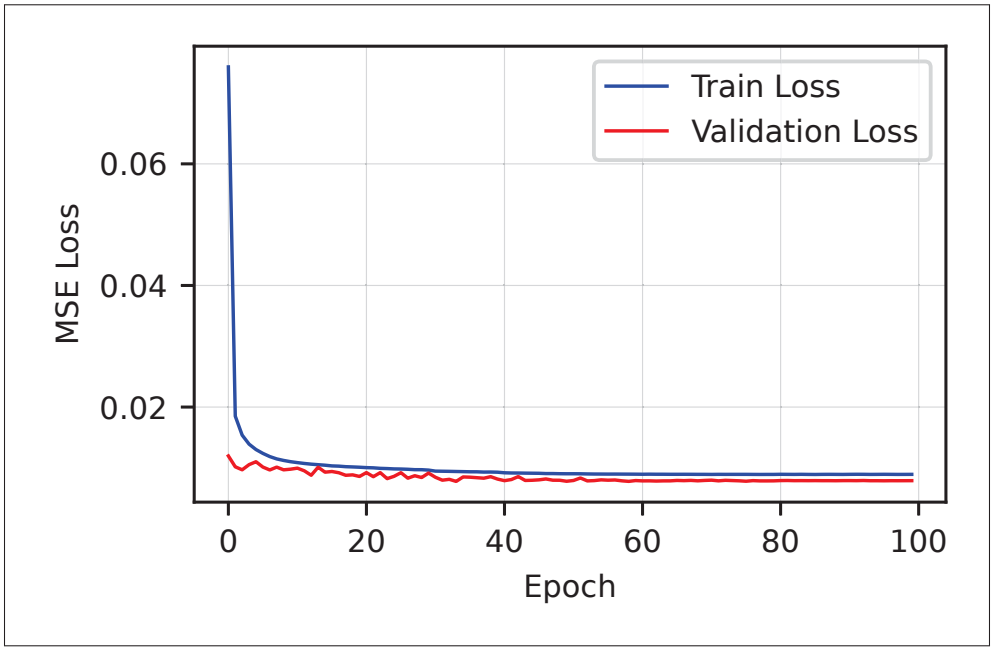

Figure 4.3 Mean Squared Error (MSE) loss progression during transformer training over 100 epochs

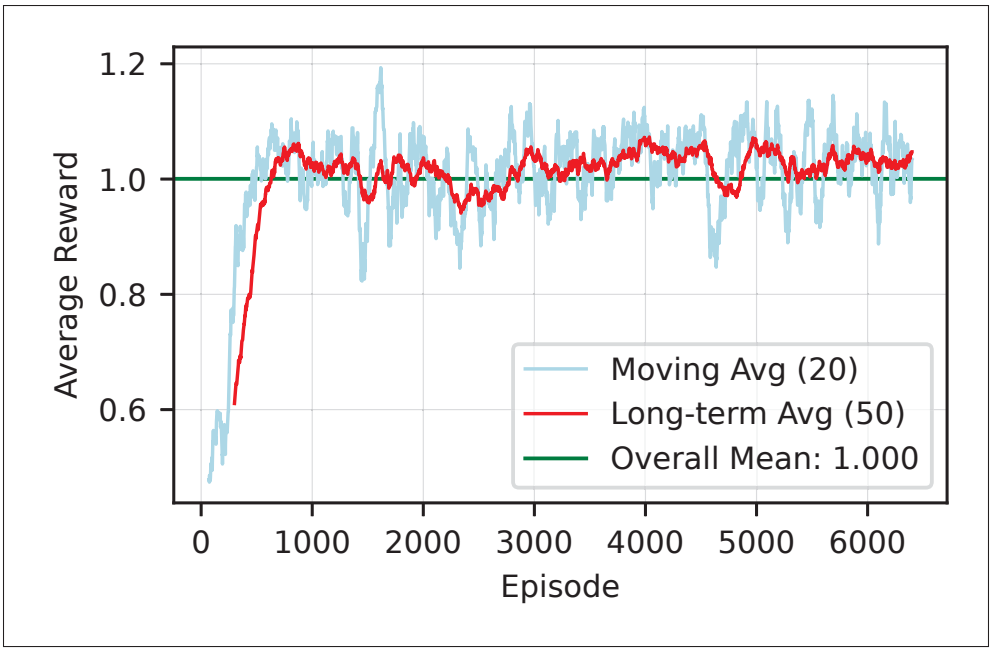

Figure 4.4 Training reward progression for the PPO-based offloading policy

Fig. 4.3 shows the Mean Squared Error (MSE) loss progression during transformer training over 100 epochs. Both training and validation losses decrease rapidly in the first 20 epochs, reaching approximately 0.02 MSE. The validation loss stabilizes around epoch 40, while training loss continues to decrease slightly, indicating appropriate model capacity without severe overfitting. The final training-validation gap remains minimal, suggesting good generalization to unseen trajectory data. The combination of these mechanisms enables the proposed framework to achieve superior performance across multiple evaluation criteria while maintaining computational efficiency suitable for real-time vehicular applications.

Fig. 4.4 illustrates the training process of the PPO-based policy over 6000 episodes. The training progresses through three distinct stages: an exploration phase (episodes 0-1000) with high variance in rewards, a convergence phase (episodes 1000-3000) where rewards rise from 0.8 to 1.1, and a stabilization phase (episodes 3000-6000) with rewards consistently around the average of 1.000. The moving averages (20-episode and 50-episode windows) indicate stable learning without oscillations or performance drops, reflecting robust policy convergence. Incorporating

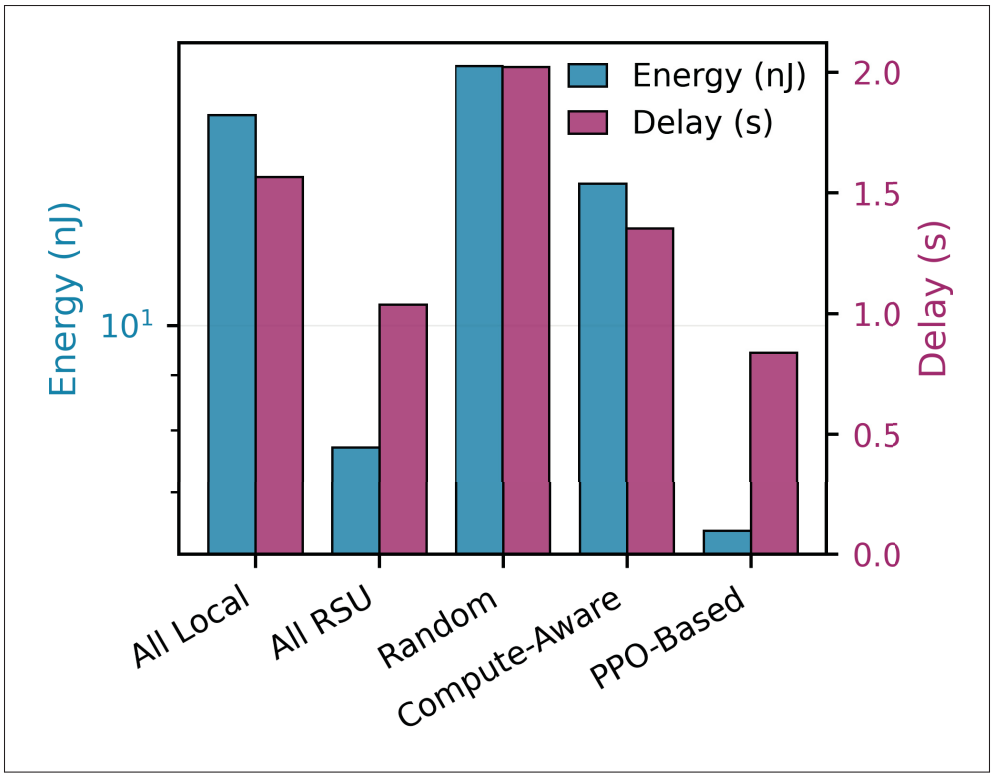

Figure 4.5 Average energy and delay comparison

trajectory prediction and action masking further enhances stability by structuring exploration and limiting the action space to feasible options.

Fig. 4.5 compares the performance of five offloading strategies in terms of average delay and energy consumption. The proposed PPO-based approach achieves a notably lower average makespan of 0.7 seconds, representing reductions of 70.8% and 58.8% compared to the all-local and compute-aware strategies, respectively. Regarding energy consumption, our method averages 4.8 nJ, corresponding to decreases of 80.3% and 71.4% relative to the all-local and compute-aware baselines. By contrast, the all-remote strategy consumes 7.6 nJ on average. These results highlight the superior energy efficiency of our approach, which smartly balances local execution with RSU offloading to reduce transmissions while taking advantage of RSU computing resources.

4.7 Conclusion

We introduced a trajectory-aware DAG-based task offloading framework for vVEC, combining a Transformer-based mobility predictor with a GAT-PPO scheduling agent. Extensive experiments using real vehicular trajectories and synthetic workloads showed that our approach consistently outperforms baseline methods in terms of delay and energy efficiency.

Future work will expand this framework to multi-agent scenarios and incorporate federated reinforcement learning, enabling collaborative coordination among RSUs and improving overall scalability.

CONCLUSION AND RECOMMENDATIONS

This chapter concludes the thesis by providing a comprehensive summary of the primary research contributions, theoretical advancements, and empirical findings presented in Section 5.1. It synthesizes the insights gained from the development of the EMDTORA, FEDORA, and Trajectory-Aware VEC frameworks. Subsequently, Section 5.2 outlines promising directions for future research, drawing upon emerging trends in AI-native 6G networks, Large Language Models (LLMs), Quantum Edge Intelligence, and Semantic Communication.

5.1 Conclusions

In this thesis, we explored, investigated, and proposed AI-driven frameworks for the intelligent orchestration of computation offloading in next generation edge networks. Mobile applications evolve from simple, atomic requests to complex workflows modeled as DAGs exemplified by augmented reality, real time video analytics, and cooperative autonomous driving. Traditional heuristic and static optimization methods fail to address the high dimensionality and dynamic nature of the environment. To bridge this gap, we leveraged advanced techniques including GNNs, DRL, and FL.

Specifically, we studied the joint optimization of computation offloading, resource allocation, and mobility management to minimize the weighted ETC, reduce end-to-end latency, and mitigate task failure rates under strict deadline constraints. The core contributions of this dissertation are summarized as follows:

In Chapter 2, we addressed the fundamental limitation of standard DRL agents e.g., Deep Q-Networks and DDPG in capturing the non-Euclidean topological dependencies between tasks. Conventional approaches typically flatten the task graph into a vector, thereby losing critical structural information regarding task precedence and potential parallelism. To overcome this, we proposed the EMDTORA framework. By integrating GAT within the actor-critic

architecture of the TD3 algorithm, we enabled the agent to explicitly learn the structural embeddings of application DAGs. The GAT-based encoder allows the agent to dynamically weigh the importance of neighbor tasks, effectively identifying the the sequence of dependent tasks that strictly bounds the application's completion time. Simulation results demonstrated that EMDTORA significantly outperforms standard DRL baselines and heuristic approaches in terms of ETC reduction. Specifically, the framework exhibited a superior ability to execute dependent tasks while maintaining low ETC even as the complexity of the DAG increased.

In Chapter 3, we tackled the challenges of data privacy, communication overhead, and statistical heterogeneity inherent in distributed edge networks. Recognizing that centralized training paradigms require uploading raw user data to a central server raising severe privacy concerns and consuming more uplink bandwidth. We introduced the FEDORA framework, FEDORA employs a decentralized training architecture where edge nodes train local models on their private data and share only model gradients with the global aggregator. To mitigate client drift caused by statistical heterogeneity, where different users execute distinct types of application DAGs, we utilized the FedProx aggregation algorithm. Furthermore, we designed a novel Ensemble DRL architecture that decouples the discrete action space (offloading decisions) from the continuous action space (resource allocation), training them via separate specialized heads. Our experimental analysis confirmed that FEDORA achieves robust convergence in non-IID environments where standard FedAvg tends to diverge. The ensemble approach allowed the global policy to generalize effectively across diverse user behaviors without requiring raw data exchange, significantly reducing the communication cost of training compared to centralized baselines.

In Chapter 4, we extended our analysis to the highly dynamic domain of VEC. We proposed a Trajectory-Aware VEC framework. We utilized Transformer-based sequence modeling to predict future vehicle trajectories with high precision over long time horizons. Unlike RNNs or

LSTMs, the Transformer's self-attention mechanism effectively captures long-range temporal dependencies in traffic data. These predictions were utilized to calculate the precise connectivity window between a vehicle and a RSU. The DRL agent was augmented with this predictive state, allowing it to proactively reject offloading requests if the estimated task completion time exceeded the connectivity window. This predictive capability ensures that tasks are offloaded only when reliable completion is guaranteed, consequently outperforming traditional methods in terms of delay and energy consumption.

5.2 Future Work

The research conducted in this thesis lays a robust foundation for intelligent edge orchestration. However, the rapid evolution toward 6G introduces new paradigms that extend beyond the capabilities of standard DRL and GNNs. Based on the literature review of emerging technologies several future research directions are identified to further advance the intelligence and resilience of edge computing systems.

5.2.1 Large Language Models (LLMs) for Intent-Based Networking

In this thesis, we relied on numerical DRL agents to optimize specific, pre-defined objective functions. However, recent surveys by Boateng et al. (2025) and Yang et al. (2025) highlight the emergence of Decision-Making LLMs in wireless networks (Boateng, Sami, Alagha, Elmekki & Guizani, 2025; Yang, Fan, Wang & Zhang, 2025). Current numerical agents lack the flexibility to understand high-level, abstract user goals. A promising extension of this work is the integration of LLMs to facilitate Intent-Based Networking (IBN) (Leivadeas & Falkner, 2022). An LLM could act as a high-level planner that interprets natural language user intents and translates them into specific reward functions or constraints for the low level DRL actor. This hierarchical approach could enhance the system's generalization to unseen application types and

user constraints that were not explicitly defined during the training phase (Hong, Tu & Hong, 2025).

5.2.2 Quantum-Enhanced Edge Intelligence for SAGIN

As the complexity of MEC systems grows with the integration of Space-Aerial-Ground Integrated Networks (SAGIN), the state space for DRL algorithms expands exponentially. (Gupta & Sultana, 2026) have proposed Quantum-Enhanced Edge Intelligence, leveraging quantum machine learning (QML) to accelerate training and inference. Classical DRL training becomes computationally prohibitive for massive-scale IoT networks involving thousands of concurrent dependent tasks. Future research could investigate the application of Quantum Reinforcement Learning (QRL) to solve the NP-hard dependent task offloading problem. Specifically, Variational Quantum Circuits (VQC) could replace the standard neural networks in the Actor-Critic architecture, potentially offering exponential speedups in processing high dimensional state spaces typical of 6G massive IoT environments.

5.2.3 Multi-Agent Coordination for Multi-RSU Vehicular Edge Computing

In Chapter 4, the Trajectory-Aware VEC framework considers a single-agent formulation where each vehicle interacts independently with one RSU. In realistic scenarios, vehicles traverse multiple RSU coverage zones, and adjacent RSUs compete for shared backhaul and computational resources. A natural extension is to reformulate the problem as a Multi-Agent Reinforcement Learning (MARL) setting, where each RSU operates as an autonomous agent coordinating with neighbors. Wang et al. Wang *et al.* (2024) proposed a federated GNN-enhanced MARL framework for VEC that combines graph-based inter-agent communication with privacy-preserving training. Similarly, Nguyen et al. Nguyen *et al.* (2025) developed a multi-agent DRL-based cooperative task offloading strategy combining V2V and V2I links with a task migration mechanism for handling RSU transitions. Building on our framework, each

RSU agent could leverage the Transformer-based trajectory predictions to anticipate incoming vehicles, while the FEDORA federated training paradigm would enable collaborative policy learning without exchanging raw vehicle data.

5.2.4 Generalization to Dynamic and Heterogeneous DAG Topologies

A shared assumption across all three contributions is that the DAG topology of each application is static and known a priori. In practice, modern applications such as adaptive video analytics exhibit dynamic DAG structures where subtasks may be spawned or pruned at runtime based on intermediate results. This directly impacts the GAT-based task embedding module in EMDTORA, which processes a fixed-topology graph at each decision step. Liu et al. Liu *et al.* (2023) proposed GA-DRL for scheduling DAG tasks over dynamic vehicular clouds, introducing non-uniform neighborhood sampling that enables generalization to unseen topologies. Shen et al. Shen *et al.* (2025) developed GATES, combining heterogeneous graph attention networks with evolution strategies for dynamic workflow scheduling using adaptive graph representations. Future work could extend the EMDTORA GAT encoder to operate on conditional DAGs, where edges and nodes are activated probabilistically based on predecessor outputs, enabling the DRL agent to reason about the expected remaining DAG structure when making offloading decisions.

5.2.5 Fast Adaptation via Meta-Reinforcement Learning

Each proposed framework is trained within a fixed environment configuration. When conditions change substantially, the agent requires complete retraining, which limits practical deployment across diverse edge environments. Meta-Reinforcement Learning (Meta-RL) addresses this by training agents that rapidly adapt to new environments with minimal fine-tuning. Wang et al. Wang, Hu, Min, Zomaya & Georgalas (2021b) demonstrated that meta-RL agents pre-trained over a distribution of MEC environments can adapt to unseen configurations with significantly fewer samples than training from scratch. Liu et al. Liu, Li, Jing, Duan & Mao (2025)

proposed ME-DRO, a meta-learning-based distributed RL algorithm for dynamic vehicular networks achieving rapid adaptation with improved latency and energy performance. A concrete direction is to train the EMDTORA agent using Model-Agnostic Meta-Learning (MAML) over diverse MEC environments, while FEDORA could benefit from personalized federated meta-learning where each client's initialization is adapted to its local distribution while still leveraging global knowledge.

ANNEXE A

AUTHOR'S PUBLICATIONS

During the course of his Ph.D. research, the author contributed to the following publications.

LIST OF REFERENCES

Khan, S., Ali-Pour, A., Avgeris, M., Gascon-Samson, J. & Leivadeas, A. (2025a). FEDORA: Federated Ensemble Reinforcement Learning for DAG-Based Task Offloading and Resource Allocation in MEC. *IEEE Internet of Things Journal*, 12(21), 44228–44242.

Khan, S., Avgeris, M., Gascon-Samson, J. & Leivadeas, A. (2025b). EMDTORA: Energy-Aware Multi-User Dependent Task Offloading and Resource Allocation in MEC Using Graph-Enabled DRL. *IEEE Transactions on Green Communications and Networking*, 9(3), 1453–1465.

Khan, S., Avgeris, M., Ali-Pour, A., Gascon-Samson, J. & Leivadeas, A. (2026). Dependent Task Offloading in Vehicular Edge Computing Using Trajectory-Aware Deep Reinforcement Learning. *IEEE International Conference on Communications (ICC)*.

Liu, L., Li, W., Jing, T., Duan, J. & Mao, W. (2025). Meta Learning-Based Deep Reinforcement Learning Algorithm for Task Offloading in Dynamic Vehicular Network. *Engineering Applications of Artificial Intelligence*, 143, 109929.

Liu, Z., Huang, L., Gao, Z., Luo, M., Hosseinalipour, S. & Dai, H. (2023). GA-DRL: Graph Neural Network-Augmented Deep Reinforcement Learning for DAG Task Scheduling over Dynamic Vehicular Clouds. *arXiv preprint arXiv:2307.00777*.

Nguyen, T. T. et al. (2025). Multi-Agent Deep Reinforcement Learning-Based Partial Offloading and Resource Allocation in Vehicular Edge Computing Networks. *Computer Communications*, 233, 108066.

Shen, Y. et al. (2025). GATES: Cost-Aware Dynamic Workflow Scheduling via Graph Attention Networks and Evolution Strategy. *arXiv preprint arXiv:2505.12355*.

Wang, J., Hu, J., Mills, J., Min, G., Xia, M. & Georgalas, N. (2021a). Federated Ensemble Model-Based Reinforcement Learning in Edge Computing. *arXiv preprint arXiv:2109.05549*.

Wang, J., Hu, J., Min, G., Zomaya, A. Y. & Georgalas, N. (2021b). Fast Adaptive Task Offloading in Edge Computing Based on Meta Reinforcement Learning. *IEEE Transactions on Parallel and Distributed Systems*, 32(1), 242–253.

Wang, J. et al. (2020). Federated Multi-Agent Actor-Critic Learning for Age Sensitive Mobile Edge Computing. *arXiv preprint arXiv:2012.14137*.

Wang, W., Wu, Q., Fan, P., Cheng, N., Chen, W., Wang, J. & Letaief, K. B. (2024). Optimizing Age of Information in Vehicular Edge Computing with Federated Graph Neural Network Multi-Agent Reinforcement Learning. *arXiv preprint arXiv:2407.02342*.

Yang, N., Wen, J., Zhang, M. & Tang, M. (2023). Multi-Objective Deep Reinforcement Learning for Mobile Edge Computing. *arXiv preprint arXiv:2307.14346*.

Yang, N., Wen, J., Zhang, M. & Tang, M. (2025). Generalizable Pareto-Optimal Offloading with Reinforcement Learning in Mobile Edge Computing. *arXiv preprint arXiv:2509.10474*.

BIBLIOGRAPHY

Adhikari, B., Khwaja, A. S., Jaseemuddin, M. & Anpalagan, A. (2024). Hybrid Network Slicing Technique for Co-existence of eMBB and URLLC Services in 6G-IoT Systems. *2024 IEEE 10th World Forum on Internet of Things (WF-IoT)*.

Ahmed, S. et al. (2025). Federated Reinforcement Learning-Based Dynamic Resource Allocation and Task Scheduling in Edge for IoT Applications. *MDPI Sensors*, 25(7), 2197.

Al-Molegi, A. et al. (2025). Transformer-Based Trajectory Prediction Using LiDAR Data for Situational Awareness in Complex Urban Environments. *IEEE Access*.

Al-Naday, M., Dobre, V., Reed, M., Toor, S., Volckaert, B. & De Turck, F. (2024). Federated deep Q-learning networks for service-based anomaly detection and classification in edge-to-cloud ecosystems. *Annals of Telecommunications*, 79(3), 165–178.

Ali, M. et al. (2025). Handover Decision with Multi-Access Edge Computing in 6G Networks: A Survey. *Results in Engineering*, 25(4), 103934.

Alotaibi, A. et al. (2024). Communication-Aware Consistent Edge Selection for Mobile Users and Autonomous Vehicles. *IEEE Access*.

An, X., Fan, R., Hu, H., Zhang, N., Atapattu, S. & Tsiftsis, T. A. (2022). Joint task offloading and resource allocation for IoT edge computing with sequential task dependency. *IEEE Internet Things J*, 9(17), 16546–16561.

Arabnejad, H. & Barbosa, J. G. (2014). List Scheduling Algorithm for Heterogeneous Systems by an Optimistic Cost Table. *IEEE Transactions on Parallel and Distributed Systems*, 25(3), 682–694. doi: 10.1109/TPDS.2013.57.

Bhatia, S., Mallikarjuna, B., Gautam, D. et al. (2023). The Future IoT: The Current Generation 5G and Next Generation 6G and 7G Technologies. *2023 International Conference on Device Intelligence, Computing and Communication Technologies (DICCT)*, pp. 212–217. doi: 10.1109/dicct56244.2023.10110066.

Boateng, G. O., Sami, H., Alagha, A., Elmekki, H. & Guizani, M. (2025). A Survey on Large Language Models for Communication, Network, and Service Management: Application Insights, Challenges, and Future Directions. *IEEE Communications Surveys & Tutorials*.

Cao, L., Huo, T., Li, S., Zhang, X., Chen, Y., Lin, G., Wu, F., Ling, Y., Zhou, Y. & Xie, Q. (2024a). Cost optimization in edge computing: a survey. *Artificial Intelligence Review*, 57(11), 312.

Cao, Z., Deng, X., Yue, S., Jiang, P., Ren, J. & Gui, J. (2024b). Dependent Task Offloading in Edge Computing Using GNN and Deep Reinforcement Learning. *IEEE Internet of Things Journal*, 11(12), 21632-21646.

Chen, J., Yang, Y., Wang, C., Zhang, H., Qiu, C. & Wang, X. (2021a). Multitask offloading strategy optimization based on directed acyclic graphs for edge computing. *IEEE Internet Things J.*, 9(12), 9367–9378.

Chen, J., Xing, H., Xiao, Z., Xu, L. & Tao, T. (2021b). A DRL agent for jointly optimizing computation offloading and resource allocation in MEC. *IEEE Internet Things J.*, 8(24), 17508–17524.

Chen, L. et al. (2024a). A Priority-Based Dynamic Rescheduling Offloading Strategy for Dependent Task Offloading in the Internet of Vehicles. *Journal of Computers*, 36(5).

Chen, L. et al. (2024b). Prioritized Task Offloading in Vehicular Edge Computing Using Deep Reinforcement Learning. *IEEE Conference Publication*.

Chen, N., Cheng, Z. et al. (2024c). Joint Dynamic Spectrum Allocation for URLLC and eMBB in 6G Networks. *IEEE Transactions on Network Science and Engineering*, 11(6).

Chen, X., Cao, J., Sahni, Y., Jiang, S. & Liang, Z. (2024d). Dynamic task offloading in edge computing based on dependency-aware reinforcement learning. *IEEE Transactions on Cloud Computing*, 12(2), 594–608.

Chen, X. & Liu, G. (2021). Energy-efficient task offloading and resource allocation via deep reinforcement learning for augmented reality in mobile edge networks. *IEEE Internet Things J.*, 8(13), 10843–10856.

Chen, X., Jiao, L., Li, W. & Fu, X. (2016). Efficient Multi-User Computation Offloading for Mobile-Edge Cloud Computing. *IEEE/ACM Transactions on Networking*, 24(5), 2795–2808. doi: 10.1109/TNET.2015.2487344.

Chen, Y. et al. (2024e). Federated Learning With Experience-Driven Model Migration in Heterogeneous Edge Networks. *IEEE Journals & Magazine*.

Chen, Y., Gu, W. & Li, K. (2022). Dynamic task offloading for internet of things in mobile edge computing via deep reinforcement learning. *International Journal of Communication Systems*, e5154.

Chen, Z. & Wang, X. (2020). Decentralized computation offloading for multi-user mobile edge computing: a deep reinforcement learning approach. *EURASIP Journal on Wireless Communications and Networking*, 2020(1), 188. doi: 10.1186/s13638-020-01801-6.

da Costa, J. B. D. et al. (2023). Mobility and Deadline-Aware Task Scheduling Mechanism for Vehicular Edge Computing. *IEEE Transactions on Intelligent Transportation Systems*, 24(10), 11345–11357.

Dai, F., Liu, G., Mo, Q., Xu, W. & Huang, B. (2022a). Task offloading for vehicular edge computing with edge-cloud cooperation. *World Wide Web*, 25(5), 1999–2017. doi: 10.1007/S11280-022-01011-8.

Dai, F., Liu, G., Mo, Q., Xu, W. & Huang, B. (2022b). Task offloading for vehicular edge computing with edge-cloud cooperation. *World Wide Web*, 25(5), 1999–2017. doi: 10.1007/S11280-022-01011-8.

Dai, Y., Lyu, L., Cheng, N. et al. (2024). A Survey of Graph-Based Resource Management in Wireless Networks - Part II: Learning Approaches. *IEEE Transactions on Cognitive Communications and Networking*, 1–1. doi: 10.1109/tccn.2024.3508777.

de Queiroz, G. F. C., de Rezende, J. F. & Barbosa, V. C. (2024). A flexible algorithm to offload DAG applications for edge computing. *Journal of Network and Computer Applications*. doi: 10.1016/j.jnca.2024.103789. Early version available as arXiv:2306.09458.

Dong, S., Tang, J., Abbas, K., Hou, R., Kamruzzaman, J., Rutkowski, L. & Buyya, R. (2024). Task offloading strategies for mobile edge computing: A survey. *Computer Networks*, 254, 110791.

Dustdar, S. et al. (2024). A Bilateral Game Approach for Task Outsourcing in Multi-Access Edge Computing. *IEEE Transactions on Network and Service Management*, 21(1).

Elkawkagy, M. et al. (2025). Genetic Algorithm-Driven Joint Optimization of Task Offloading and Resource Allocation for Fairness-Aware Latency Minimization in Mobile Edge Computing. *IEEE Access*.

Fan, W. et al. (2022). DNN deployment, task offloading, and resource allocation for joint task inference in IIoT. *IEEE Trans. Ind. Inf.*, 19(2), 1634–1646.

Fan, Y., Ge, J., Zhang, S., Wu, J. & Luo, B. (2023). Decentralized scheduling for concurrent tasks in mobile edge computing via deep reinforcement learning. *IEEE Transactions on Mobile Computing*, 23(4), 2765–2779.

Feng, C., Han, P., Zhang, X., Zhang, Q., Liu, Y. & Guo, L. (2024). Dependency-aware task reconfiguration and offloading in multi-access edge cloud networks. *IEEE Trans. Mob. Comput.*, 23(10), 9271–9288.

Feriani, A. & Hossain, E. (2021). Single and multi-agent deep reinforcement learning for AI-enabled wireless networks: A tutorial. *IEEE Commun. Surv. Tutorials*, 23(2), 1226–1252.

François-Lavet, V., Henderson, P., Islam, R., Bellemare, M. G. & Pineau, J. (2018). An Introduction to Deep Reinforcement Learning. *Foundations and Trends® in Machine Learning*, 11(3-4), 219–354.

Fujimoto, S., Hoof, H. & Meger, D. (2018a). Addressing Function Approximation Error in Actor-Critic Methods. *International Conference on Machine Learning*, pp. 1587–1596.

Fujimoto, S., Hoof, H. & Meger, D. (2018b). Addressing function approximation error in actor-critic methods. *Int. Conf. on ML*, pp. 1587–1596.

Gholipour, N., De Assuncao, M. D., Agarwal, P., Gascon-Samson, J. & Buyya, R. (2023). TPTO: A Transformer-PPO based Task Offloading Solution for Edge Computing Environments. *2023 IEEE 29th International Conference on Parallel and Distributed Systems (ICPADS)*, pp. 1115–1122.

Gu, H., Zhao, L., Han, Z., Zheng, G. & Song, S. (2023). AI-enhanced cloud-edge-terminal collaborative network: Survey, applications, and future directions. *IEEE Communications Surveys & Tutorials*, 26(2), 1322–1385.

Guo, P. et al. (2025). A New Federated Learning Framework Against Gradient Inversion Attacks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 39(16), 16969–16977.

Guo, S., Liu, J., Yang, Y., Xiao, B. & Li, Z. (2018). Energy-efficient dynamic computation offloading and cooperative task scheduling in mobile cloud computing. *IEEE Trans. Mob. Comput.*, 18(2), 319–333.

Gupta, A. & Sultana, A. (2026). Quantum-Enhanced Edge Intelligence Leveraging Large Language Models for Immersive Space–Aerial–Ground Communications: Survey, Challenges, and Open Issues. *Sensors*, 26(4), 1181.

Haarnoja, T., Zhou, A., Abbeel, P. & Levine, S. (2018). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *International conference on machine learning*, pp. 1861–1870.

Hasan, M. et al. (2025). A Federated Learning Approach for Predicting Resource Allocation in Multi-Access Edge Computing (MEC). *International Journal of Computer Trends and Technology*, 73(5).

He, Z., Zhang, T. & Lee, R. B. (2020). Attacking and protecting data privacy in edge–cloud collaborative inference systems. *IEEE Internet of Things Journal*, 8(12), 9706–9716.

Hong, J., Tu, N. V. & Hong, J. W.-k. (2025). A Comprehensive Survey on LLM-Based Network Management and Operations. *International Journal of Network Management*, 35(6), e70029.

Hortelano, D., de Miguel, I., Barroso, R. J. D., Aguado, J. C., Merayo, N., Ruiz, L., Asensio, A., Masip-Bruin, X., Fernández, P., Lorenzo, R. M. & Abril, E. J. (2023). A comprehensive survey on reinforcement-learning-based computation offloading techniques in Edge Computing Systems. *Journal of Network and Computer Applications*, 216, 103669. doi: 10.1016/j.jnca.2023.103669.

Huang, Y. (2025). Multimedia Tasks-Oriented Edge Computing Offloading Scheme Based on Graph Neural Network in Vehicular Networks. *IEEE Access*, 13, 9780–9791. doi: 10.1109/access.2025.3526627.

Huynh, L. N. T., Pham, Q.-V., Nguyen, T. D. T. et al. (2021). Joint Computational Offloading and Data-Content Caching in NOMA-MEC Networks. *IEEE Access*, 9, 12943–12954. doi: 10.1109/access.2021.3051278.

Jia, X. et al. (2025). Federated Multi-Agent Reinforcement Learning for AoI Minimization in UAV-Enabled IoV-MEC Systems. *IEEE Conference Publication*.

Jiang, H., Dai, X., Xiao, Z. & Iyengar, A. (2022). Joint task offloading and resource allocation for energy-constrained mobile edge computing. *IEEE Trans. Mob. Comput.*, 22(7), 4000–4015.

Kang, H., Li, M., Lin, L., Fan, S. & Cai, W. (2024). Bridging incentives and dependencies: An iterative combinatorial auction approach to dependency-aware offloading in mobile edge computing. *IEEE Transactions on Mobile Computing*, 23(12), 12113–12130.

Karimireddy, S. P., Kale, S., Mohri, M., Reddi, S., Stich, S. & Suresh, A. T. (2020). Scaffold: Stochastic controlled averaging for federated learning. *International conference on machine learning*, pp. 5132–5143.

Khan, S., Avgeris, M., Gascon-Samson, J. & Leivadeas, A. (2024). EMDTORA: Energy-aware multi-user dependent task offloading and resource allocation in MEC using graph-enabled DRL. *IEEE Trans. Green Commun. Networking*.

Khan, S., Ali-Pour, A., Avgeris, M., Gascon-Samson, J. & Leivadeas, A. (2025a). FEDORA: Federated Ensemble Reinforcement Learning for DAG-Based Task Offloading and Resource Allocation in MEC. *IEEE Internet of Things Journal*, 12(21), 44228–44242.

Khan, S., Avgeris, M., Ali-Pour, A., Gascon-Samson, J. & Leivadeas, A. (2025b). Dependent Task Offloading in Vehicular Edge Computing Using Trajectory-Aware Deep Reinforcement Learning. *IEEE International Conference on Communications (ICC)*.

Khan, S., Avgeris, M., Gascon-Samson, J. & Leivadeas, A. (2025c). EMDTORA: Energy-Aware Multi-User Dependent Task Offloading and Resource Allocation in MEC Using Graph-Enabled DRL. *IEEE Transactions on Green Communications and Networking*, 9(3), 1453–1465.

Kim, J. et al. (2024). A Federated Reinforcement Learning Framework via a Committee Mechanism for Resource Management in 5G Networks. *PMC*.

Kim, J. et al. (2025). Transformer-Based Vehicle-Trajectory Prediction at Urban Low-Speed T-Intersection. *Sensors (MDPI)*, 25(14), 4256.

Kipf, T. N. & Welling, M. (2016). Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

Konečnỳ, J. et al. (2016). Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*.

Kwok, Y.-K. & Ahmad, I. (1996). Dynamic Critical-Path Scheduling: An Effective Technique for Allocating Task Graphs to Multiprocessors. *IEEE Transactions on Parallel and Distributed Systems*, 7(5), 506–521. doi: 10.1109/71.506548.

Lee, J., Ko, H., Kim, J. & Pack, S. (2020). Data: Dependency-aware task allocation scheme in distributed edge clouds. *IEEE Transactions on Industrial Informatics*, 16(12), 7782–7790.

Leivadeas, A. & Falkner, M. (2022). A survey on intent-based networking. *IEEE Communications Surveys & Tutorials*, 25(1), 625–655.

Li, J. et al. (2024a). Dependent Task Graph Offloading Model Based on Deep Reinforcement Learning in Mobile Edge Computing. *Electronics (MDPI)*, 14(16), 3184.

Li, J. et al. (2024b). A DRL-based algorithm for Dependent Task Offloading in Multi-access Edge Computing. *IEEE Conference Publication*.

Li, J., Gu, B., Qin, Z. & Han, Y. (2023). Graph Tasks Offloading and Resource Allocation in Multi-Access Edge Computing: A DRL-and-Optimization-Aided Approach. *IEEE Trans. Network Sci. Eng.*

Li, Q., Wen, M., Dang, S., Basar, E., Poor, H. V. & Chen, F. (2019). Opportunistic spectrum sharing based on OFDM with index modulation. *IEEE Trans. Wireless Commun.*, 19(1), 192–204.

Li, T. et al. (2025). Multi-Agent Deep Reinforcement Learning With Trajectory Prediction for Task Migration-Assisted Computation Offloading. *IEEE Transactions on Mobile Computing*.

Li, T., Sahu, A. K., Zaheer, M., Sanjabi, M., Talwalkar, A. & Smith, V. (2020). Federated optimization in heterogeneous networks. *Proceedings of Machine learning and systems*, 2, 429–450.

Li, X. et al. (2024c). Evolutionary Algorithms for Edge Server Placement in Vehicular Edge Computing. *IEEE Access*.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D. & Wierstra, D. (2015a). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

Lillicrap, T. P. et al. (2015b). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

Lim, W. Y. B., Luong, N. C., Hoang, D. T., Jiao, Y., Liang, Y.-C., Yang, Q., Niyato, D. & Miao, C. (2020). Federated learning in mobile edge networks: A comprehensive survey. *IEEE communications surveys & tutorials*, 22(3), 2031–2063.

Lin, J., Cai, J., Zhang, H., Zhang, R., Yang, X. & Zhao, P. A Novel Deep Reinforcement Learning Based Dependent Tasks Offloading in Multi-Access Edge Computing. *Available at SSRN 4196751*.

Lin, T., Lin, C.-K., Chen, Z. & Cheng, H. (2022). Computation Offloading Algorithm Based on Deep Reinforcement Learning and Multi-Task Dependency for Edge Computing. *International Computer Symposium*, pp. 111–122.

Liu, J., Ren, J., Zhang, Y., Peng, X., Zhang, Y. & Yang, Y. (2021a). Efficient dependent task offloading for multiple applications in MEC-cloud system. *IEEE Trans. Mob. Comput.*, 22(4), 2147–2162.

Liu, J., Ren, J., Zhang, Y., Peng, X., Zhang, Y. & Yang, Y. (2023a). Efficient Dependent Task Offloading for Multiple Applications in MEC-Cloud System. *IEEE Trans. Mob. Comput.*, 22(4), 2147-2162. doi: 10.1109/TMC.2021.3119200.

Liu, J., Mao, X., Fang, Y., Zhu, D. & Meng, M. Q.-H. (2021b). A Survey on Deep-Learning Approaches for Vehicle Trajectory Prediction in Autonomous Driving. *arXiv preprint arXiv:2110.10436*.

Liu, L., Chen, C., Pei, Q., Maharjan, S. & Zhang, Y. (2021c). Vehicular edge computing and networking: A survey. *Mobile Networks and Applications*, 26(3), 1145–1168.

Liu, Q., Tian, Z., Wang, N. & Lin, Y. (2024a). DRL-based dependent task offloading with delay-energy tradeoff in medical image edge computing. *Complex & Intelligent Systems*, 10(3), 3283–3304.

Liu, S. et al. (2024b). DAG-Based Dependent Tasks Offloading in MEC-Enabled IoT With Soft Cooperation. *IEEE Transactions on Mobile Computing*.

Liu, S. et al. (2024c). Latency-Constrained Multi-User Efficient Task Scheduling in Large-Scale Internet of Vehicles. *IEEE Transactions on Mobile Computing*.

Liu, S. et al. (2025). Task Offloading Optimization in Vehicular Edge Computing with Meta-Policy Models. *IEEE Conference Publication*.

Liu, S. et al. (2023b). Dependent task scheduling and offloading for minimizing deadline violation ratio in mobile edge computing networks. *IEEE J. Sel. Areas Commun.*, 41(2), 538–554.

Liu, Y. et al. (2020). Dependency-aware task scheduling in vehicular edge computing. *IEEE Internet Things J.*, 7(6), 4961–4971.

Liu, Z., Liwang, M., Hosseinalipour, S., Dai, H., Gao, Z. & Huang, L. (2023c). RFID: Towards low latency and reliable DAG task scheduling over dynamic vehicular clouds. *IEEE Trans. Veh. Technol.*, 72(9), 12139–12153.

Lu, Z., Pan, H., Dai, Y., Si, X. & Zhang, Y. (2024). Federated Learning With Non-IID Data: A Survey. *IEEE Internet of Things Journal*, 11(12), 19188–19209. doi: 10.1109/jiot.2024.3376548.

Lv, S., Liu, W., Chen, D. & Qin, Z. (2021). Task Offloading and Serving Handover of Vehicular Edge Computing Networks Based on Trajectory Prediction. *IEEE Access*, 9, 130793–130804. doi: 10.1109/ACCESS.2021.3112077.

Mach, P. & Becvar, Z. (2017). Mobile edge computing: A survey on architecture and computation offloading. *IEEE Commun. Surv. Tutorials*, 19(3), 1628–1656.

Mahmoodi, S. E., Uma, R. & Subbalakshmi, K. (2016). Optimal joint scheduling and cloud offloading for mobile applications. *IEEE Trans. Cloud Comput.*, 7(2), 301–313.

Mao, Y., Zhang, J. & Letaief, K. B. (2016). Dynamic Computation Offloading for Mobile-Edge Computing with Energy Harvesting Devices. *IEEE Journal on Selected Areas in Communications*, 34(12), 3590–3605. doi: 10.1109/JSAC.2016.2611964.

Mao, Y., You, C., Zhang, J., Huang, K. & Letaief, K. B. (2017a). A survey on mobile edge computing: The communication perspective. *EEE Commun. Surv. Tutorials*, 19(4), 2322–2358.

Mao, Y., You, C., Zhang, J., Huang, K. & Letaief, K. B. (2017b). A Survey on Mobile Edge Computing: The Communication Perspective. *IEEE Communications Surveys & Tutorials*, 19(4), 2322–2358. doi: 10.1109/COMST.2017.2745201.

Marwani, M. & Kaddoum, G. (2024). Graph Neural Networks Approach for Joint Wireless Power Control and Spectrum Allocation. *IEEE Transactions on Machine Learning in Communications and Networking*, 2, 717–732.

McMahan, B., Moore, E., Ramage, D., Hampson, S. & y Arcas, B. A. (2017a). Communication-efficient learning of deep networks from decentralized data. *Artificial intelligence and statistics*, pp. 1273–1282.

McMahan, B. et al. (2017b). Communication-Efficient Learning of Deep Networks from Decentralized Data. *Artificial Intelligence and Statistics*, pp. 1273–1282.

Mnih, V. et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540), 529–533.

Moon, S.-H. & Lim, Y. (2022). Federated Deep Reinforcement Learning Based Task Offloading with Power Control in Vehicular Edge Computing. *Sensors*, 22(24), 9595. doi: 10.3390/s22249595.

Naval, P. & Pushparajesh, V. (2025). 6G Mobile Network Latency Mobile Edge Computing Cloud Architecture. *2025 First International Conference on Advances in Computer Science, Electrical, Electronics, and Communication Technologies (CE2CT)*.

Nieto, G., de la Iglesia, I., López-Novoa, U. & Perfecto, C. (2023). Deep Reinforcement Learning-based Task Offloading in MEC for energy and resource-constrained devices. *2023 IEEE Int. Mediterranean Conf. Commun. Netw. (MeditCom)*, pp. 127–132.

Nikhil, N. & Morris, B. T. (2018). Convolutional Neural Network for Trajectory Prediction. *arXiv preprint arXiv:1809.00696*.

Ning, Z., Zhang, K., Wang, X., Guo, L., Hu, X., Huang, J., Hu, B. & Kwok, R. Y. (2020). Intelligent edge computing in internet of vehicles: A joint computation offloading and caching solution. *IEEE Transactions on Intelligent Transportation Systems*, 22(4), 2212–2225.

of Transportation Federal Highway Administration, U. D. (2016). Next Generation Simulation (NGSIM) Program US-101 Videos. ITS DataHub through Data.transportation.gov.

Oikonomou, E., Plastras, S., Tsoumatidis, D., Skoutas, D. N. & Rouskas, A. (2024). Workload Prediction for Efficient Node Management in Mobile Edge Computing. *2024 IFIP Networking Conference (IFIP Networking)*.

Ouyang, B., Li, J. & Chen, X. (2023). DDPG-FL: A Reinforcement Learning Approach for Data Balancing in Federated Learning. *China Conference on Networking*, pp. 33–47.

Park, J. et al. (2025). Task Offloading and Scheduling Under Hard Deadlines in Vehicular Edge Computing Systems. *IEEE Transactions on Vehicular Technology*, 74(6), 10029–10034.

Park, S. H., Kim, B., Kang, C. M., Chung, C. C. & Choi, J. W. (2018). Sequence-to-Sequence Prediction of Vehicle Trajectory via LSTM Encoder-Decoder Architecture. *2018 IEEE Intelligent Vehicles Symposium (IV)*, pp. 1672–1678. doi: 10.1109/IVS.2018.8500658.

Peng, X., Zhang, Y., Zhang, X., Zhang, C. & Yang, W. (2024). Collaborative optimization strategy for dependent task offloading in vehicular edge computing. *Mathematics*, 12(23), 3820.

Pistilli, F. & Averta, G. (2023). Graph learning in robotics: a survey. *IEEE Access*, 11, 112664–112681.

Ponzi, V. & Napoli, C. (2025). Graph Neural Networks: Architectures, Applications, and Future Directions. *IEEE Access*, 13, 62870–62891.

Qadeer, A. & Lee, M. J. (2023). Deep-Deterministic Policy Gradient Based Multi-Resource Allocation in Edge-Cloud System: A Distributed Approach. *IEEE Access*, 11, 20381–20398.

Qian, Y., Xu, J., Zhu, S., Xu, W., Fan, L. & Karagiannidis, G. K. (2022). Learning to optimize resource assignment for task offloading in mobile edge computing. *IEEE Communications Letters*, 26(6), 1303–1307.

Qu, G., Wu, H., Li, R. & Jiao, P. (2021). DMRO: A deep meta reinforcement learning-based task offloading framework for edge-cloud computing. *IEEE Trans. Netw. Serv. Manage.*, 18(3), 3448–3459.

Sabella, D. et al. (2019). Developing software for multi-access edge computing. *ETSI white paper*, 20, 1–38.

Saeik, F. et al. (2021a). Task offloading in Edge and Cloud Computing: A survey on mathematical, artificial intelligence and control theory solutions. *Computer Networks*, 195, 108177. doi: https://doi.org/10.1016/j.comnet.2021.108177.

Saeik, F. et al. (2021b). Task offloading in Edge and Cloud Computing: A survey on mathematical, artificial intelligence and control theory solutions. *Computer Networks*, 195, 108177.

Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M. & Monfardini, G. (2008). The graph neural network model. *IEEE transactions on neural networks*, 20(1), 61–80.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A. & Klimov, O. (2017a). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A. & Klimov, O. (2017b). Proximal Policy Optimization Algorithms. *arXiv preprint arXiv:1707.06347*.

Shao, L. W., Qian, L. P., Li, M. Q., Jiang, W. & Jia, W. (2025). SC-DRL: A Status Correction-empowered Deep Reinforcement Learning Algorithm for Dependency-aware Application Offloading. *IEEE Transactions on Services Computing*.

Shao, Y., Li, R., Hu, B., Wu, Y., Zhao, Z. & Zhang, H. (2021). Graph attention network-based multi-agent reinforcement learning for slicing resource management in dense cellular network. *IEEE Trans. Veh. Technol*, 70(10).

Shen, Q., Hu, B.-J. & Xia, E. (2022). Dependency-Aware Task Offloading and Service Caching in Vehicular Edge Computing. *IEEE Transactions on Vehicular Technology*, 71(12), 13182-13197. doi: 10.1109/TVT.2022.3196544.

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

Shu, C., Zhao, Z., Han, Y., Min, G. & Duan, H. (2019). Multi-user offloading for edge computing networks: A dependency-aware and latency-optimal approach. *IEEE Internet Things J.*, 7(3), 1678–1689.

Sun, B., Theile, M., Qin, Z. et al. (2024). Edge Generation Scheduling for DAG Tasks Using Deep Reinforcement Learning. *IEEE Transactions on Computers*, 73(4), 1024–1037. doi: 10.1109/TC.2024.3350243.

Sundar, S. & Liang, B. (2018). Offloading dependent tasks with communication delay and deadline constraint. *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, pp. 37–45.

Suter, F. & Hunold, S. (2013). Daggen: A synthetic task graph generator. Github.

Sutton, R. S. & Barto, A. G. (2018a). *Reinforcement Learning: An Introduction*. MIT press.

Sutton, R. S. & Barto, A. G. (2018b). *Reinforcement Learning: An Introduction*. MIT press.

Taleb, T., Samdanis, K., Mada, B., Flinck, H., Dutta, S. & Sabella, D. (2017). On multi-access edge computing: A survey of the emerging 5G network edge cloud architecture and orchestration. *IEEE Communications Surveys & Tutorials*, 19(3), 1657–1681.

Tang, M. & Wong, V. W. (2020). Deep reinforcement learning for task offloading in mobile edge computing systems. *IEEE Trans. Mob. Comput.*, 21(6), 1985–1997.

Tariq, M. et al. (2024). Vehicular Communication Network Enabled CAV Data Offloading: A Review. *IEEE Access*.

Tong, H., Chen, C. & Zhu, J. (2025). Adaptive Edge Task Offloading via Parameterized Multi-Objective Reinforcement Learning with Hybrid Action Space. *IEEE Transactions on Network Science and Engineering*.

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

Van Hasselt, H., Guez, A. & Silver, D. (2016). Deep reinforcement learning with double q-learning. *Proceedings of the AAAI conference on artificial intelligence*, 30(1).

Vaswani, A. et al. (2017). Attention is all you need. *Advances in Neural Information Processing Systems*, pp. 5998–6008.

Veličković,Petar, e. a. (2017). Graph attention networks. *arXiv preprint arXiv:1710.10903*.

Wang, B., Wang, C., Huang, W., Song, Y. & Qin, X. (2020a). A survey and taxonomy on task offloading for edge-cloud computing. *IEEE Access*, 8, 186080–186101.

Wang, H. et al. (2024a). Dependent Task Offloading in Edge Computing Using GNN and Deep Reinforcement Learning. *IEEE Internet of Things Journal*, 11(12), 21632–21646.

Wang, H. et al. (2024b). Dependent Task Offloading in Edge Computing Using GNN and Deep Reinforcement Learning. *IEEE Internet of Things Journal*, 11(12), 21632–21646.

Wang, J., Zhao, L., Liu, J. & Kato, N. (2019a). Smart resource allocation for mobile edge computing: A deep reinforcement learning approach. *IEEE Trans. Emerging Top. Comput.*, 9(3), 1529–1541.

Wang, J., Liu, Q., Liang, H., Joshi, G. & Poor, H. V. (2020b). Tackling the objective inconsistency problem in heterogeneous federated optimization. *Advances in neural information processing systems*, 33, 7611–7623.

Wang, J., Hu, J., Min, G., Zhan, W., Zomaya, A. Y. & Georgalas, N. (2021a). Dependent task offloading for edge computing based on deep reinforcement learning. *IEEE Transactions on Computers*, 71(10), 2449–2461.

Wang, S., Tuor, T., Salonidis, T., Ko, K. K., Leung, K. K., Fallah, C. & Joshi, G. (2019b). Adaptive federated learning in resource constrained edge computing systems. *IEEE Journal on Selected Areas in Communications*, 37(6), 1205–1221.

Wang, S., Zhang, X., Zhang, Y., Wang, L., Yang, J. & Wang, W. (2017). A Survey on Mobile Edge Networks: Convergence of Computing, Caching and Communications. *IEEE Access*, 5, 6757–6779. doi: 10.1109/ACCESS.2017.2685434.

Wang, X., Wang, C., Li, X., Leung, V. C. & Taleb, T. (2020c). Federated deep reinforcement learning for Internet of Things with decentralized cooperative edge caching. *IEEE Internet Things J.*, 7(10), 9441–9455.

Wang, Y., Fang, W., Ding, Y. & Xiong, N. (2021b). Computation offloading optimization for UAV-assisted mobile edge computing: a deep deterministic policy gradient approach. *Wireless Networks*, 27(4), 2991–3006.

Wang, Z., Schaul, T., Hessel, M., Hasselt, H., Lanctot, M. & Freitas, N. (2016). Dueling Network Architectures for Deep Reinforcement Learning. *International Conference on Machine Learning*, pp. 1995–2003.

Wu, H., Gu, A. & Liang, Y. (2024a). Federated reinforcement learning-empowered task offloading for large models in vehicular edge computing. *IEEE Transactions on Vehicular Technology*, 74(2), 1979–1991.

Wu, Q. et al. (2024b). Graph Convolutional Network Augmented Deep Reinforcement Learning for Dependent Task Offloading in Mobile Edge Computing. *IEEE Conference Publication*.

Wu, Y. et al. (2025). Task offloading delay minimization in vehicular edge computing based on vehicle trajectory prediction. *Digital Communications and Networks*, 11(2), 537–546.

Wu, Y.-C., Dinh, T. Q., Fu, Y., Lin, C. & Quek, T. Q. (2021). A hybrid DQN and optimization approach for strategy and resource allocation in MEC networks. *IEEE Trans. Wireless Commun.*, 20(7), 4282–4295.

Wu, Z., Jia, Z., Pang, X. & Zhao, S. (2024c). Deep reinforcement learning-based task offloading and load balancing for vehicular edge computing. *Electronics*, 13(8), 1511.

Xiao, H., Xu, C., Ma, Y., Yang, S., Zhong, L. & Muntean, G.-M. (2022). Edge Intelligence: A Computational Task Offloading Scheme for Dependent IoT Application. *IEEE Trans. on Wireless Communications*, 21(9), 7222-7237. doi: 10.1109/TWC.2022.3156905.

Xiao, H., Hu, Z., Zhang, X., Xu, A., Zheng, M. & Li, K. (2024a). Federated deep reinforcement learning for task offloading in MEC-enabled heterogeneous networks. *IEEE Internet of Things Journal*, 12(8), 10238–10252.

Xiao, H., Hu, Z., Zhang, X., Xu, A., Zheng, M. & Li, K. (2024b). Federated deep reinforcement learning for task offloading in MEC-enabled heterogeneous networks. *IEEE Internet of Things Journal*, 12(8), 10238–10252.

Xiao, L. et al. (2024c). Edge Intelligence Empowered Social Image Recognition using Microservices Architecture. *2024 IEEE International Conference on Social Computing and Networking (SocialCom)*.

Xie, B., Cui, H., He, Y. & Guizani, M. (2024). A deep reinforcement learning approach for dependent task offloading in multi-access edge computing. *IEEE Transactions on Cognitive Communications and Networking*, 11(4), 2601–2617.

Xu, C. et al. (2023). Energy consumption and time-delay optimization of dependency-aware tasks offloading for industry 5.0 applications. *IEEE Trans. Consum. Electron.*, 70(1), 1590–1600.

Yan, J., Bi, S., Zhang, Y. J. & Tao, M. (2019). Optimal task offloading and resource allocation in mobile-edge computing with inter-user task dependency. *IEEE Trans. Wireless Commun.*, 19(1), 235–250.

Yan, R., Gu, Y., Zhang, Z. & Jiao, S. (2023). Vehicle Trajectory Prediction Method for Task Offloading in Vehicular Edge Computing. *Sensors*, 23(18), 7954. doi: 10.3390/s23187954.

Yang, N., Fan, M., Wang, W. & Zhang, H. (2025). Decision-Making Large Language Model for Wireless Communication: A Comprehensive Survey on Key Techniques. *IEEE Communications Surveys & Tutorials*.

Ye, H., Li, J., Gao, J. & Wen, H. (2025). A Hierarchical Deep Reinforcement Learning Approach for Joint Dependent Task Offloading and Service Placement in MEC. *Electronics*, 14(24), 4816.

Yin, F., Guan, J., Liu, D., Jin, L. & Zhang, Y. (2024). Task Offloading and Resource Allocation for MEC-Assisted Satellite-Terrestrial IoT Networks. *2024 IEEE Globecom Workshops (GC Wkshps)*.

You, Q. & Tang, B. (2021). Efficient task offloading using particle swarm optimization algorithm in edge computing for industrial internet of things. *Journal of Cloud Computing*, 10(1), 41. doi: 10.1186/s13677-021-00256-4.

Yu, F., Tu, B., Wang, Y., Gu, H., Li, Y., Lin, M., Ding, H., Zhou, G. & Zhao, L. (2024). Federated Deep Reinforcement Learning-enabled Task Offloading in Cloud-Edge-Terminal Collaborative Networks. *2024 IEEE 99th Vehicular Technology Conference (VTC2024-Spring)*, pp. 1-5. doi: 10.1109/VTC2024-Spring62846.2024.10683160.

Yuan, Y., Xu, X., Sun, M. & Zhang, P. (2022). Terminal cooperative interdependent computing task offloading for 6g. *IEEE Trans. Network Sci. Eng.*, 9(4), 2846–2856.

Zhang, H. et al. (2024a). Trust-Based Secure Task Offloading in Digital Twin Empowered Vehicular Edge Computing. *IEEE Conference Publication*.

Zhang, H. et al. (2024b). Cost-Efficient Federated Learning for Edge Intelligence in Multi-Cell Networks. *IEEE/ACM Transactions on Networking*, 32(5), 4472–4487.

Zhang, H., Zhao, H., Liu, R., Gao, X. & Xu, S. (2024c). Leader federated learning optimization using deep reinforcement learning for distributed satellite edge intelligence. *IEEE Transactions on Services Computing*, 17(5), 2544–2557.

Zhang, J., Letaief, K. B. et al. (2023). Edge Intelligence in Intelligent Transportation Systems: A Survey. *IEEE Transactions on Intelligent Transportation Systems*, 24(9), 8709–8729.

Zhang, J., Luo, H., Chen, X., Shen, H. & Guo, L. (2024d). Minimizing response delay in UAV-assisted mobile edge computing by joint UAV deployment and computation offloading. *IEEE Transactions on Cloud Computing*, 12(4), 1372–1386.

Zhang, M. et al. (2024e). Dependent Task Offloading for End-Edge-Cloud Collaborative Computing Based on Deep Reinforcement Learning. *IEEE International Conference on High Performance Computing and Communications (HPCC)*.

Zhang, M. et al. (2025a). Joint Scheduling and Placement for Vehicular Intelligent Applications Under QoS Constraints: A PPO-Based Precedence-Preserving Approach. *Mathematics (MDPI)*, 13, 3130.

Zhang, X., Wang, C., Zhu, Y., Cao, J. & Liu, T. (2025b). Multi-Agent Deep Reinforcement Learning With Trajectory Prediction for Task Migration-Assisted Computation Offloading. *IEEE Transactions on Mobile Computing*, 1–18. doi: 10.1109/tmc.2025.3539945.

Zhao, C. et al. (2025). Asynchronous DRL and Trajectory Prediction Based Vehicle Edge Computing Task Offloading. *17th International Congress on Image and Signal Processing (CISP-BMEI)*.

Zhao, G., Xu, H., Zhao, Y., Qiao, C. & Huang, L. (2021). Offloading Tasks With Dependency and Service Caching in Mobile Edge Computing. *IEEE Trans. Parallel Distrib. Syst.*, 32(11), 2777-2792. doi: 10.1109/TPDS.2021.3076687.

Zhao, T., He, L., Huang, X. & Li, F. (2022). DRL-Based Secure Video Offloading in MEC-Enabled IoT Networks. *IEEE Internet Things J.*, 9(19), 18710-18724. doi: 10.1109/JIOT.2022.3161680.

Zhao, T., Li, F. & He, L. (2023). Secure video offloading in MEC-enabled IIoT networks: A multicell federated deep reinforcement learning approach. *IEEE Trans. Ind. Inf.*, 20(2), 1618–1629.

Zhao, X., Wu, Y., Zhao, T., Wang, F. & Li, M. (2024a). Federated deep reinforcement learning for task offloading and resource allocation in mobile edge computing-assisted vehicular networks. *Journal of Network and Computer Applications*, 229, 103941.

Zhao, Y., Li, M., Lai, L., Suda, N., Civin, D. & Chandra, V. (2018). Federated learning with non-iid data. *arXiv preprint arXiv:1806.00582*.

Zhao, Z., Perazzone, J., Verma, G. & Segarra, S. (2024b). Congestion-Aware Distributed Task Offloading in Wireless Multi-Hop Networks Using Graph Neural Networks. *ICASSP 2024-2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*.

Zheng, C., Pan, K., Dong, J., Chen, L., Guo, Q., Wu, S., Luo, H. & Zhang, X. (2024a). Multi-agent collaborative optimization of UAV trajectory and latency-aware DAG task offloading in UAV-assisted MEC. *IEEE Access*, 12, 42521–42534.