Multi-agent Deep Reinforcement Learning-based Task Offloading and Resource Allocation in Fog Computing

Endris Mohammed Ali

A Dissertation Submitted to
The Department of Computer Science and Engineering
College of Electrical Engineering and Computing

Presented in Fulfillment of the Requirement for the Degree of
Doctor of Philosophy in Computer Science and Engineering

Office of Graduate Studies
Adama Science and Technology University

November 11, 2025
Adama, Ethiopia

Multi-agent Deep Reinforcement Learning-based Task Offloading and Resource Allocation in Fog Computing

Endris Mohammed Ali
Main Supervisor: Prof.Jemal Hussien Abawajy
Co-Supervisor: Dr.-Ing. Frezewd Lemma

A Dissertation Submitted to
The Department of Computer Science and Engineering
College of Electrical Engineering and Computing

Presented in Fulfillment of the Requirement for the Degree of
Doctor of Philosophy in Computer Science and Engineering

Office of Graduate Studies
Adama Science and Technology University

November 11, 2025
Adama, Ethiopia

Declaration

I declare that this dissertation entitled "**Multi-agent Deep Reinforcement Learning-based Task Offloading and Resource Allocation in Fog Computing**" is my original work. That is, it has not been submitted for the award of any academic degree, diploma, or certificate in any other university. All sources of materials used for this dissertation have been duly acknowledged through appropriate citations.

Endris Mohammed	_____	_____
Name of student	Signature	Date

Recommendation

We, the supervisors of this dissertation, hereby certify that I/we have read and revised the dissertation entitled **"Multi-agent Deep Reinforcement Learning-based Task Offloading and Resource Allocation in Fog Computing"** by Endris Mohammed, submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Science and Engineering. Therefore, we recommend the submission of the dissertation to the department for further review and defense.

Prof. Jemal Hussien Abawajy	_____	_____
Main Supervisor	Signature	Date
Dr.-Ing. Frezewd Lemma	_____	_____
Co-Supervisor	Signature	Date

Approval Page

I/we hereby certify that the recommendations and suggestions made by the board of examiners are appropriately incorporated into the final version of the dissertation entitled **"Multi-agent Deep Reinforcement Learning-based Task Offloading and Resource Allocation in Fog Computing"** by **Endris Mohammed Ali**.

Prof. Jemal Hussien Abawajy	_____	_____
Main Supervisor	Signature	Date
Dr.-Ing. Frezewd Lemma	_____	_____
Co-Supervisor	Signature	Date

Approval Board of Reviewers

We, the undersigned, members of the Board of Examiners of the dissertation open defense by **Endris Mohammed Ali**, have read and evaluated the dissertation entitled **"Multi-agent Deep Reinforcement Learning-based Task Offloading and Resource Allocation in Fog Computing"** and examined the candidate during open defense. This is, therefore, to certify that the Dissertation is accepted for partial fulfillment of the requirements of the degree of Doctor of Philosophy in Computer Science and Engineering.

Chairperson	Signature	Date
Internal Examiner	Signature	Date
External Examiner 1	Signature	Date
External Examiner 2	Signature	Date

Finally, approval and acceptance of the dissertation is contingent upon submission of its final copy to the Office of Postgraduate Studies (OPGS) through the candidate's Department Graduate Council (DGC) and College Graduate Committee (CGC).

Department Head	Signature	Date
College Dean	Signature	Date
Office of Postgraduate Studies, Dean	Signature	Date

Acknowledgments

I would like to express my sincere thanks to my supervisors, Professor Jemal Hussien Abawajy and Dr.-Ing. Frezewd Lemma, for their incredible and unwavering support and guidance during my study. I would like to express my gratitude to Professor Jemal Hussien Abawajy for his extraordinary support when I faced a difficulty in the publication process.

I want to extend my acknowledgment to Professor Ramasamy Srinivasagan for his unwavering supervision and all-around support for my PhD study.

I would like to express my deepest gratitude to Adama Science and Technology University (ASTU) for providing me with the opportunity, resources, and supportive environment to pursue and complete this PhD research.

My sincere thanks also goes to the Department of Computer Science and Engineering (CSE) for their guidance, academic support, and continuous encouragement throughout this journey. I am also profoundly grateful to the Department Graduate Committee (DGC) and Doctoral Committee for their invaluable continuous feedback and constructive recommendations, which significantly contributed to the improvement of this research work. Furthermore, I extend my appreciation to the College of Electrical Engineering and Computing for creating a collaborative and resourceful academic environment that made this work possible.

Finally, I am deeply grateful to my family, especially my wife, Jemila Mufti, whose love and compassion have been my greatest source of energy and motivation. I could not have become the person I am today without your unwavering support. To my sweet children — Reyan, Ferhan, Afrin, Newal and Genet — your smiles have been my constant source of relief and hope. My mother, you mean everything to me — words cannot express my love and gratitude for you. May you live long and stay blessed. I am truly grateful for your patience, love, and understanding throughout my academic journey.

Contents

List of Figures

List of Tables

List of Acronyms and Abbreviations

Acronyms	Abbreviations
AI	Artificial Intelligence
F_n	Fog Nodes
ML	Machine Learning
DL	Deep Learning
RL	Reinforcement Learning
MAFCPTORA	Multi-Agent Fully Cooperative Task Offloading and Resource Allocation
MADRL	Multi-Agent Deep Reinforcement Learning
MARL	Multi-agent Reinforcement Learning
DDPG	Deep Deterministic Policy Gradient
MADDPG	Multi-agent Deep Deterministic Policy Gradient
IDDPG	Independent Deep Deterministic Policy Gradient
MAPPO	Multi-agent Proximal Policy Optimization
TD3	Twin Delayed Deep Deterministic
MATD3	Multi-agent Twin Delayed DDPG
DRL	Deep Reinforcement Learning
MDP	Markov Decision Process
QL	Q Learning
DQL	Deep Q Learning
DQN	Deep Q Network
DNN	Deep Neural Network
MEC	Mobile Edge Computing
MDP	Markov Decision Process
TD	Temporal Difference
QoS	Quality of Service
U_n	User Device
F2F	Fog-to-Fog
U2F	User-to-Fog

ABSTRACT

Fog computing presents a significant paradigm for extending the computational capabilities of resource-constrained devices executing increasingly complex applications. This approach is applicable to real-time and latency sensitive smart user devices such as consumer electronics (CE), Internet of Things (IoT), TinyML, unmanned air vehicles (UAV), mobile devices, and others are in high demand for resources to process their task according to QoS objectives. However, effectively leveraging this potential critically depends on the implementation of efficient task offloading mechanisms to proximal fog nodes, particularly under conditions of high resource contention. Recent advances in fog computing have enabled decentralized task offloading from resource-constrained smart devices to resource-rich fog nodes. However, determining optimal task placement and resource allocation across distributed, dynamic, and resource-limited fog environments remains a major challenge. The problem scales largely especially when striving to meet stringent Quality of Service (QoS) requirements. Moreover, the existing task offloading model in distributed fog face exhaustive search for selecting right fog node leads to a prolonged decision time problem. Deep reinforcement learning (DRL) has emerged as a promising solution to these challenges, offering adaptive, data-driven decision-making in real-time and uncertain conditions. However, cooperation between fog nodes and dynamic partial offloading model not fully explores in the existing DRL based offloading models. Following that, this work presents a comprehensive and focused analysis on the full-scale application of DRL to the task offloading problem in fog computing environments involving multiple user devices and multiple fog nodes. To address this challenge, we introduce multi-agent fully cooperative partial task offloading and resource allocation(MAFCPTORA) decentralized model for cooperative task offloading and resource allocation. The main contributions of this dissertation include: (i) a decentralized multi-agent DRL architecture for horizontal fog-to-fog offloading, (ii) a cooperative reward function is formulated that optimizes both latency and energy, and (iii) an enhanced evaluation environment for parallel offloading scenarios.

The simulation is conducted in four DRL baseline algorithms such as IDDPG, PPO, SAC, and TD3. TD3 outperform the other three approach, then TD3 algorithm modified

to enable MAFCPTORA parallel task execution. The performance of TD3 base MAFCPTORA are evaluated and compared it against recent baseline approaches. MAFCPTORA demonstrated superior performance compared to baseline methods, achieving a significantly higher average reward (0.36 $\pm$ 0.01), substantially lower average latency (0.08 $\pm$ 0.01), and reduced energy consumption (0.76 $\pm$ 0.14).

Keywords: Fog Computing, IoT, Task Offloading, Deep Reinforcement Learning, Multi-agent DRL, Resource Allocation, TD3

Chapter 1

INTRODUCTION

1.1 Overview

Recently, an explosive growth of resource-intensive smart user devices connected to the network to run their applications. Thus, devices such as consumer electronics (CE), Internet of Things (IoT), TinyML, unmanned air vehicles (UAV), mobile devices, and others are in high demand for resources to process their task according to QoS objectives Javeed et al. (2023); Hortelano et al. (2023). These network-connected devices are capable of sensing, processing, and communicating data. However, because they have limited computing power, they often need to offload tasks to more powerful systems to meet QoS requirements (Hortelano et al., 2023). Figure 1.1 presents the representative smart user

Figure 1.1: Smart user devices

device that face resource constrain problem to compute their task.

Task offloading refers to the process of transferring computational workloads from resource-limited devices to more powerful computing systems, based on a machine-to-machine collaboration model Shah-Mansouri and Wong (2018); Liu et al. (2018). Traditionally,

IoT devices offload tasks to cloud environments. However, device-to-cloud offloading introduces significant latency and bandwidth overhead, as well as potential privacy concerns (Songhorabadi et al., 2023), making it unsuitable for delay-sensitive applications. To meet the growing demands of user devices, Bonomi et al. (2012) introduced fog computing as a mechanism to provide a personalized context-aware service. This architecture enhances real-time task processing and user experience using local information such as user location, search history, and social media activity about environment, behavior, and preference to provide personalized recommendations. According to the OpenFog Consortium Reference Architecture (OFCRA), Fog architecture aims to bring a horizontal computational resource near the data source.

Recent research has suggested that fog computing becomes a candidate solution that allows the user device to execute its computational tasks and store resources on the nearest device. In support of this, Aqib et al. (2023) reveals fog-based offloading to resolve service delay problems in heterogeneous networks. The application of fog computing in industries represented by object tracking Benrazek et al. (2023), Big-IoT data indexing for labeling data Benrazek et al. (2020), smart camera selection for the tracking mission Benrazek et al. (2023). In Fog architecture, user tasks run near the edge of the device and make them more responsive to latency-critical applications for end users Wang and Varghese (2022). Enhances overall communication, and computation costs can be optimized with task QoS requirements and user experience. However, compared to the cloud; fog computing environments still have resource constraint problems. Therefore, to maintain the required QoS level, fog environments demand an optimal and distributed solution to the problem of optimal use of its scarce resources from different nodes. Plus, the Fog architecture allows for re-adjusting the context dynamically to adopt the current context of its resource, and application status to enhance performance. Furthermore, this turned out to be a major challenge in adapting the Fog architecture for offloading tasks.

Task offloading is the process of sending a task from a resource-constrained machine to a capable resourceful computing machine with a machine-to-machine collaboration approach Shah-Mansouri and Wong (2018); Liu et al. (2018). Offloading has become common due to the growing number of smart (IoT) devices connected to the network, and service delivery approaches are becoming more diversified. Specifically, the impact of task offloading observed in real-time applications for those smart technologies to enhance the QoS requirement in various ways: in terms of response time, performance, and providing context-aware servicesWang et al. (2019c); Zhang et al. (2021d); Wang et al. (2019d). This pervasive way of service delivery approaches generates an enormous amount of data that often overwhelms IoT devices in processing it themselves. In this regard, cloud-based solutions provide resources such as network, storage, and computing power to the user device to solve the different computational problems. With this, user devices used this opportunity to solve their problem by offloading computational tasks to the cloud for

processing and sending the result back to end users.

Similarly, maintaining QoS requirements in cloud-based task offloading services faces different challenges, mainly delays in response time and security and privacy. Mouradian et al. (2018) and Salaht et al. (2020) point out the problem of the large geographic distance between the centralized cloud architecture and the end-user device extends response time. In practical service provisioning scenarios, sending a large amount of data to the cloud in limited network resources is inapplicable Hou et al. (2020), and creates a computation and transmission delay Dong and Wen (2019); Puliafito et al. (2019); Al-Khafajiy et al. (2019b). Furthermore, high network traffic leads to latency problems in the round trip to send the task to the cloud and return the results to the user device with high transport cost, and fault tolerance is impractical Ghanavati et al. (2020b). Therefore, this cloud-based offloading does not guarantee real-time service delivery and reiterates the importance of fog computing.

Thus, to run real-time business applications successfully, optimized task offloading mechanisms are proposed from resource-constrained user devices to more capable devices, which are often servers and gateways in a fog Zhu et al. (2021a); Guo et al. (2020). Baek and Kaddoum (2021a) claim that, the computing capability of the fog node is still limited, and running computing-intensive tasks on a single fog node poses QoS problems. To handle this challenge, offloading tasks to multiple fog nodes is proposed as a solution. This also comes with many challenges, such as minimizing communication and task computation delay for latency-critical applications Wang et al. (2019c); Ren et al. (2020a), security, privacy, data management Zhang et al. (2022a), and scalability.

Task offloading and resource allocation in distributed fog computing environments are inherently challenging due to the loosely coupled, highly dynamic, heterogeneous, and failure-prone nature of fog nodes (Ghanavati et al., 2022). Furthermore, due to a dynamic network setting where computing tasks are stochastic and node resources vary in time, task offloading in fog follows a Markov decision process (MDP) Ke et al. (2019); Cheng et al. (2019). Offloading can be more complex in the context of exhaustive search and larger task sizes in fog environments. This complexity is further compounded by variable resource availability, user mobility, and the diverse performance needs of latency-sensitive applications. Achieving minimal communication and computation delays is crucial for enabling real-time analytics and mission-critical IoT operations. Moreover, the decentralized nature of fog computing introduces heightened concerns around data security, privacy protection, and effective data management, all of which are essential for maintaining trust and ensuring regulatory compliance (Zhang et al., 2022a). Different approaches used to deal with task offloading problems between user device and fog environments such as Alternating Direction Method of Multiplier (ADMM) Liu et al. (2018), nature-inspired algorithms (Ant Colony Hussein and Mousa (2020) and ant-Mating heuristics Ghanavati et al. (2020b)), DRL and meta-heuristics compared at Sami et al. (2021), Lagrangian

approach and policy gradient Tang et al. (2020), and greedy algorithms Yousefpour et al. (2019). However, these methods are difficult to converge because the search space grows exponentially and increases run-time processing. Furthermore, applying continuous updating is challenging in fog-based task offloading in the mentioned approaches; those approaches focus on immediate results, which leads to overall performance degradation in the long run Qiu et al. (2021).

Recently, DRL techniques have become an environment in which to play the optimization game. This study pays attention to the practical problems of optimizing task offloading and allocating resources in a fog while satisfying QoS requirements. To address those offloading problems, DRL-based task offloading approaches are getting more attention and becoming a better candidate in fog environments compared to other approaches. Most importantly, this learning-based offloading approach allows us to manage the continuous action space and the system dynamics through continuous learning from interacting with environments Cheng et al. (2019); Zhang et al. (2020); Qiu et al. (2021); Bi et al. (2021); Ke et al. (2019). Meanwhile, this approach also deals with dynamic and heterogeneous fog architecture by observing the current network state information. The implementation of DRL for task offloading problems in fog was presented in previous studies. Some of them focus on DRL for conventional centralized Edge and Mobile Edge Computing (MEC) offloading, while others consider task offloading to the edge, similar to fog Hortelano et al. (2023), but in their implementation, a centralized edge is considered in a top-down fashion.

This dissertation addresses the aforementioned gap by exploring how DRL is used in many-to-many task offloading and resource allocation between smart devices and fog nodes, with a focus on real-time, latency-sensitive applications. In addition, continuous search space and convergence were ignored in their studies. The objectives of this dissertation are to address those gaps with the objectives of (*i*) exploring opportunities to improve decision-making with continuous analysis and learning from data, (*ii*) study dynamic adaptations to deal with changes in network conditions, and (*iii*) assess the architectural setting and providing personalized experience from user data and device information.

This dissertation aims to answer the following fundamental question: How can we minimize system-wide communication and computation latency while simultaneously reducing the energy consumption of fog nodes in the context of executing demanding IoT applications? To this end, we propose a novel approach leveraging multi-agent deep reinforcement learning (MADRL) for efficient task offloading and scheduling on heterogeneous and resource-limited fog computing resources. Integral to this approach is Fog-to-Fog (F2F) cooperation, allowing fog nodes to communicate and coordinate their actions to optimize both local and global performance.

1.2 Fog Computing and Related Paradigm

In this section, a four-layered task offloading architecture, an edge and fog paradigm, and the context of offloading architecture are explored.

1.2.1 Device-Edge-Fog-Cloud Architecture

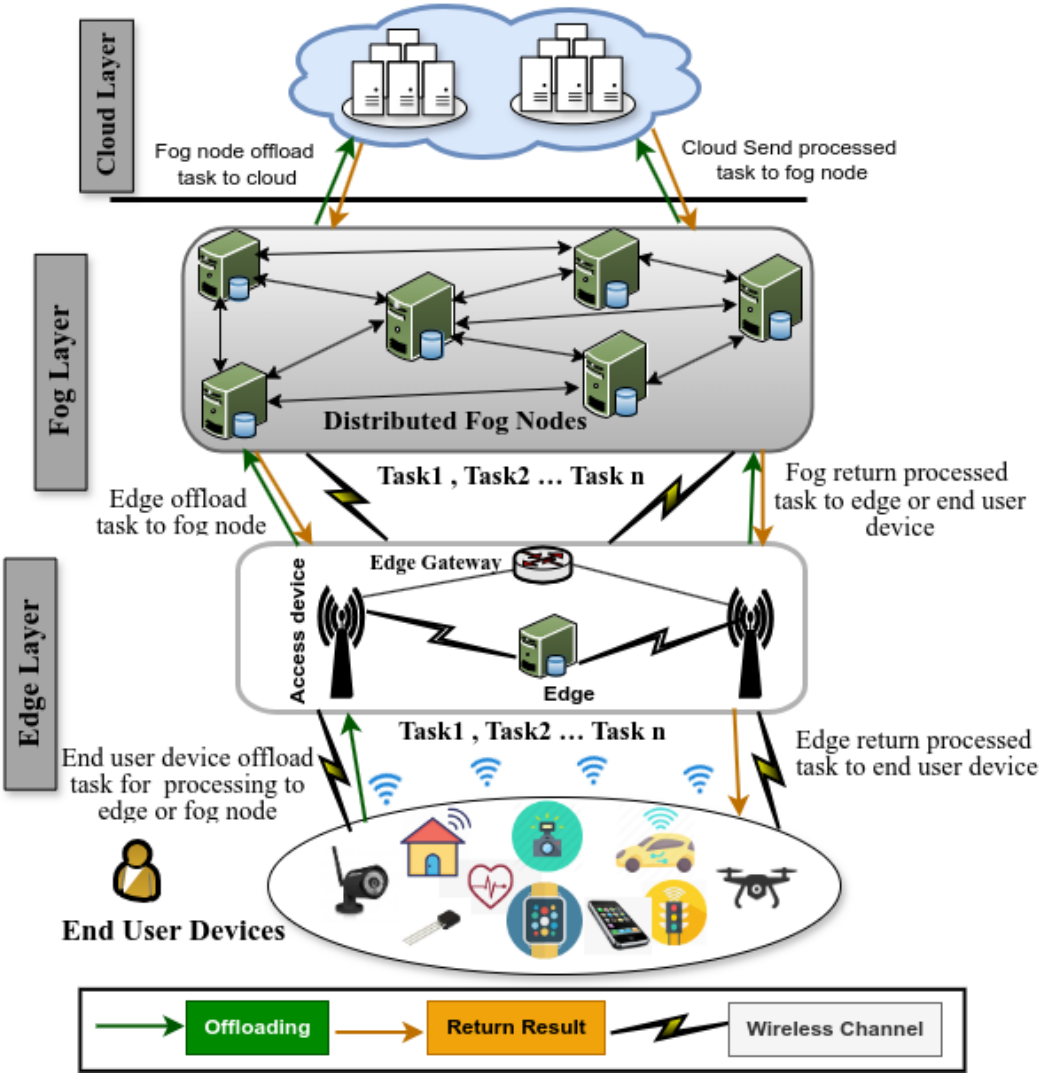

Figure 1.2: End device-Edge-Fog-Cloud Architecture

Figure 1.2 shows a high-level task offloading architecture consisting of four layers: end-user devices, edge, fog, and cloud. The first layer includes various end-user devices with limited computing power. The second layer consists of edge nodes that offer immediate, low-latency support for task offloading and often operate closely with the end-user devices. The third layer, the primary focus of this article, is the fog layer, which provides distributed, intermediate computing resources located near the data source.

Finally, the fourth layer comprises centralized cloud data centers, suited for large-scale processing tasks that are not time-sensitive.

Edge computing is a distributed computing platform in which computation, storage, and networking resources are deployed at or near the network's edge—close to where data are generated. This infrastructure can include gateways, base stations, or local edge servers, enabling low-latency processing without relying solely on centralized cloud resources (Yousefpour et al., 2019). This proximity-based model, illustrated in Figure 1.2, shows access nodes operating parallel to the end-user device. Edge computing is inherently device-centric and localized, with performance highly dependent on the strategic placement of edge servers—particularly in dynamic platforms like the Internet of Vehicles (IoV) (Zhou and Abawajy, 2025).

In contrast, fog computing introduces an additional intermediate layer between edge devices and the cloud, comprising a distributed network of fog nodes capable of more complex, scalable, and context-aware processing tasks Songhorabadi et al. (2023). It adopts a collaborative and horizontally scalable architecture, enabling multiple nodes to share resources, orchestrate services, and perform real-time analytics across wider geographic areas. This makes it especially well-suited for large-scale, latency-sensitive applications, such as smart transportation and industrial IoT, that require rapid decision-making without relying solely on centralized cloud infrastructure.

In summary, fog and edge computing paradigm share the goal of bringing computation closer to data sources to reduce latency and bandwidth usage, yet they differ in architecture, scope, management, scalability, security, and functionality (Yousefpour et al., 2019). Singh et al. (Singh et al., 2019) and Laroui et al. (Laroui et al., 2021) note that both paradigms follow distributed architectural principles but characterize fog computing as decentralized, flat, and hierarchical, with horizontal node collaboration enabling scalable and flexible service deployment.

1.2.2 Fog Offloading Architecture

The architectural scope of task offloading in fog computing defines the structural arrangement and interaction between user devices U_n, fog nodes F_m, and cloud computing (CC) infrastructure. These architectures determine where and how computational tasks are offloaded and processed, which has direct implications for latency, energy efficiency, bandwidth utilization, and QoS. They also influence how learning-based offloading decisions are designed, specifically, how models explore and exploit task-resource coordination in distributed environments. In the user Device–to-Fog setup, tasks are offloaded directly from IoT devices to nearby fog nodes, which provide low-latency processing capabilities closer to the edge. In contrast, the user Device–to-Fog–to-Cloud architecture introduces a hierarchical processing framework, where tasks are first offloaded to fog nodes, and

if resource constraints are encountered, are further offloaded to the cloud. The User Device–to-Cloud model bypass intermediate fog layers, sending tasks directly to the cloud, which may result in higher latency but greater computational power. Lastly, the Fog–to-Cloud architecture supports inter-layer communication between fog and cloud for load balancing, resource sharing, and overflow handling.

As depicted in Figure 1.2, the issue is where to deploy the offloading algorithms in the user device (IoT) layers or the fog layers. Moreover, managing the overall offloading process can be centralized using a single server or distributed across each Fog node is another issue. The other issue is that, considering the mobility of task-executing machines in a dynamic network environment, such as a UAV, the user may lose connectivity when task processing times exceed the existence of the machine in the network area. Most offloading methods rely on fixed rules or predefined criteria, which may not be able to keep up with the dynamic and constantly evolving nature of fog environments.

The current task offloading scenarios in fog computing are illustrated through the three architecture scenarios shown in Figure 1.3 (many-to-one), Figure 1.4 (many-to-many without fog-to-fog cooperation), and Figure 1.5 (many-to-many with fog-to-fog cooperation), where user devices offload computational tasks to fog nodes for processing.

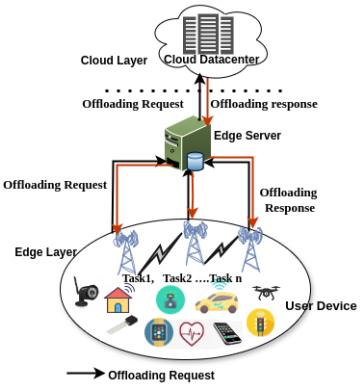

Figure 1.3: Multiple user device offloads their tasks to a single central fog server

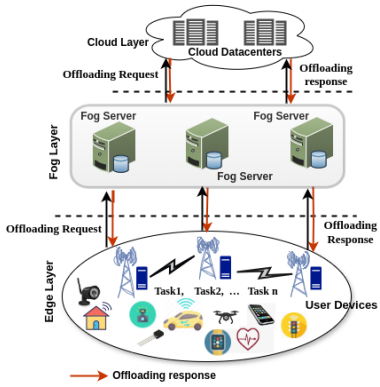

Figure 1.4: Multiple user device offloads tasks to multiple fog nodes, but no cooperation between fog nodes

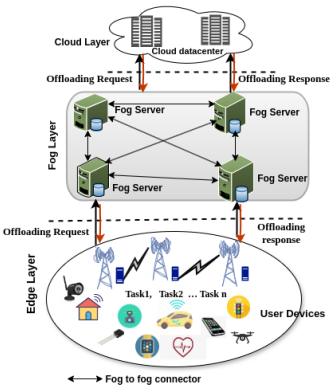

Figure 1.5: Multiple user device offload tasks to multiple cooperating fog nodes

Figure 1.3 illustrates a many-to-one offloading approach, where multiple end-user devices offload their tasks to a single edge server, as presented by Hortelano et al. (2023). This configuration primarily represents an edge computing scenario, as it involves only one computing node positioned above the user devices. In contrast, a true Fog architecture should involve multiple interconnected computing nodes within the Fog layer to support distributed processing. Figure 1.4, adapted from Zabihi et al. (2023), depicts a many-to-many task offloading model, where multiple user devices offload tasks to multiple fog servers using various communication protocols. However, this architecture does not address inter-fog node communication, which is essential for dynamic task migration

and resource balancing across the network. Similarly, Figure 1.5, based on the work of Tran-Dang et al. (2022), presents another many-to-many offloading architecture that introduces the concept of communication between fog nodes in the network. However, the mechanisms of fog-to-fog communication, particularly how fog nodes interact and collaborate, are not explicitly detailed.

A major challenge in fog computing, especially under constrained computational capacity, lies in understanding the appropriate computation model and architectural configuration required to meet QoS standards. In this context, Luong et al. (2020) emphasizes the critical question: "How can limited fog resources be efficiently allocated to IoT users?" Addressing this issue has led to increased interest in leveraging artificial intelligence (AI) and machine learning (ML) techniques to enhance offloading decisions and resource allocation strategies discussed at Zhu et al. (2019); Wang et al. (2019d); Yang et al. (2021).

Recent studies have demonstrated the potential of learning-based approaches in improving task offloading and resource management in distributed fog computing environments Li et al. (2018); Guo et al. (2019); Min et al. (2019b); Ren et al. (2020b). However, a key question remains largely unresolved: How can learning algorithms be further improved and applied to optimize task offloading and resource allocation in fog computing, particularly for real-time applications? In order to meet the QoS demands of real-time applications, a well-designed architecture must integrate an optimized task computation model, intelligent offloading strategies, effective resource allocation models, and robust optimization techniques. Furthermore, appropriate performance metrics are essential for modeling and evaluating offloading decisions to ensure that QoS requirements are consistently achieved.

Overall, the architectural scope plays a critical role in shaping the effectiveness of learning-based task offloading frameworks. A robust decision-making model must align with the operational characteristics of the deployed architecture, whether flat (e.g., device-fog) or hierarchical (e.g., device-fog-cloud), to ensure optimal performance in distributed fog computing environments. Based on this comparative analysis, this study recommends the extended many-to-many task offloading architecture (Figure 1.4) as the most suitable model for applying DRL in fog computing environments. Unlike previous models, it incorporates the critical feature of inter-fog node communication, which is essential for collaborative decision-making, dynamic task migration, and efficient resource balancing. This architecture best aligns with the decentralized and latency-sensitive nature of fog computing and offers the structural flexibility needed for DRL-based task offloading in dynamic IoT systems (Ali et al., 2025).

1.3 Motivation

The rapid growth of smart user devices in smart cities, healthcare systems, and mission critical applications, disaster response networks demands low-latency, and energy-efficient computation at the network edge Hazra et al. (2023). Fog computing handles different real-time applications such as industry applications that generate large and heterogeneous amounts of data from user devices Fahimullah et al. (2023). These data require an efficient process to meet user devices limitations, and this is where bringing AI algorithms to the edge of the network becomes more useful. Traditionally, AI models are deployed on a central server to coordinate offloading decisions and resource allocation Gebrekidan et al. (2024). However, in the fog architecture, the nodes are autonomous and designed in a distributed manner. Therefore, AI learning models can be deployed using either a centralized or decentralized approach in fog architecture. Recently, the autonomous and decentralized nature of fog nodes has led to the adoption of cooperative methods Tran-Dang and Kim (2023), which is the focus of this dissertation, for various applications. The study highlights that DRL algorithms plays a crucial role in coordinating these independent fog nodes, enabling them to make individual decisions while collaborating effectively. This dissertation work delves into designing dynamic multi-agent DRL techniques to optimize both task offloading decisions and resource allocation between smart user devices and fog computing environments.

1.4 Statements of Problem

With the explosive growth of smart user devices and connecting directly with fog computing networks; executing service transactions on this network become a common practice. However, due to resource constraints of fog computing environment processing delay critical real-time application opens a challenge to achieve QoS requirements Baek and Kaddoum (2021a), Zhu et al. (2021a). Furthermore, due to the dynamic nature of fog computing network as well as stochastic computing tasks and time varying fog node resource task offloading become more complex Ghanavati et al. (2022),Baek and Kaddoum (2021b).The fundamental challenge to address in this dissertation is: “how to offload task and allocate scarce resources to applications with varying QoS requirements in a prudent manner to maximize QoS while optimizing system performance for fog computing environment?” . Wakgra et al. (2024b) proposed a ratio-based vertical offloading approach between edge and vehicular fog; however, horizontal fog-to-fog cooperation was not addressed.

The problem is formulated as an optimization problem aimed at minimizing latency

and energy consumption.

$$\min \lim_{T \to \infty} \tau + E \tag{1.1}$$

where (τ) and (E) are latency and energy consumption, respectively. This equation 1.1 is further refined as an average weighted sum in chapter 4 (Section 4.9).

1.5 Research Questions

The following research questions designed to guide the research process. The scope, objectives and methodology are modeled based on those research questions.

1. **RQ1**: How can the decision of where to run a task (i.e., locally on IoT device or offload it to fog) is made under the constraint that both QoS requirements and system performance are optimized?
2. **RQ2**: How to jointly optimize task offloading with response time and energy efficiency requirements of different types in fog computing while ensuring system performance?
3. **RQ3**: How to adaptively allocate resource to applications while considering the performance of the system optimization?
4. **RQ4**: How to balance the trade-off between latency QoS requirements and the overall energy consumption without impacting the system performance?

1.6 Objectives

1.6.1 Major objective

The major objective of this research is to develop a deep reinforcement learning (DRL)-based task Offloading and resource allocation for distributed fog computing networks.

1.6.2 Specific Objectives

The specific objectives of this research work are:

- To develop task offloading and resource allocation algorithm to ensure appropriate decision as to where the task execution is done.
- To design latency and energy consumption aware DRL model while ensuring good performance.

- To develop a multi-agent DRL-based adaptive resource allocation model for optimizing system performance in fog environments.
- To design and evaluate a latency–energy trade-off mechanism for optimizing fog node performance in decentralized task offloading.

1.7 Significance of this study

In this study, we addressed the challenges of task offloading in dynamic, resource-constrained smart user device to fog computing environments, particularly under scenarios with high computational demands, such as large-scale events. These computing nodes become overloaded and congested in the context of events such as football games, concerts, and contests with high computation and traffic demands. Recently, the adoption of fog-to-fog computing architecture has significantly improved task offloading to deal with these problems while achieving task QoS requirements.

DRL-based task offloading in fog computing networks enhances different applications while addressing their QoS requirements. Those applications benefited from the DRL-based fog computing architecture to handle real-time task processing with low latency demands. Some of the applications benefited from fog, smart transportation for tracking vehicles, optimizing routes, and managing peak time efficiently. In healthcare, DRL can optimize the processing of medical data generated by different smart healthcare devices, such as wearable health monitors. The wearable devices offload patient data to the fog node for processing in a real-time manner. By intelligently offloading tasks to fog nodes, healthcare providers can ensure critical patient data are processed quickly and efficiently, enabling real-time monitoring and timely interventions. This approach improves patient care while minimizing computational delay in data analysis. In smart city user devices, edge and fog nodes are integrated to manage utilities such as waste management, resource utilization, distribution, and lightning systems while improving decision-making with DRL algorithms. DRL-based offloading also optimizes power distributions, and demand management can be facilitated with fog-based smart grid ecosystems. Applying DRL-based offloading in smart industries enables the fog paradigm to minimize task processing delay and increase productivity in manufacturing pipelines. Therefore, in industrial automation, the allocation of computational resources across fog nodes can improve operational efficiency, reduce downtime, and improve predictive maintenance strategies. Table 1.1 presents some of the applications that use task offloading supported by fog computing architectures.

Table 1.1: Some representative application of DRL-based task offloading in fog.

Application Type	QoS Objectives	Description	Reference
Smart transportation	Minimizing latency and improving reliability	Traffic monitoring, choosing best route, locating people waiting for service, tracking cars, managing roads, and accidents	Rihan et al. (2020); Raju et al. (2024)
Smart cities	Scalability and energy efficiency	Utility monitoring, smart home locks, smart cameras, smart light control, and smart doorbells	Lee and Lee (2020); Yu et al. (2021)
Smart healthcare	Low latency and high reliability	Real-time decisions, analysis of patients, and data gathering	Tuli et al. (2020); Min et al. (2019a); Le et al. (2024); Seid et al. (2023)
Autonomous vehicle	Latency and reliability	Object recognition and path planning	Hou et al. (2020); Ning et al. (2019); Lee and Lee (2020)
Smart grid	Efficiently balancing energy distribution	Balancing the idle time, resource wastage and peak time demand, and grid status	Zhang et al. (2020)
Smart Industry	To maintain low latency, high throughput, and efficiency	Delay-sensitive task processing demand within pipeline	Li et al. (2018); Ren et al. (2020a)
Disaster recovery	Low latency	Real- time response and context mapping	Zhang et al. (2021d); Wang and Varghese (2022)
Smart supply chain	Enhancing task transmission and communication latency	Monitoring the resource distribution from the source to the end user	Le et al. (2024)
Security and surveillance	Security and reliability	Real-time detection, identifying suspicious activity, and tracking of objects	(Qiu et al., 2021; Benrazek et al., 2023)

1.8 Research Challenges

Task offloading in fog computing and its associated problems are discussed as a research challenge that needs to be addressed in this dissertation.

1. **Challenge 1:** Dynamic task requirements affect the task success rate while optimizing energy consumption and response time.

2. **Challenge 2:** Fog node collaboration in a distributed architecture to share the responsibility of task execution makes the computational efficiency easy. This is directly involving the tasks to be split into sub-tasks so that dependency-heavy tasks coordination needs to be addressed. Implementing the proposed algorithm in practical application scenarios for deployment is also important, as it provides valuable insights.

3. **Challenge 3:** The absence of a multi-objective optimization framework to jointly balance latency, energy usage, bandwidth constraints, and battery health under dynamic and uncertain network conditions is a major gap.

4. **Challenge 4:** Decentralized resource allocation approach causes communication overhead when a fog-to-fog cooperation is important in mission-critical tasks.

1.9 Research Contribution

To deal these research challenges mentioned above, this work explores task offloading and resource allocation problem in F2F cooperation and parallel task execution. In more detail, the offloading problem is a multi-objective mathematical problem whose goal is to optimize task offloading delay and energy consumption. Therefore, we model the problem in an MDP tuple and design F2F offloading algorithm. The main contributions of this dissertation are summarized as follows:

1. We investigate task offloading decisions and resource allocation approaches in fog computing architecture to present detailed concepts, principles, standards, and mechanisms to achieve the application QoS requirements.

2. In this comprehensive exploration, we dive into the patterns of the task-offloading topology model and thoroughly examine its various facets within the realms of resource management and control architectures. Our investigation extends a diverse spectrum that encompasses the interactions of user devices and fog architecture in mixed, mobile, and fixed categories, gaining an understanding of the layered dynamics to play in each of these domains.

3. We explore detailed DL, RL, and DRL-based task offloading and resource allocation in the dynamic landscape of smart user devices, edge, and fog nodes setting to show present and future opportunities. We dive deep into this domain to understand the existing paradigms that pave the way for innovative solutions and strategies that shape the future of smart user devices and Fog resource management. The goal is to bring offloading decisions towards edge of the network, more intelligent, flexible, and efficient systems that can fully utilize the potential of AI/ML techniques, particularly DRL.

4. We investigate an offloading approach by synthesizing cutting-edge research with real-world use cases, providing a comprehensive overview of how this method can foster advancements of QoS requirements of smart user devices, edge computing, and fog computing in their cooperation effort. This study serves as an important resource for those seeking to harness the full potential of the DRL-based offloading approach, ultimately shaping the future of technology and innovation.

5. Formulate the task offloading and resource allocation problem as a Markov decision process (MDP) with a stochastic continuous action space to address computational and transmission delays in a multi-task, multi-node environment.

6. Develop a multi-agent fully cooperative partial task offloading and resource allocation (MAFCPTORA) algorithm that enables parallel task execution across adjacent fog nodes in an F2F architecture, leading to faster execution, reduced latency and energy consumption, and balanced workload distribution.

7. We conduct a simulation and evaluate the performance of our algorithm, demonstrating its effectiveness in improving the deadline fulfillment rate for user device tasks while optimizing the total energy consumption and makespan of the system, thereby benefiting fog service providers.

8. We discuss research direction, open issues, and challenges related to DRL based task offloading in the fog computing paradigm.

These contributions have resulted in two peer-reviewed publications:

- Endris Mohammed Ali, Frezewd Lemma, Ramasamy Srinivasagan, and Jemal Abawajy. Efficient delay-sensitive task offloading to fog computing with multi-agent twin delayed deep deterministic policy gradient. Electronics, 14(11), 2025. ISSN 2079-9292. doi: https://doi.org/10.3390/electronics14112169.

- Endris Mohammed Ali, Jemal Abawajy, Frezewd Lemma, and Samira A.Baho (2025). Analysis of Deep Reinforcement Learning Algorithms for Task Offloading and Resource Allocation in Fog Computing Environments. Sensors, 25(17), 5286. https://doi.org/10.3390/s25175286

1.10 Structure of the dissertation

The structure of this dissertation is organized as follows to give the reader a clear idea. This dissertation is designed with six chapters. Chapter 2 provides background information from the literature related to task offloading techniques used to address the problem and challenges mentioned at (Section 1.8). It also presents associated knowledge domain such as heuristic (Section 2.4), game theory (Section 2.5), reinforcement learning (RL) (Section 2.7), hybrid (Section 2.6) approaches with related work in (Section 2.9). Chapter 3 outlines the research methodology. Chapter 4 presents the DRL-based task offloading and allocation of resource in fog computing environments and system implementations. Chapter 5 presents results and discussion analysis. Chapter 6 concludes the dissertation by providing summaries and future research directions.

Chapter 2

LITERATURE REVIEW AND RELATED WORKS

2.1 Introduction

Recently, task offloading decisions and resource allocation processes between smart user devices and fog computing environments have been a major challenge to improve application QoS requirements. Different approaches have been used to help the user device to choose an offloading target from the available fog nodes. This improves the task offloading process in terms of enhancing resource allocation and task execution delay from the end device to the fog node. Computational tasks and resources are two fundamental elements to consider in fog-based task offloading approaches. In addition, knowing the nature of the application as delay tolerant and delay sensitive affects the level of decision performance.

The key is to explore the knowledge of fog node resources and task requirements for efficient resource allocation. Recently, learning algorithms have been used to extract knowledge from networks. In this regard, Cho and Xiao (Cho and Xiao, 2021) discussed a learning approach for the relationship between task resource demand and the current workload status of the active fog node in a volatile vehicular computing network. In addition, Qi et al. (Qi et al., 2019) claimed a knowledge-driven service offloading decision in heterogeneous computing resources using DRL, considering access network, user mobility, and data dependency. Wang et al. (Wang et al., 2019d) presented task offloading in a joint task scheduling and resource allocation problem to design an optimal policy using a deep Q-learning approach for NOMA-based fog computing networks. Currently, the application of DRL plays a significant role in optimizing the task offloading decision process while adopting an optimal policy. The application of DRL is used to advance the decision process. In addition, hybrid learning techniques are used to solve offloading decision problems.

In the literature review process, the review conducted considering articles published

between 2017 and June 2025 on DRL-based task offloading in fog computing across major academic journals and publishers. We mainly focus on recent advances and emerging trends in DRL-based task offloading and resource allocation in fog computing. The trend analysis between 2016 and 2018, research on fog-based resource management primarily used either DL alone or conventional RL. Starting in 2019, DRL-specific publications grew dramatically in numbers. The increasing awareness that DRL was well adapted to handle the complexity and movement of fog surroundings through its combination of perception (DL) and sequential decision-making (RL) which drove this shift. More sophisticated paradigms including multi-agent DRL (MADRL) and federated DRL approaches began to be researched in 2022. Therefore, they tackle scalability and decentralization more efficiently. These patterns imply that the field is maturing in terms of methodological complexity and practical relevance as well as increasing in volume.

Regarding dissemination channels, there is a significant divide between conference and journal publications. Conference proceedings account for the majority, approximately 60%, of the research output, with IEEE-sponsored venues such as ICC, INFOCOM, and Globecom being the most prominent. These platforms are particularly preferred for the rapid dissemination of novel ideas and early-stage prototypes, often showcasing new DRL-based algorithms or architectures for task offloading. Journal publications, on the other hand—which make up roughly 40% of the literature—usually show up in high-impact venues like IEEE Access, ACM Transactions, and Elsevier's Future Generation Computer Systems. These works tend to provide more comprehensive evaluations, real-world use cases, and longitudinal performance assessments, often addressing challenges like energy efficiency, service latency, and resource utilization in depth. Additionally, to capture relevant information about included studies, the data extraction form considers the year of publication, journal impact factors, study design, fog architecture, DRL techniques and algorithms, task characteristics and evaluation methodology. The inclusion criteria are outlined as follows:

- Research work on task offloading and resource allocation between a user device, edge, and fog computing
- Articles published in peer-reviewed journals addressing task offloading problems using DL, RL, and DRL techniques.
- Studies that propose and evaluate novel DRL-based techniques for task offloading and resource allocation in user device-fog computing
- Consider these QoS performance, including improving latency and energy consumption.
- We consider full-text articles from journals indexed in Web of Science, Scopus, and SCI databases

- Studies published between October 2017 and July 2025

In this section, the major areas of the problem domain and related works addressing the stated problem are investigated. First, key concepts such as fog computing and task computation models are discussed. Next, various algorithms for solving task offloading and resource allocation problems—such as heuristics, game theory, hybrid methods, DL, RL, DRL, and multi-agent DRL approaches—are presented. Following this, the TD3 architecture, which is the proposed DRL algorithm in this study, is discussed in detail. Lastly, related works conducted on the identified problem are reviewed

2.2 Fog computing

Fog computing is an architectural paradigm in which distributed computing nodes collaborate to deliver context-aware, real-time services closer to the data source. It extends cloud computing by decentralizing computing resources and enabling processing at intermediate layers of the network. The concept was first introduced by Bonomi et al. (2012), who described fog computing as a mechanism to provide personalized, context-aware services by situating data, computation, storage, and applications between the cloud and the data-generating sources, such as user devices. Unlike edge computing, fog computing is conceptualized as a multi-layer architecture that spans a wider portion of the network and supports dynamic, reconfigurable applications. It includes edge components while also offering broader computational capacity, acting as a bridge between the edge and the cloud.

2.3 Task Computation Model

In today's pervasive network environments, where user devices are often resource-constrained, offloading computational tasks to fog nodes has proven effective in reducing local processing overhead. However, a key challenge lies in deciding whether a task should be executed locally, offloaded to a nearby fog node, or forwarded to a neighboring fog node. The literature identifies two primary application scenarios in this context. First, executing the application locally may help avoid transmission delays and reduce overall response time. Second, for delay-sensitive applications, executing tasks on resource-limited devices may not satisfy Quality of Service (QoS) requirements such as meeting deadlines or minimizing energy consumption, thus necessitating task offloading.

In practical settings, task execution decisions depend on balancing trade-offs among competing QoS objectives. A task may be offloaded or processed locally based on real-time system constraints and application demands. To work on these challenges, this dissertation proposes a Deep Reinforcement Learning (DRL)-based intelligent offloading

framework that dynamically learns and determines whether to execute tasks locally or offload them to the appropriate fog node, optimizing performance across diverse and dynamic computing scenarios.

2.3.1 Local Model

Practically, computing tasks locally using a user device makes the task transportation and communication costs almost zero. However, the energy consumption and computing power constraints on local user devices while running the application locally incur reliability and performance problems Baek et al. (2019); Guo et al. (2020). The decision criteria to be considered for local task execution can be computation time, energy consumption, device capabilities, and data privacy. For instance, the low battery lifetime and limited computing power in the end-user device affect the performance while achieving QoS objectives. In the literature, researchers consider both local user device resources and offloading some tasks to the fog node to utilize the local resources while boosting performance. The task is divided into numerous sub-tasks, which are then assigned to the local device for local execution. The remaining tasks are then offloaded to fog nodes. Local execution overhead is expressed in terms of execution time (required CPU cycles and computing capability) and local energy consumption.

Extracting both the historical and current task execution status of devices, along with their available resources, enables more informed offloading decisions. Building on this concept, Zhu et al. (Zhu et al., 2019) proposed a computation offloading mechanism aimed at minimizing task completion time and energy consumption within the local fog environment. As reported in previous studies, local execution time and energy consumption overhead are measured by considering the utilization of the CPU cycle and computing capabilities of user devices. In this regard, Wang et al. (2019d) demonstrates the mathematical approach to illustrate the local energy consumption of IoT devices determined by the number of CPU cycles required and the energy consumption of each cycle. As a result, the cost of local execution overhead of end devices can be obtained from the summation of the weight of local execution overhead in terms of the local weight of energy consumption and the weight of task execution delay.

Assume that τ_i^l, f_i^l, and C_i denote the local computation delay to compute task β_i, computing capabilities of the task node i, and the number of CPU cycles required, respectively. So, $\tau_i^l = \dfrac{C_i}{f_i^l} + Q_i^t$, where C_i depends on the size of the task T_i and considers the queue time for each task Q_i^β.

In addition, the energy consumption per CPU cycle ζ_i and the energy consumption for computing tasks E_i^l are considered to calculate the energy cost as $E_i^l = \zeta_i C_i$.

The overall cost of the local task execution Ω_i^l is obtained from the weight of the delay in the execution of the local task plus the weight of the energy consumption cost for that

particular task.

$$\Omega_i^l = \omega_i^t \tau_i^l + \omega_i^e E_i^l \tag{2.1}$$

ω_i^t and ω_i^e are the weights of local task execution delay and energy consumption, respectively. Eq 2.1 allows us to compare the cost of task execution with offloading approaches. In Wang et al. (2019d) propose a learning-based partial offloading approach. That means parallel use of local computing resources to avoid communication and offloading costs, whereas the offloading model is used to benefit from the rich fog computing resources.

2.3.2 Offloading Model

The application of smart user devices grows in a wider perspective, despite that their limited resources pose performance problems. Due to resource constraints on the end user devices, delay-sensitive applications are unable to execute tasks while fulfilling their QoS requirements, such as energy consumption and task computation deadline Ren et al. (2020b); Yang et al. (2020). To handle this problem, offloading computational tasks to resource-rich fog nodes is a promising approach to meet QoS requirements for delay-sensitive applications. This reduces local execution overhead while considering communication and transmission costs. If the decision is to offload the task to the adjacent fog nodes, the major question becomes determining where to offload a task among the distributed fog nodes. The decision criteria to offload the task to the fog node need careful consideration. In this case, the decision metrics can be fog node network resource, communication cost, computation cost, task latency requirements, fog node availability, resource availability, performance requirements, and task complexity. Wang et al. (2019d) also discuss the model of mathematical equations to calculate and analyze each computational aspect concerning the offloading cost. Starting from task transmission delay to each node i can be computed as $\tau_{i,t}^o = \dfrac{T_i}{r_i}$, and computation delay $\tau_{i,p}^o = \dfrac{C_i}{f_i}$, where T_i, p, and r_i are task size, transmit power, and achievable rate for the ith task node respectively. With that, the energy cost can be given as follows in equation 2.4

$$E_{i,t}^{(o)} = p_i \tau_{i,t}^{(o)} \tag{2.2}$$

Where p_i is the idle time after task offload to the ith fog node. The total cost of offloading Ω_i^o the task node i can be calculated as follows using Eq 2.3:-

$$\Omega_i^o = \omega_i^t \left(\tau_{i,t}^o + \tau_{i,p}^o\right) + \omega_i^e \left(\frac{p_i T_i}{r_i} + \frac{p_i C_i}{f_i}\right) \tag{2.3}$$

Energy consumption is measured at the idle time p_i as $E_{i,p}^{(o)} = \frac{p_i C_i}{f_i}$. Where f_i is the assigned computational resources, and this shouldn't exceed the total fog node resources. $\sum_{i=1}^{K} \delta_i f_i \leq F$, where δ is the offloading decision parameter such that $\delta = 0$ the task executed at the local node and $\delta = 1$ the task offloaded to the adjacent fog node. Subsequently, the total cost of task execution at at local and adjacent nodes is expressed in Eq. 2.4, taking into account both the delay and the cost of energy consumption outlined as follows.

$$\Omega_{all} = \sum_{i=1}^{I} \left((1 - \delta_i)\Omega_i^l + \delta_i \Omega_i^o \right) \tag{2.4}$$

However, the major question is how task offloading decisions from user device-to-fog nodes are coordinated. The task offloading and execution process needs to be controlled and managed concerning computing resources and QoS objectives. Accordingly, three offloading decision approaches identified in the literature are partial offloading Wang et al. (2019b), full offloading Jošilo and Dán (2018), and preference-based offloading Tang et al. (2020).

Generally, in the process of offloading decisions, the user device needs to know information about the neighboring task-receiving fog nodes. In this context, Al-Khafajiy et al. (2019b) discuss the request offloading method with a collaboration strategy among fog nodes to balance the fog computation overload. Mostly, the offloading decision problem is associated with communication delay, task queuing delay, and execution delay Wang et al. (2019c), plus the types of application. In the literature, three primary offloading decision approaches have been identified: partial offloading (Wang et al., 2019b), full offloading (Jošilo and Dán, 2018), and preference-based offloading (Tang et al., 2020).

1. **Full Offloading:** Refers to the practice of transferring all data and computation associated with a task from the resource-constrained end-user device to a fog node for complete processing. In this case, the IoT device always favors offloading the task to fog nodes. Jošilo and Dán (2018) applies a game theory model to a full offloading approach in which computational tasks are always offloaded to the neighboring edge, fog nodes, or cloud to optimize computational performance. The problem with this approach is the under-utilization of local device resources.

2. **Partial Offloading:** The assumption is that tasks are divided into sub-tasks; then offload tasks to assign sub-tasks to the fog node and process other sub-tasks locally. In their work,Wang et al. (2019b) proposed a collaborative task offloading scheme for the partial assignment mode, which considers the relationship between the task server, the characteristics of edge servers, and divides the task into several sub-tasks. After that, sub-tasks are offloaded according to the characteristics of the fog server, such as transmission distance and computing power capacity. This proved that this approach solves the random assignment of tasks among servers,

which leads to the large task that needs high computational intensity being assigned to the resource-constrained server. In addition, local device resources are effectively utilized considering QoS requirements.

3. **Preference Offloading:** is a dynamic offloading where the execution decision can be made based on contextual factors such as task needs, device status, and network conditions. In this case, two computational options are considered: processing the whole task locally or offloading partially with conditions such as content similarity and user preference. In this regard, Tang et al. (2020) designed a preference decentralized task offloading approach using the Markov decision process, assuming that the IoT device may have partial state information about the neighbor fog node since node status may change constantly. The challenge in this approach is updating the information about task preference and active node status.

Generally, in the process of offloading decisions, the user device needs to know information about the neighboring task-receiving fog nodes. In this context, Al-Khafajiy et al. (2019b) discusses the request offloading method with a collaboration strategy among fog nodes to balance the fog computation overload. Mostly, the offloading decision problem is associated with communication delay, task queuing delay, and execution delay Wang et al. (2019c), plus the types of application.

The following section provides an overview of various task offloading and resource allocation algorithms designed to solve these optimization problems, highlighting their advantages and challenges within the context of stochastic fog computing networks.

2.4 Heuristic Approach

For computationally intensive environments, these search-based heuristic algorithms are designed to find an optimized solution to complex problems. To mention a few, genetic algorithm (GA) is used for task offloading and resource allocation Shi et al. (2020, 2021), and ant colony optimization (ACO) Hussein and Mousa (2020).

Despite their utility, the literature identifies several limitations of these approaches in a dynamic and complex fog environment. Among these limitations, the lack of generalizability and adaptability in a dynamic and uncertain domain to adapt in real-time network conditions such as vehicle mobility, or diverse task requirements Raju et al. (2024). Traditional optimization and search-based methods can be time consuming and incur prohibitively high computational complexity, especially in large-scale and dynamic systems Bi et al. (2021). They often require a large number of numerical iterations to produce a satisfying solution and may not be suitable for real-time decision-making. Many conventional approaches necessitate complete and accurate network information or prior information of environment statistics Chen et al. (2022). Obtaining such precise and

timely information is often impractical in dynamic fog environments, where conditions such as wireless channels and available computing resources vary continuously. These methods often prioritize immediate performance Qiu et al. (2021), which might lead to performance degradation in the long run if long-term consequences are not adequately considered. Finally, these algorithms relying on exhaustive searches or complex mathematical models can face scalability challenges as the network size and complexity increase Chen et al. (2022).

2.5 Game Theory approach

Game theory in a fog computing environment defines the mathematical model for participating decision makers to cooperate and manage conflicts Rapoport (2012). For this particular case, decision makers are the fog nodes (agents) their main motivation is to enhance individual performance. In essence, game theory offers a principled way to model interactions in task offloading and resource allocation, particularly in multi-user, competitive, or cooperative environments Shahryari et al. (2021). Cao and Cai (2018) considered a non-cooperative game approach to deal with communication and computation costs and machine learning to support distributed offloading decisions under dynamic conditions. Game theory is a powerful analytical tool widely applied in the context of task offloading and resource allocation within various computing environments, such as vehicular fog computing (VFC), mobile edge computing (MEC), and fog-cloud systems Wang et al. (2022). It is particularly useful for modeling scenarios where multiple independent entities (e.g., user devices, vehicles, fog nodes, edge servers) make decisions that impact each other, often in dynamic and competitive environments Mebrek and Yassine (2021).

The players in this game environment (G) are the smart user device U_n and the fog node F_n, mainly between F_n in cooperative strategies.

$$G = (U, \{S_n\}_{n \in u}, F, \{S_n\}_{n \in m,j}(R_f)) \tag{2.5}$$

where m and j are the cooperating adjacent horizontal fog nodes F_n. In equation 2.5, (R_f) stands for resource and (S) is the strategy of the game design, in this case cooperative resource-aware load balancing for collective reward. The agent plays the game to optimize the cost, which is defined as the energy consumption plus the delay in the local and offloading computations as presented in equations 2.1 and 2.3.

However, these game theory approaches have been reported as time-consuming and not applicable to real-time dynamics scenarios due to the issue of scalability and high computational complexity Jiang et al. (2021).

2.6 Hybrid Approach

Conventional offloading approaches, such as rule-based, linear programming, game theory, heuristics, meta-heuristics, matching theory, and various others, consider between user devices and fog nodes to deal with task offloading decisions and resource allocation problems. In addition, different studies also explore hybrid algorithms, including learning algorithms and conventional approaches, to tackle task-offloading decision challenges in the fog computing environment. Some of the representative works are; RL with Heuristics Lee and Lee (2020), Lyapunov guided RL Bi et al. (2021), Federated RL Qi et al. (2021), DRL with FL Yu et al. (2021); Zheng et al. (2022). Recently, more attention has been given to a combined DL with RL called DRL Wang et al. (2020); Zhang et al. (2022a), and FL with RL Qi et al. (2021); Qiu et al. (2021); Zheng et al. (2022) algorithms are used for offloading problems.

However, the constantly changing formation of the fog and unpredictable updates of services make the environment challenging to converge quickly. To deal with this convergence problem, different learning algorithms and optimization approaches are combined to enhance the offloading decision and resource allocation problem. Lee and Lee (2020) presents RL with heuristic approaches for resource management and resource allocation, respectively. Most of these approaches focus on global maxima and the immediate performance of the network. However, incorporating collective local and global knowledge can enhance the decision-making process. In conclusion, the application of the DRL algorithm for solving task offloading problems in such stochastic fog environments where the action space continuously grows becomes more applicable.

2.7 Reinforcement Learning

We model task offloading between smart user devices, edge, and fog as a combinatorial sequential problem. The application of RL becomes more important when the decision-making process is sequential and the objective is long-term, such as in game playing, resource management, and robotic logistics. The environment, state, agent, action, reward, and transition state are the elements that define the RL model. Here, the environment is the playing ground of the user device, edge server, and fog server interaction, and the state is defined by the context of the observed fog resources. An agent is a learning program that perceives and interprets the environment from the network to take action and learn from the exploitation and exploration. The next step in this process is the action when an agent decides whether or not to offload the task based on the fog node status. When the agent takes action, the consequence of feedback can be a positive or negative reward. In RL, the state of an agent is represented in terms of the resource status of the fog nodes, channel condition, or movements of agents from one context of nodes to another

to maximize its reward, which is called the transitional state. In RL, the learning process is dictated using value-based, policy-based, model-based, and model-free training and decision approaches to reward and punish an offloading agent.

Nowadays, different RL algorithms are used to bring long-term performance for resource allocation problems. Currently, most studies use RL algorithms such as state-action-reward(SARSA), Q-learning, Deep-Q-Network, and actor-critic (A2C) to solve these offloading problems with exploitation and exploration. In support of this, Min et al. (2019b) presented an RL-based offloading scheme to select the edge device with an optimal offloading rate based on the current battery level. The author considers the parameters of energy consumption, computation latency, and utility of an IoT device with energy harvesting to design system models. Zhang et al. (2019) deal with the task offloading challenge using a deep Q-learning approach to designing optimal offloading schemes while jointly considering the selection of target server and determination of data transmission mode for mobile Edge computing in urban informatics. However, the challenge is decision-making in dynamic network environments, and resource diversity can not be addressed by considering a stable data set. To address this problem, DRL DRL-based energy-efficient task offloading model is proposed in Khan et al. (2020) for both delay-sensitive and delay-tolerant types of service.

2.8 Deep Reinforcement Learning (DRL)

Recognizing that task offloading in dynamic fog environments, characterized by stochastic computing tasks, varying resources, and fluctuating network channels Ke et al. (2019), aligns well with DRL's capabilities, researchers have explored its application. Zhang et al. (2020) proposed a DRL-based approach for multi-user, multi-fog environments with dynamic workloads and server capacities, aiming to minimize task execution delay and energy consumption, albeit requiring coordination for their hybrid action space. Qiu et al. (2021) addressed the limitations of traditional long-term performance optimization by introducing a distributed and collective training method to maximize cumulative rewards in multi-edge, multi-user offloading scenarios, claiming improved convergence and reduced system cost. Furthermore, Baek and Kaddoum (2021a) presented a deep recurrent Q-network (DRQN) for partially observable, dynamic fog environments with heterogeneous resources, focusing on CPU and memory. Their approach emphasizes local cooperation to maximize aggregated local rewards, with nodes relying solely on local observations. However, the search for a suitable fog node can lead to an exponentially growing search space as task size increases. More recently, Wu et al. (2023) proposed a task-offloading algorithm using an improved multi-agent proximal policy optimization (MAPPO) to minimize delay and energy consumption, while Bai et al. (2023) applied a DQN-based solution to address user request delays and energy consumption.

From an architectural and management standpoint, a key challenge lies in deciding

the optimal placement and management strategy for learning-based offloading algorithms within distributed fog environments, whether centrally managed through the fog gateway or distributed across individual fog nodes. Ghanavati et al. (2020a) emphasized the importance of fault-aware task scheduling in fog computing platforms for ensuring reliability, efficiency, and resource utilization. Thus, dynamic fault-tolerant learning automata (DFTLA) is proposed as an approach to fault-tolerant learning automata task scheduling Jalali Khalil Abadi et al. (2023). The author show case the cooperation of nodes, in each fog node an active variable learning automaton is integrated for a DFTLA-based system, which continuously control the process and makes necessary decisions in response to inputs from the environment. Comparing the proposed DFTLA scheduler with three baseline methods, and evaluated its performance. The proposed algorithm demonstrated reliable task execution can be accomplished with fast response time and low energy consumption. All performance evaluation metrics demonstrate that the proposed approach outperformed the baseline algorithms. Cao et al. (2020) investigated a distributed deep reinforcement learning (DRL) approach, asserting that multi-agent deep deterministic policy gradient (MADDPG) and multi-agent proximal policy optimization (MAPPO) offer more accurate and faster decision-making compared to single-agent systems in real-time scenarios. Similarly, Hussein and Mousa (2020) considered latency constraints arising from the distributed nature of fog nodes, aiming to identify nodes that meet quality of service (QoS) requirements.

Chen et al. (2022) proposed a cooperative multi-agent deep reinforcement learning (CMADRL) framework where IoT devices learn independently but are centrally trained to minimize system-wide costs associated with energy consumption and server rentals. To address these dynamic contexts, multi-agent DRL (MADRL) has emerged as a prominent approach, with agents deployed in both decentralized Gao et al. (2022); Li et al. (2022) and centralized Seid et al. (2023) architectures. Furthermore, Zhang et al. (2021c) categorized multi-agent settings in stochastic task offloading games based on their reward structures: fully cooperative, fully competitive, and mixed. In this vein, Suzuki et al. (2023) and Zhu et al. (2021b) implemented MADRL techniques employing central training with decentralized execution strategies. Recently, the twin delayed deterministic policy gradient (TD3) outperforms other DRL algorithms Wakgra et al. (2024a); Tadele et al. (2024); Ali et al. (2025).

2.8.1 DRL for QoS Optimization Strategies

This section presents a DRL-based approach to task offloading and resource allocation in fog computing, taking into account resource types, QoS objectives, and relevant QoS metrics. Several survey studies have examined the optimization of task offloading across user devices, edge, and fog environments, focusing on factors such as resource availability,

the number of QoS objectives, and performance metrics. Broadly, these optimization strategies are categorized into mono-objective and multi-objective approaches, each aiming to satisfy specific QoS requirements under varying system constraints.

Table 2.1: Resource and QoS objective optimization

Reference	Resource	Objectives	Pros(+) and Cons(-)
Wang et al. (2019c)	Bandwidth and CPU cycle	Minimize task offloading latency	+ Considering fog node and task size for decision − No comparison made with similar works
Zhang et al. (2021d)	CPU, memory, and bandwidth	Minimize energy consumption and computation latency	+ Focus on system stability and task offloading ratio − Offloading relationship is many-to-one
Huang et al. (2021)	Channel state and CPU	Minimize learning and training delay in FL	+ Deal with computation and communication resource − Trade-off cost analysis not covered
Tang et al. (2020)	Battery, CPU, and buffer size	Maximize network performance	+ Analyze decentralized partially observable offloading − Offloading decision and state not covered
Yang et al. (2020)	CPU cycle and time	Optimize offloading decision and resource allocation	+ Weighted cost for local and offloading computation − No network resource included
Cho and Xiao (2021)	CPU and fog resource	Minimize network latency	+ Consider dynamic resource on time-varying fog − No resource utilization status report by adversarial
Guo et al. (2020)	Energy, time slot, and CPU cycle	Minimizing the training delay	+ Compare centralized and federated learning − No comparison on training delay and size of resource
Baek et al. (2019)	Queue space	Minimize the processing time and overloading probability	+ Overall traffic and resource load analysis − Model-free approach raises bias on exploration
Min et al. (2019b)	Battery level and time slot	Reduces the energy consumption, latency, and task drop rate	+ Resource trade-off analysis − Offloading queue space not addressed
Zhu et al. (2019)	CPU and bandwidth	Minimizing task completion time and energy consumption	+ Considers network efficiency and user experience − No application type analysis

Table 2.1 discusses resource optimization with its objective functions and the pros and cons of each work. This table shows the relationship between task offloading and resource allocation in fog computing network settings for different applications. This table demonstrates that optimizing CPU resources is given high attention in the literature.

2.8.2 Training and Decision Approach

DRL combines the representational power of DL with the adaptive decision-making capabilities of RL. While both contribute to DRL's effectiveness, the two paradigms differ substantially in their training processes and decision-making mechanisms. DL is typically suited for scenarios with stable network conditions, where a fixed and pre-defined dataset can be used. In such cases, data are collected, features are extracted and labeled, and the dataset is split into training, validation, and testing subsets. The model is then iteratively trained over multiple epochs to optimize performance. In contrast, RL excels in highly dynamic environments where network conditions change continuously, making fixed datasets impractical. RL agents do not require pre-labeled

data; instead, they interact directly with the environment, observe states, take actions, and receive rewards, gradually improving their policies through trial and error. DRL extends this process by incorporating a replay buffer, which stores past experiences and randomly samples them during training (Zhang et al., 2021d). This mechanism improves learning stability, enhances sample efficiency, and enables the agent to adapt effectively to non-stationary conditions, such as in fog computing, where resource availability and task patterns frequently change.

A key distinction is that DL-based decision-making remains static after training, whereas RL-based decisions evolve continuously as the agent learns from ongoing interactions. The choice between DL, RL, or a hybrid DRL approach depends on the stability of the operating environment and the level of adaptability required.

In fog computing, the DRL training process involves an iterative interaction with the environment, where the agent collects experiences as *state–action–reward* sequences and updates its policy or value function to improve decision-making over time. Depending on application requirements and system constraints, DRL training can follow different paradigms, as summarized in Table 2.2. From a deployment perspective, DRL models can be trained and applied using several modes:

- **Offline Training (Batch RL):** uses a static, pre-collected dataset to train the model before deployment. This allows safe policy learning without affecting live systems, but limits adaptability to unseen scenarios.
- **Online Training:** continuously updates the model during real-time interaction with the environment, enabling adaptive decision-making but potentially increasing computational and communication overhead.
- **Centralized Training:** aggregates all data at a central server for joint model training, offering strong coordination but raising concerns over scalability and privacy.
- **Distributed Training:** splits the workload across multiple computing nodes, reducing training time and improving scalability though requiring robust synchronization mechanisms.
- **Federated Learning-Based DRL:** allows multiple agents or devices to collaboratively train a shared model without exchanging raw data, preserving privacy and reducing communication overhead—especially valuable in heterogeneous and sensitive fog computing environments.
- **Testing:** conducted under both stable and dynamic conditions to evaluate model performance using metrics such as accuracy, latency, energy efficiency, and reliability, ensuring robustness across diverse scenarios.

By selecting an appropriate training mode—offline, online, centralized, distributed, or federated—and leveraging the complementary strengths of DL and RL, DRL-based task offloading systems can combine the efficiency of pre-trained models with the adaptability required for real-world, dynamic fog computing environments. Each training paradigm presents trade-offs between latency, scalability, adaptability, and resource utilization, and the choice should align with the operational objectives, infrastructure capabilities, and privacy requirements of the intended deployment.

Table 2.2: Related works on DRL-Based task offloading with respect to QoS objectives and Training approaches

Reference	Objective	Algorithms	Decision	Training					Pros(+) and Cons(-)
				Online	Offline	Central	Distributed	Federated	
Zhang et al. (2021d)	Optimize joint performance and cost	DQN	Online		✓		✓		+Considering dynamic network -Single Edge node with no control model
Tuli et al. (2019)	Enhance network Latency to meet response time	CNN	Online		✓		✓		+Applying distributed decision - Challenging to update model
Wang et al. (2019d)	Minimize delay and energy consumption cost	DQL	Online	✓		✓			+Optimize offloading between fog nodes -Unknown Helper node status
Sami et al. (2021)	Enhance long learning time	DQN	Online	✓			✓		+Compare DRL with heuristic -Didn't compare learning time of DRL
Xu and Zhu (2021)	Defect inspection problem in production line	CNN	Online		✓		✓		+Distributed assignment with Lie Group -Offloading decision criteria not included
Shi et al. (2023)	Addressing high dimensional continuous action space	DDPG	Online	✓		✓			+Optimal offloading decision on Edge -Left dependency aware task offloading
Wei et al. (2019)	Minimize the service latency	DNN	Online	✓	✓		✓		+Minimize end-to-end offloading delay -critic and actor affect performance
Ren et al. (2020b)	Minimizing long-term system energy consumption	MADQN	Online		✓	✓			+Intelligent updating of computing utilities -No result comparison with previous works
Gebrekidan et al. (2024)	Designing efficient task offloading	DQL	Online		✓		✓		+Consider reliability in time-varying topology -Centralized Offloading decision
Baek et al. (2019)	An optimal offloading decision	DQL	Online		✓		✓		+Offloading using task size and queue -Searching Queue space affect performance
Zhang et al. (2022b)	Computation offloading and resource allocation	DDPG	Online		✓		✓		+Considering time-varying network - Training delay registered
Zhu et al. (2019)	Minimize task Completion time and energy consumption	DNN	Online		✓		✓		+Analysis on getting accurate channel-state -Network resource analysis only
Li et al. (2018)	Maximize accuracy	CNN	Online		✓		✓		+Consider trade-of low latency and energy -No offloading control strategy
Tadele et al. (2024)	Enhance offloading decision	MATD3	Online	✓	✓		✓		+ Introduce ratio-based offloading approach -Left long waiting delay
Wakgra et al. (2024a)	To minimize the latency and energy energy consumption	TD3	Online	-	✓	✓	✓	✓	+Considering dynamic traffic loads -Latency problem on hotspot traffic scenario
Wang et al. (2020)	Application execution delay	A3C	Online	✓	✓		✓		+Comparison of offloading cost -Computation cost not discussed at fog node
Qi et al. (2019)	Design optimal offloading decision policy	A3C	Online		✓	✓			+Propose continual online learning - Search space grow performance issue
Zheng et al. (2022)	Balance learning accuracy and energy consumption	DDPG	Online	✓	✓			✓	+ discuss task size and energy trade-off -Task offloading not covered

2.8.3 DRL Agent Offloading Decision Criteria

DRL algorithms enhance the user device and consumer electronics task offloading decisions to different types of applications based on QoS requirements. The goal of this decision is to select the best fog node based on the QoS parameters to offload its task. The DRL agent uses criteria to make offloading decisions and maximize cumulative reward.

Task characteristics such as compute size, priority, and tolerable delay (Jamil et al., 2023); load information (computational capacity and task queue length); available energy/battery level; local computing capability and channel gains/conditions (Zhang et al., 2022b), network condition (Jamil et al., 2023); offloading mode (local, fog, and cloud) (Rahman et al., 2020; Zhang et al., 2021a); action space (discrete (binary) (Zhang et al., 2022b); continuous (ratio or probability) and hybrid (Chen et al., 2022); CPU cycles/frequency), communication resources (e.g., bandwidth, power allocation) to allocate for the task, either locally or at the offloading destination (Baek and Kaddoum, 2021a); and previous experiences (replay memory storing past states, actions, rewards, and next states) are used to train the agent and inform decisions (Tadele et al., 2024); and device local computation capabilities (Bai et al., 2023).

- **Types of tasks:** The architectural nature of an end-user device network is to execute real-time tasks and generate real-time data. In this context, the problem is how to optimize QoS objectives such as response time and energy consumption. For example, in a delay-sensitive context, task completion time becomes a major criterion while offloading tasks. For delay tolerance tasks, offloading decision criteria might be varied based on the problem to be addressed.

- **Available energy level:** Mainly, one of the criteria from the end user device perspective is task offloading decisions made based on the available energy level of the fog node. For instance, the offloaded tasks are executed to maximize the remaining energy from the fog node by estimating the sum of computation and communication energy consumption (Gebrekidan et al., 2024).

- **Time-slot:** a specified time designed as a change in the balance between transmission time and execution time, with the complete response time the fog node takes.

- **Bandwidth:** the availability of the bandwidth concerning the latency requirement of the application to be offloaded leads to optimized decisions.

- **Deadline:** deals with guaranteeing the resource in the fog nodes to accomplish a task up to its lifetime with the current state of scheduled time and bandwidth.

- **CPU Cycles and Task Size:** The computation task of an end-user device is characterized by two key parameters: the number of required CPU cycles and the

data size, whether for local processing or offloading. These parameters enable each device to make informed task-processing decisions based on both local system state and global system state information.

- **Channel condition and end user device features:** location, priority, satisfaction degree, and computational task are metrics to offload tasks to the fog node.

2.8.4 Evaluation Metrics

Validation metrics are performance indicators used to evaluate a model during or after training, assessing how effectively it meets the objectives of the task offloading problem. Common metrics in DRL-based fog computing include the following:

- Accuracy – the proportion of correct offloading decisions compared to an optimal or benchmark policy.
- Latency reduction – the decrease in task execution time achieved through optimal offloading.
- Energy efficiency – the reduction in energy consumption of devices and fog nodes.
- Throughput improvement – the increase in the number of successfully processed tasks within a given time frame.
- Task completion delay – the total time required to execute and return task results.
- Reward convergence – how quickly and stably the DRL model's reward function reaches an optimal value.

Numerous studies have applied these metrics to evaluate system performance. For example, some have targeted in minimizing latency as the primary objective (Wang et al., 2019e,c; Shi et al., 2020; Alfakih et al., 2020; Rani and Pounambal, 2021). Others have focused on reliability by minimizing offloading costs or balancing trade-offs between energy consumption and delay to optimize overall system cost (Zhu et al., 2019; Wang et al., 2019d; Alfakih et al., 2020; Rani and Pounambal, 2021; Tuli et al., 2019). Additional works address both delay and reliability to reduce network delay costs and enhance offloading decisions (Li et al., 2018; Hou et al., 2020; Rihan et al., 2020; Guo et al., 2019). Finally, some studies emphasize energy efficiency to maintain network reliability (Ibrar et al., 2022).

By selecting appropriate validation metrics, researchers can align evaluation strategies with the intended operational goals of the DRL model, ensuring that performance improvements translate into tangible benefits in real-world fog computing environments.

2.8.5 Common DRL algorithms

To provide contextual understanding, RL is broadly classified into model-based and model-free approaches. Model-based RL leverages knowledge of the environment's dynamics, typically represented by an MDP with transition probabilities $P(s'|s,a)$, in terms of policy iteration $\pi_\theta(s)$ and value iteration $V(s)$ to guide decision-making in stochastic fog computing environments. In contrast, model-free RL does not assume prior knowledge of the environment's dynamics. Instead, it learns optimal policies through interaction, using methods such as policy gradient optimization, off-policy algorithms like Q-learning $Q(s,a)$, and on-policy algorithms including Temporal Difference (TD) learning and SARSA. These approaches are widely used for task offloading and resource management in dynamic fog environments.

DRL extends both paradigms by integrating deep neural networks to approximate policies or value functions, enabling scalability to high-dimensional state and action spaces. Examples include Deep Q-Networks (DQN), which are model-free, and Actor–Critic (AC) methods, which combine policy and value learning. Some DRL frameworks also incorporate model-based components for planning and prediction. This hybridization improves adaptability and efficiency in complex, dynamic systems such as fog computing.

Table 2.3: Common DRL algorithms used for task offloading

DRL algorithms	Methods	Reference
Deep Q-Networks (DQN)	Value-Based	(Wang et al., 2019c; Baek and Kaddoum, 2021a; Ren et al., 2020a; Sami et al., 2021; Khan et al., 2020; Sellami et al., 2022)
Proximal Policy Optimization (PPO)	Actor-Critic	(Lee and Lee, 2020; Wang and Varghese, 2022)
Asynchronous Advantage Actor-Critic (A3C)	Policy Based	(Zhang et al., 2020; Qi et al., 2019; Shi et al., 2020; Lu et al., 2020; Chen et al., 2022)
Deep Deterministic Policy Gradient (DDPG)	Actor-Critic	(Zhang et al., 2021d; Qiu et al., 2021; Gebrekidan et al., 2024; He et al., 2020; Rihan et al., 2020; Seid et al., 2023; Guo et al., 2019; Zheng et al., 2022)
Twin Delayed DDPG (TD3)	Actor-Critic	(Wakgra et al., 2024a; Tadele et al., 2024; Ali et al., 2025)

Several notable DRL algorithms have been applied to address the task offloading problem in fog computing, as summarized in Table 2.3. These algorithms can be categorized into Value-Based (VB), Policy-Based (PB), and Actor-Critic (AC) methods. VB algorithms focus on estimating the value of being in a particular state or performing a specific action. A prominent example is the Deep Q-Network (DQN), which integrates Q-learning with deep neural networks. It incorporates mechanisms such as experience replay and target networks, making it well-suited for environments with discrete action spaces.

PB methods, on the other hand, learn a policy function directly by optimizing the probability distribution over actions. An example is the Asynchronous Advantage Actor-Critic (A3C), a multi-threaded version of A2C that enables faster and more stable training through parallelism. In contrast, AC approaches combine elements of both value-based and policy-based strategies. The Proximal Policy Optimization (PPO) algorithm is widely used due to its simplicity and robust performance, employing a clipped surrogate objective to enhance training stability. The Deep Deterministic Policy Gradient (DDPG) algorithm, designed for continuous action spaces, uses deterministic policies and an actor-critic framework. Twin Delayed DDPG (TD3) further improves upon DDPG by introducing twin critics and delayed policy updates to reduce overestimation bias and enhance stability.

Algorithm 1 provides a generalized pseudo-code that synthesizes common DRL-based task offloading processes from the literature (Seid et al., 2021; Xu et al., 2022; Suzuki et al., 2023; Wakgra et al., 2024a; Ali et al., 2025).

Algorithm 1: Generalized DRL Workflow for Task Offloading in Fog Computing

Input: State of environment$S$; set of possible actions A; reward function R; learning rate α; discount factor γ; replay buffer $\mathcal{D}$

Output: Trained DRL policy π_θ for task offloading decisions

Step 1: Initialize neural network parameters θ and replay buffer $\mathcal{D}$;
for ***Step 2:*** *episode* $= 1$ *to* E **do**
  Step 3: Observe initial state s_0 from fog environment;
  for ***Step 4:*** *timestep* $t = 1$ *to* T **do**
    Step 5: Select action a_t using ϵ-greedy or policy network $\pi_\theta(s_t)$;
    Step 6: Execute a_t: process locally or offload to neighbor/cloud;
    Step 7: Observe reward r_t and next state s_{t+1};
    Step 8: Store (s_t, a_t, r_t, s_{t+1}) in replay buffer $\mathcal{D}$;
    Step 9: Obtain random minibatch sample from $\mathcal{D}$;
    Step 10: Update θ by minimizing policy gradient objective or temporal difference loss;
    Step 11: Set $s_t \leftarrow s_{t+1}$;
Step 12: return optimal π_θ;

2.8.6 Key Input Parameters and Learning Mechanisms in DRL

The agent captures input features from the current network state of the system that the agent observes and utilizes for decision-making in DRL-based task offloading. In stochastic fog computing environments, the agent's decision-making process uses this state information as the building blocks, allowing it to adjust to shifting circumstances. In practical implementations, numerous input features, including CPU capacity, memory usage, channel conditions, energy levels, and application deadlines used in DRL models.

By considering both historical data and real-time measurements, the DRL model optimizes offloading decision-making and resource utilization, registers high QoS under varying conditions, and allows the system to be more adaptive and context-aware.

The replay buffer, which retains the agent's prior experiences, including offloading decisions, system states, plus associated rewards, is a crucial learning mechanism in DRL. In order to improve learning stability, sample efficiency, and avoid overfitting to recent interactions, these stored experiences are randomly sampled and reused during training. This approach also allows generalization to unknown scenarios, which is particularly important in non-stationary scenarios—such as vehicular fog networks, UAVs, and others —where resource availability and task patterns change frequently.

DRL-based offloading strategies can achieve stability and adaptability by combining robust learning mechanisms, such as the replay buffer, with rich state information. This ensures more dependable decision-making in complex, real-world deployments.

2.8.7 Single Agent DRL

Fog computing is inherently distributed, relying on decentralized management for low-latency and scalable service delivery. To support intelligent offloading and efficient resource allocation, AI/ML methods are increasingly adopted. In such architectures, users can execute their applications on one or more fog nodes as a single-agent approach. In this case, the participating fog nodes may be either single or multiple; however, the offloading decision and resource allocation for the given task are managed by a single agent functioning as a central controller.

In this setting, Baek and Kaddoum (2021a) emphasizes a multi-fog environment, employing a single-agent deep recurrent Q-network (DRQN) approach under partial observability, where offloading and resource allocation decisions are made based on local observations. Moreover, in vehicular fog computing, a single-agent approach is applied to select suitable computing nodes, such as RSUs or service vehicles, as discussed by Jamil et al. (2023). Abdelghany (2025) applies single-agent Deep Q-Learning (DQL) to address the dynamic and heterogeneous nature of fog environments, which are affected by unstable computational nodes, fluctuating channel conditions, and node mobility patterns. In these single-agent models, task and resource management can hinder performance in decentralized fog environments where nodes operate autonomously.

2.8.8 Multi-Agent DRL

Multi-agent deep reinforcement learning (MADRL) has emerged as a powerful paradigm for tackling complex resource management challenges in distributed systems. This MADRL has emerged as a robust alternative, enabling fog nodes to coordinate their decisions. This cooperation helps reduce computational delays and enhances the overall utilization

of resources across the system. Here, the challenge is how to coordinate the agent policy and inter-agent communications. Two MAL settings are considered in RL-based task processing; the first is the cooperative approach, where the fog nodes (DRL agents) collaborate to achieve common objectives; agents share information to achieve the best result for the team. The other is the adversarial approach, in which agents compete to outperform their individual goals. Recently, DRL enabled MAL to take its stage to apply to different problem domains. The MADRL used in MEC enabled the Industrial Internet of Things (IIoT) for multi-channel access and task offloading problems Cao et al. (2020). Lu et al. (2020) design a multi-agent DRL approach for independent learning agents to solve stochastic end-user massive requirements change and computational resource. MADRL is an extension of reinforcement learning that enables multiple agents to learn optimal policies through direct interaction with uncertain and dynamic environments (Suzuki et al., 2023). A key strength of MADRL lies in its capacity to accelerate policy learning for complex tasks via parallel execution, which is particularly effective in cooperative and distributed systems such as fog computing (Zhang et al., 2022c; Oroojlooy and Hajinezhad, 2023).

However, coordinating multiple autonomous agents introduces several inherent challenges that significantly complicate policy optimization. A primary difficulty is the environment non-stationarity(Canese et al., 2021). In a multi-agent system, the agent's environment changes as other agents update their strategies, violating the stationary environment assumption underpinning most single-agent RL algorithms. This continual policy drift makes it harder for agents to converge toward stable, optimal solutions. Another critical issue is the credit assignment problem, where determining which agent's actions contributed most to a team's success or failure is non-trivial(Foerster et al., 2018). In cooperative scenarios, improper reward allocation can hinder effective policy updates, especially in sparse-reward environments. Scalability also presents a substantial challenge. As the number of agents grows, the joint state-action space increases exponentially—known as the curse of dimensionality—making centralized learning computationally prohibitive (Zhang et al., 2021b). Similarly, many real-world applications shows partial observability, where agents have only local information about the system state (Lee et al., 2021). This limitation forces agents to reason about hidden variables and infer the intentions of other agents, increasing decision-making complexity. Finally, communication and coordination remain open research problems. Designing protocols that enable agents to exchange information efficiently without incurring excessive bandwidth or processing overhead is critical (Nguyen et al., 2020). Chen et al. (2022) introduces cooperative MADRL for multi-device multi-cloud dynamic network settings for real-time computing requirements in varied wireless channels in a centralized training and distributed execution strategy.

The task execution methods in MAL can be sequential or parallel; the choice between them depends on the specific learning algorithm and the interaction pattern between

agents. For instance, the sequential method can be applicable in an environment where the agent learns independently from their own experiences without needing to interact with others. Sequential execution is also applied in hierarchical learning in such a way that agents learn at different levels, with higher-level agents making decisions that influence the learning of lower-level agents sequentially. Therefore, this approach becomes more practical when computing resources are limited. In this approach Gebrekidan et al. (2024) discusses it from a centralized management perspective, where a central coordinator is designed to manage the multi-agent setting. Moreover, with parallel execution scenarios, agents learn concurrently through interacting and sharing information, allowing them to influence each other's learning and actions in real-time. Additionally, it is also applicable for faster learning; When dealing with complex tasks, parallel execution can accelerate learning compared to sequential execution, especially for cooperative settings.

Its applicability spans diverse domains, including UAV-based task offloading and resource allocation in air-to-ground (A2G) networks Seid et al. (2021), where it optimizes for average local and global maxima in continuous search spaces. MADRL has also been instrumental in enhancing vehicular fog resource utilization for many-to-many task offloading in VFC networks Wei et al. (2023) and in managing task offloading within space-aerial-ground integrated networks (SAGIN) Li et al. (2022), while ensuring user device quality of service (QoS). Furthermore, Suzuki et al. (2023) reinforce the utility of cooperative MADRL in addressing reinforcement learning problems within multi-cloud edge networks. In this context, cooperative decentralized MADRL refers to a system where multiple agents operate autonomously, making decisions based on their local information, but with an overarching goal of achieving a shared objective or maximizing a common system-wide reward Baek and Kaddoum (2021a); Suzuki et al. (2023). Building upon this foundation and drawing inspiration from Seid et al. (2023), which explores multi-UAV environments with unreliable computational resources, this work formulates the joint task offloading and resource allocation problem as a stochastic game. Multi-agent TD3 (MADRL-based TD3) Tadele et al. (2024) is proposed to address challenges in large systems by reducing the input and action space for individual agents, leading to faster convergence and more efficient processing than single-agent TD3. A key distinction of our proposed fog network lies in its composition of both reliable and unreliable computational sources and communication networks, enabling adaptation to a wider range of situational scenarios.

2.8.9 Agent Controller Strategies

The coordination of agent communication between the task-owned node and the task executor node is discussed in the literature as a centralized, distributed, and federated approach. In the centralized model, a central controller is responsible for making all task offloading decisions. The central controller gathers information from various fog nodes

to assign tasks and monitor execution across the system. The coordination of agent communication between the task-owned node and the task executor node is discussed in the literature as a centralized, distributed, and federated approach. In the centralized model, a central controller is responsible for making all task offloading decisions. The central controller gathers information from various fog nodes to assign tasks and monitor execution across the system. The central server is where the decision-making and model training processes take place. Although this architecture makes task offloading coordination easier, scalability becomes a serious problems that raise the possibility of a single point of failure. In addition, this model is not appropriate for real-time, on-the-fly processing of raw data or for dynamic, latency-sensitive applications. This model works best when one organization or entity manages all of the computational resources (Wang et al., 2019e).

The distributed architecture typically follows one of two approaches. In the first, task computation is delegated to multiple fog nodes, while offloading decisions are still managed centrally. In the second, both task computation and offloading decisions are handled in a fully distributed manner. For the latter approach, distributed learning techniques are employed to support the offloading process by sharing communication and execution information across nodes. In this approach, to enable decentralized control of the offloading process in each fog node is responsible for controlling resource coordination and the task execution process (Huang et al., 2018; Liu and Liu, 2018; Qiu et al., 2021). Conversely, reliability, security, and privacy issues arise when private information is offloaded to unidentified or unreliable fog nodes F_n. Frequent task transmissions may also result in an increase in network traffic, which could impair system performance.

Locally generated data from user devices is frequently offloaded to edge, fog, or cloud servers for training and decision-making in traditional centralized and distributed learning approaches. Federated learning was introduced by (Wahab et al., 2021) as an alternative offloading strategy to address important issues like network traffic, data privacy, and security. Federated architectures eliminate the need to send raw data to fog nodes by performing model training locally on user devices or in a heterogeneous network environment. Rather, distributed nodes—where decision-making can also take place—only receive the trained model or its updates. This method is ideal for latency-sensitive and privacy-critical applications, as it improves data security and privacy while minimizing communication overhead.

2.8.10 Coordination Mechanisms

Task offloading architectures are typically classified according to the coordination mechanisms among nodes that provide computational resources for executing tasks. These architectures fall into three main topological categories: (i) static (fixed) topology, (ii) dynamic topology characterized by node mobility, and (iii) hybrid topology, which combines features of

both static and dynamic configurations. This section examines each of these architectural models, discussing their practical application scenarios, the unique challenges they present, and the solutions proposed in the existing literature.

Static Model: In this architectural approach, both the fog nodes and the client user devices' status and placement remain unchanged over time. Here, the architecture follows a controlled approach since the task offloading node and receiving fog nodes are known. The representative application scenarios mentioned, such as manufacturing industries (Hou et al., 2020), involve fog nodes and IoT devices such as cameras and sensors that are deployed in fixed locations. Furthermore, smart traffic monitoring scenarios for stationary traffic nodes are also considered an example of fixed topology, since both the cameras and traffic light sensors are installed in a constant location (Markus and Kertesz, 2020). Smart city utilities such as public safety, optimized service delivery, and others (He et al., 2018) can be designed with this type of architecture. Resource management in this architectural model exhibits lower complexity compared to dynamic and hybrid models (Hong and Varghese, 2019); however, potential concerns relate to underutilized resources and inflexibility in allocating them for collaborative tasks involving other devices.

Dynamic Model: In this architectural model, the topology designed for both the condition of servers and clients is constantly changing their status (Wang et al., 2019a). This dynamism reflects both the resource status of the participating nodes and their mobility in joining or leaving the network, such as in vehicular fog computing. This implies that the workload, location, and resource availability of a node are all unpredictable. Key features like dynamic resource allocation, node mobility predictions, handling smooth tasks and data offloading, context-aware computing, and others are taken into account in this mobility-aware fog computing network configuration (Waqas et al., 2019). In particular, by using location and contextual information to optimize task processing and decision-making at the respective nodes, it guarantees resource utilization and service continuity between dynamic fog nodes when devices move between coverage areas. Tan et al. (2019) compared stationary and non-stationary contexts to study task offloading in dynamic fog computing networks with non-stationary topologies and variable computing resources. While their model assumes constant unit-task offloading delay under static or slow-changing conditions, real-world node mobility—characterized by unpredictable speed, distance, and duration within coverage—requires intelligent handling. In multi-hop ad hoc fog environments, such mobility can significantly impact application performance. Similar concepts have been applied in UAV-based fog computing for the Internet of Medical Things (IoMT) in rural, resource-limited areas (Seid et al., 2023) and in vehicular fog computing (Lee and Lee, 2020), where both task-requesting and task-executing nodes

are in motion.

Hybrid Model: The hybrid model is where the network context has a mix of dynamic and static models. In conventional hybrid fog architecture, the fog nodes typically consist of fixed devices, whereas the user devices are mobile (Hazra et al., 2023). However, in this mixed network setting, dynamic and static settings are introduced, and the architecture becomes even more dynamic and requires additional considerations. This category considers the fog node as having a constant location as static, and clients' status is mobile or vice versa according to priority settings (Verma et al., 2021). The application scenarios for this model are represented as traffic monitoring and surveillance (Tuli et al., 2023), environmental monitoring (Jan et al., 2019), and UAV-based service provision mechanisms in a rural area presented in (Rejiba et al., 2019), where there is no stable network connection. The UAV acts as a server to provide an access point for the local end devices. It can also be applicable to provide communication services in emergency response in a disaster context. (Hazra et al., 2023) explains the use case of a health monitoring environment where fog nodes are fixed, but patient status monitoring IoT devices are in mobility modes. The opportunities in the mixed architectural model are the separation of layers, simplifying resource management, improved security, and data privacy through local processing in each context. The challenge in this fixed architecture is potentially higher upfront costs compared with other approaches.

In dynamic and hybrid architectural models, managing the complexity of mobile fog nodes requires advanced control mechanisms. These nodes often have limited battery life, making efficient task processing and communication protocols essential for optimization. Their unstable nature can lead to task failures, disrupting dependent tasks offloaded to other nodes. Ensuring secure communication and data privacy in such dynamic environments is also a key challenge. To address these issues, learning-based mechanisms that predict node mobility patterns and network characteristics—such as long-term connectivity and stable resource availability—are increasingly necessary. Recent research proposes distributed, decentralized control strategies to adapt to changing network conditions, with emerging focus areas including self-organizing networks and AI-driven resource management.

An important aspect in these models is awareness of fog node and resource status for effective task offloading. Pu et al. (2016) introduced a device-to-device (D2D) communication approach to enable collaboration between edge devices, supported by an incentive scheme similar to peer-to-peer networks. Similarly, (Al-Khafajiy et al., 2019b) proposed collaborative strategies between fog nodes for shared workload processing. Through D2D communication, IoT devices can exchange status information with nearby devices to identify idle resources, enabling adjacent devices and fog nodes to coordinate via a wireless access point. In fixed-to-fixed architectures, device routing tables advertise

and update resource status for all connected nodes. In both cases, the collected knowledge supports informed offloading decisions. Future work should explore DRL-based self-organizing network settings for smart user devices and fog computing environments that integrate both fixed and dynamic architectures.

2.8.11 Twin Delayed Deep Deterministic Policy Gradient (TD3)

Twin-delayed DDPG (TD3) is a DRL algorithm designed for tasks with continuous action spaces. The deep deterministic policy gradient registered significance performance level; however, frequently tuning the hyperparameter creates a drastic Q-value over estimation. This overestimation leads to a policy break, because it exploits the error in the Q-function. This is where the TD3 algorithm is designed to address the issue by introducing three critical tricks presented by Fujimoto et al. (2018). These first tricks, Clipped Double-Q Learning, where TD3 learns two Q-functions, and uses the smaller of the two Q-values to form the targets in the Bellman error loss functions. Second, delayed policy updates, that TD3 updates the policy and target networks less frequently than the Q-function. The paper recommends one policy update for every two Q-function updates Fujimoto et al. (2018). Lastly, target policy smoothing, in this case TD3 adds noise to the target action, to make it harder for the policy to exploit Q-function errors by smoothing out Q along changes in action.

Wakgra et al. (2024a) use TD3 to handle stochastic states of vehicle to fog communication and to maintain stability, increase learning efficiency, and minimize overestimation bias compared to DDPG. Two value functions estimate state-action pair values in a clipped Double-Q Learning, and the minimum estimate is used as the goal value during updates. The target value clips a specific range, which helps to mitigate overestimation problems that are frequently seen with single Q-value estimators. The offloading action used the target policy smoothing from the Q-learning target based on target policy ($\mu\theta_{targ}$) with clipped noise, and then the action satisfied the probability of ($\alpha_{low} \leq \alpha \leq \alpha_{high}$). Therefore, the target offloading actions are defined as

$$\alpha'(s') = clip(\mu\theta_{targ}(s') + clip(\epsilon, -c, c), \alpha_{low}, \alpha_{high}, \quad \epsilon \sim \mathcal{N}(0, \sigma) \tag{2.6}$$

Equation (2.6) adds exploration noise $\mathcal{N}$ to adapt the deterministic policy into a stochastic behavior during the critic update, which helps stabilize learning.

The clipped double-Q learning is defined where both of the Q-functions use the single target to find a smaller target value using the two Q functions:

$$y(r, s', d) = r + \gamma(1 - d) \min_{i=1,2} Q_{\phi_{i,\text{targ}}}(s', a'(s') + \epsilon) \tag{2.7}$$

Equation 2.7 shows the mean square Bellman error (MSBE), then, both Q-functions learn

by regressing to the target as Equation (2.8) and (2.9):

$$L(\phi_1, \mathcal{D}) = \mathbb{E}_{(s,a,r,s',d)\sim\mathcal{D}}\left[\left(Q_{\phi_1}(s,a) - y(r,s',d)\right)^2\right] \tag{2.8}$$

$$L(\phi_2, \mathcal{D}) = \mathbb{E}_{(s,a,r,s',d)\sim\mathcal{D}}\left[\left(Q_{\phi_2}(s,a) - y(r,s',d)\right)^2\right] \tag{2.9}$$

The parameter $(\mu\theta(s))$ is then updated by carrying out the gradient descent of the performance measure $Q(\phi 1)$.

A TD3 agent is an actor-critic DRL agent that aims to find the optimal policy to maximize the expected cumulative rewards over the long term. The actor network determines the action, such as the offloading decision or ratio distribution. In contrast, the critic network evaluates the value (expected reward) of taking that action in a given state.

In summary, existing research predominantly focuses on vertical task offloading (to edge, fog, or cloud), with less attention given to horizontal offloading among peers or neighbors. Furthermore, simultaneously considering factors like mobility and task dependencies remains a significant challenge. Notably, few prior works in fog computing address task priority, neighbor service availability, or resource contribution incentives. The complexity and robustness of decision-making algorithms, especially their decision time, are also often overlooked.

2.9 Related Work

For a long time, many researchers have conducted task offloading on fog computing for different business applications. These articles discuss resource allocation, infrastructure, algorithms, architecture, management model, and computation offloading Puliafito et al. (2019); Lan et al. (2019); Lahmar and Boukadi (2020); Kar et al. (2023). For instance, Lahmar and Boukadi (2020) consider traditional optimization approaches while exploring resource allocation issues in fog computing networks to improve QoS for latency-critical applications.

Hong and Varghese (2019) discussed the classification of architecture, infrastructure, and underlying algorithms, and Salaht et al. (2020) explained a problem of service placement and resource management in edge and fog computing. Kar et al. (2023) present offloading in a federated system and provide a classification between edge, fog, and cloud, with a road map on offloading approaches for different federated scenarios. Mukherjee et al. (2018) discuss cloudlets-based mobile offloading for mobile and non-mobile applications.

In the papers mentioned above, conventional centralized task offloading and resource allocation approaches have been employed, such that many smart user devices send tasks to the central edge node server for task execution decisions. This approach affects

the flexibility and efficiency of offloading decisions in a distributed fog architecture. The potential of utilizing DRL approaches to optimize the task offloading process in different architectural settings is discussed in advance. From this perspective, Fahimullah et al. (2023) investigate the resource management problem in dynamic fog computing environments, but task offloading is not their main concern. Among the work on task offloading, Shakarami et al. (2020) provides a detailed overview of computation offloading using supervised, unsupervised, and RL for IoT-MEC ecosystems. Similarly, Rodrigues et al. (2020) explore the use of ML and the expectation of a high level of dimensional data on offloading decisions. Despite the use of ML, the status of resources allocated for the task and the inter-server communication needs online updating. To deal with this type of dynamic context update Hortelano et al. (2023) present DL and RL techniques for task offloading problems in IoT-Edge networks. However, the authors consider many to-one offloading approaches, depicted in their architecture, where many user devices offload to single central edge nodes. Further building on, the survey by Tran-Dang et al. (2022) shows that RL is used for fog resource allocation, optimizing task offloading to improve execution performance, but deciding where to offload tasks not yet fully addressed remains the main challenge.

The investigation of task offloading consider multiple user device have opportunities to offloads their task to multiple fog nodes. Existing research tries to address task offloading problem where to place task among multiple alternative neighbor nodes. This coordination directly influence different QoS requirements such as latency, energy consumption, and service reliability. The network topology explores in this domain mainly categorize as vertical offloading and hybrid initiated from user device Wakgra et al. (2024a). Some studies propose efficient offloading strategies Wakgra et al. (2024b); Bai et al. (2023) aim to minimize task completion time and energy usage while maintaining acceptable service levels for user applications in fluctuating fog node workloads. Despite growing interest in fog computing, the horizontal F2F offloading topology and its coordination mechanisms are still insufficiently investigated.

Together, these related works highlight the need for further investigation of task offloading in dynamic and heterogeneous fog computing ecosystems. This dissertation explores the use of AI/ML techniques specifically, DL, RL, and DRL for many-to-many task offloading and resource allocation in the smart user device-to-Fog settings. We addresses these limitations by developing a multi-agent DRL model that jointly optimizes latency and energy efficiency while maintaining QoS guarantees in F2F task offloading scenarios. The scenario involves multiple user devices considering offloading their tasks to multiple distributed Fog nodes to meet real-time application demands. Moreover, it addresses how DRL can optimize offloading decisions and resource utilization for latency-critical applications in IoT, edge, and fog environments. Finally, this study provides insights on knowledge-driven multi-agent DRL-based task offloading and its

applications in emerging technologies.

Table 2.4: Related Research Work.

Domain	Problem	DRL Algorithm	QoS Objectives	Agent Type	Network	Papers
VFC	V2V Partial Computation Offloading	SAC	Minimize task execution delay and energy consumption	Multi-Agent (UAV)	Dynamic	Shi et al. (2021)
IoT-edge-UAV	Task offloading and Resource allocation	MADDPG	Minimize computation cost	Multi-Agent (UAV)	Dynamic	Seid et al. (2021)
Vehicular Fog	Optimize Ratio-Based Offloading	DQN	Minimize system latency and decision time	Single-Agent	Dynamic	Wang and Varghese (2022)
VFC	Minimize waiting and delay	PPO	Minimize delay and packet loss; maximize timely task completion	Single-Agent	Dynamic	Jamil et al. (2023)
Multi-Fog Networks	Joint Task Offloading and Resource Allocation	DRQN	Maximize tasks, minimize buffer overflows	single-agent	Dynamic	Baek and Kaddoum (2021a)
Edge-UAV	Task offloading	MADDPG	Minimize energy consumption and delay	Multi-Agent UAV	Unstable	Li et al. (2022)
IoMT	Computation offloading and resource allocation	MADDPG	Latency Reduction	Multi-Agent	Unstable	Seid et al. (2023)
Multi-cloud (Fog)	Dynamic Computation Offloading	CMATD3	Minimize long-term system cost	Multi-Agent (Cooperative)	Dynamic	Bai et al. (2023)
MEC	Multi-channel access and task offloading	DDPG	Minimize computation delay	MTA	Stable	Cao et al. (2020)
Vehicular Fog	Load Balancing problem	TPDMAAC	Minimize system cost	MV	Hybrid	Gao et al. (2022)
Vehicular Fog	Task offloading and computation load	AC	Maximize resource trading	MA-GAC	Hybrid	Wei et al. (2023)
Fog-RAN	Computation Offloading and Resource Allocation	DDPG	Minimize delay and energy	Multi-agent (Federated)	Dynamic	Tong et al. (2024)
MEC and Vehicular Fog	Traffic congestion and overload	TD3	Minimize latency and energy	Multi-agent	Hybrid	Wakgra et al. (2024a)

Table 2.4 provides a comprehensive summary of the key components addressed in this dissertation, including the primary domain areas under investigation, the specific research problems explored, the DRL algorithms employed, the targeted QoS objectives, the types of agents involved, and the corresponding network contexts within which the proposed approaches are applied.

Chapter 3

RESEARCH METHODOLOGY

3.1 Introduction

In this chapter, research designs, methods and materials presented to deal with the problem and develop proposed system. The simulation experiment approaches were discussed to evaluate the application of DRL algorithms to optimize task offloading and resource allocation in a stochastic fog environment. The following subsection describes the research design and the data collection methods used in this study.

3.1.1 Research Design

In this dissertation, we follow an experimental research design approach. This approach allows for the creation of a controlled environment where variables such as resource availability, channel conditions, and task QoS requirements can be systematically modeled. The overall process is demonstrated in Figure 3.1, which shows the research design process used in this study. As noted in Davoli et al. (2022), experimental methods are essential for validating theoretical models in dynamic and complex systems like fog computing. Moreover, this experimental approach allows an evaluation of performance analysis between different DRL algorithms in a controlled environment. Experimental methods focus on quantifiable results in terms of task execution delay, energy consumption, and resource utilization. This approach provides precise and objective measurements of these metrics, enabling rigorous statistical analysis. As stated by Cavazzuti (2013), quantitative experimentation is essential for deriving actionable insights in optimization problems. The detail network settings with communication details are presented in Chapter 4 of Section 4.1, Section 4.2 and Section 4.2.1 This study is structured into five phases:

- Phase 1: Review of the literature and background analysis: conduct a comprehensive review of the existing literature on task offloading, resource allocation, and energy

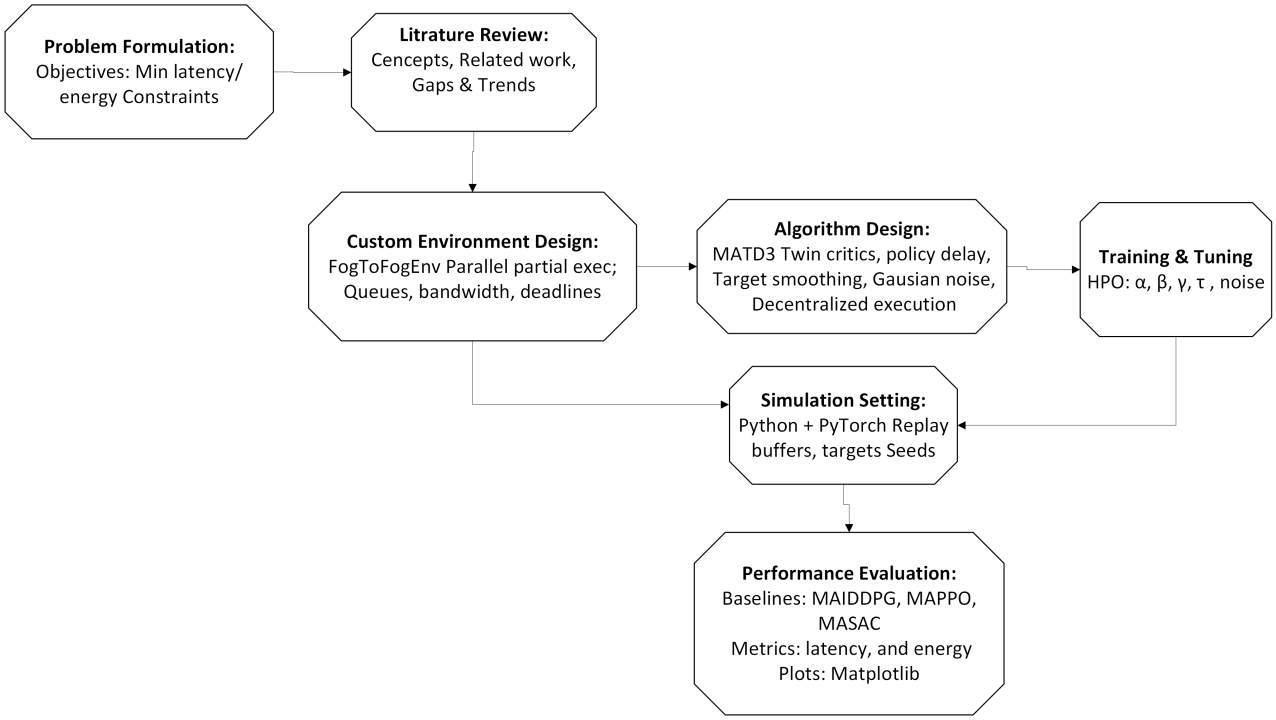

Figure 3.1: Research Design

optimization in fog computing. Exploration of existing task offloading techniques and MADRL-based optimization methods were conducted to establish a baseline for comparative evaluation. Analyze state-of-the-art DRL algorithms and their applications in distributed systems. This phase is critical to identify gaps in current research and define the scope of the proposed study. In addition, it ensures that the research is grounded in a solid theoretical foundation and aligns with the latest advancements in the field.

- Phase 2: Problem Formulation: Develop a mathematical model to represent an optimization problem in task offloading, resource allocation, and energy consumption, in dynamic fog environments

- Phase 3: System Modeling and Algorithm Development: Design and implement the proposed DRL-based algorithms using the baseline algorithms, such as Deep Deterministic Policy Gradient (DDPG), Soft Actor-Critic (SAC), Proximal Policy Optimization (PPO), and Twin Delayed DDPG (TD3) to optimize the latency of task transmission and execution delay along with energy consumption.

- Phase 4: Simulation and Data Collection: We conducted extensive simulations using the custom fog computing environment designed to evaluate the proposed algorithms.

- Phase 5: Analysis and Validation: collect data from agent-environment interaction

as replay buffer ($\mathcal{D}$) to analyze, validate, and compare the performance of the proposed TD3 based MAFCPTORA algorithms on top of the stated baseline DRL approaches.

3.1.2 Deep Reinforcement Learning model selection

The fog-to-fog (F2F) task offloading model is designed by considering continuous action space and with decentralized training and execution. DRL algorithms are investigated that works with this types of setting in literature Puliafito et al. (2019). To deal with this environment, DRL models such as SAC, Independent DDPG (IDDPG), PPO and TD3 were the candidate learning models. Additionally, F2F cooperation considers parallelization and decentralized nature, and its adaptability for continuous action space.

Table 3.1: DRL model comparison

Model	**Continuous action space**	**Pros**	**Cons**
SAC	✓	Handles dynamic environments	Complex to implement and resource intensive
PPO	✓	Useful for general purpose control	Sample efficiency problem for continuous control
IDDPG	✓	Works for decentralized learning	Does not support coordination among agents and often converge to a suboptimal policy
TD3	✓	Highly suitable for F2F task offloading with continuous action space in multi-agent setting	Slightly more complex than DDPG

Table 3.1 presents a DRL model comparison with respect to adaptability for continuous action space, applicability, and their limitations. The table shows all four DRL model works well for continuous action space settings. From decentralized training and execution perspective all models are fit with better adaptability in TD3. The stochastic nature of the fog resource makes the task offloading process dynamic, following a continuous action space with probabilistic transition. Research work such as Wakgra et al. (2024a) and Tadele et al. (2024) highlight the applications of TD3 to enable stability using replay buffer and to solve the problem of overestimation that face in DDPG for continuous

action space. TD3 become an effective option to model the proposed problem using delayed policy. We design the typical TD3 architecture with its original network model, with one input layer, two fully connected hidden layers with 256 neurons each and ReLU activation function, and one output layer for both actor and critic network. Additionally, this network model is designed with a batch size 64, an 10000 episode length of 100 steps per each, and a policy delay of 2.

Fujimoto et al. (2018) explain the advantages of using the rectified linear unit (ReLU) activation function in a continuous action space, particularly the TD3 network, to enable fast convergence and reliability. The gradient in ReLU, expressed in Equation (3.1), is designed to avoid vanishing gradient issues by allowing the TD3 to train critics and actor network deeper effectively with multiple layers.

$$\mathrm{ReLU}(x) = \max(0, x) \quad \Rightarrow \quad \frac{\partial\,\mathrm{ReLU}(x)}{\partial x} = \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{if } x \leq 0 \end{cases} \tag{3.1}$$

To encourage the agent to explore the new action and to ensure randomness and avoid being stuck in local minima, the Gaussian noise added to the network as presented in Equation(3.2).

$$\mathcal{N}(0, \sigma) \sim \frac{1}{\sqrt{2\pi}\sigma} \exp\left(- \frac{x^2}{2\sigma^2} \right) \tag{3.2}$$

- X: A random variable sampled from the distribution.
- σ: The standard deviation, which controls the magnitude of exploration.

3.1.3 Data Collection Method

In this work the important parameter data are collected from literature to frame the agents training experience from fog environments. To collect data from the literature, data extraction form is designed based on DRL algorithm, and QoS metrics such as energy consumption, latency, throughput, resource utilization, and fault tolerance.

Furthermore, the actual dataset was generated from the simulation experiments. The data extraction from a simulation using custom F2F network settings, the data frame designed as a replay buffer based on agents interaction with the environments. Figure 3.2 illustrates how DRL agents communicate with fog computing environments to monitor task and resource statuses, enabling informed offloading decisions. Task offloading and resource allocation problem are modeled as an MDP process to manage the resource state transition with continuous training and learning process of RL. The data collected from the fog environments are framed in the forms of (s_t, a_t, s_{t+1}, d_t).

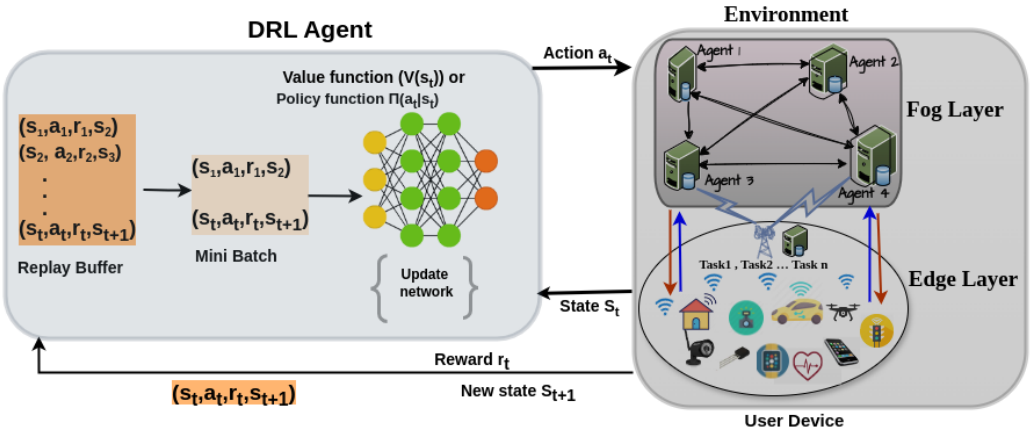

Figure 3.2: DRL-agent interaction with fog environments

- **State:** represents the current context or situation of the environment as perceived by the agent. It includes all relevant information needed to make a decision. In fog computing, a state may include variables such as CPU usage, bandwidth availability, energy level, task queue length, and network latency at a fog node.

- **Agent**: is the decision-making entity in a DRL system that interacts with the environment. It observes the current state, selects actions based on a learned policy, and updates its strategy over time to maximize long-term cumulative rewards. In fog computing, the agent is typically deployed at the fog node, where it learns when and where to offload tasks or allocate resources. In multi-agent environments, the agent interaction models are described as cooperative, adversarial, mixed (cooperative and competitive), hierarchical, learning based and communication-based (Zhang et al., 2021a; Le et al., 2024).

- **Action:** is defined as to execute the task locally and/or offload to fog nodes. Which means the selection of fog nodes and allocation of resources according to the agent's policy in the fog environments. The agents start the action by learning the network from their interactions based on resource status and task QoS requirements. Then, after the decision is made, either to execute locally or offload to fog nodes.

- **Reward:** in DRL can be positive and negative with respect to agents' decisions. Maximizing the positive reward by minimizing task completion time and enhancing the utilization of resources while executing the offloaded task. The negative reward is interpreted as the incurring of the maximum cost of resources, such as energy consumption and latency. The agent uses the value function and the policy function to calculate the reward from the agent's present observation and transition probability distribution of the proposed action. The goal of the agent is to maximize the reward using the local maximum of the policy $\pi^*(s)$ in each state of iteration.

Hence, DRL is formulated with MDP in terms of transition probabilities (P), rewards (r), the initial state of each fog node (f_i), and the transition state (s'). Figure 3.3

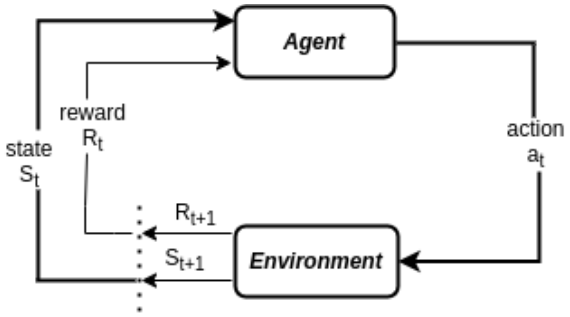

Figure 3.3: DRL Modeling in MDP architecture

models the sequential agent learning dynamics are formally framed with the MDP tuples (S, A, P, R) where :

- S is a set of states. $s_t \in S$ is the state in step t
- A is a set of actions. $a_t \in A$ is the action in step t
- $P(s'|s, a)$ is the state transition function indicates the probability that state s' happens after action a is taken in state s.
- $R(s, a, s')$ is the reward when action a is taken in state s, and the next state is s', i.e represented as s_{t+1}

The replay buffer is stored as an MDP table in the forms of action (a), state (s), reward (r), transition state at time t as (s_{t+1}) and done flag(d). Figure 3.4 shows the memory structure as replay buffer that stores agents past experience from the fog environment in the forms of MDP tuples (s_t, a_t, s_{t+1}, d_t).

The simulation setting network model designed with 12 state dimensions presented in Table 3.2, four (4) action in Table 3.1, and two 256 hidden layer.

Table 3.2: State (s) representation

State
Task size d_n
Deadline threshold
CPU utilization
Available Memory
Energy level
Priority
Bandwidth to F_m
Bandwidth to F_j
Transmission delay
Queue length F_m
Queue length F_j
Time step

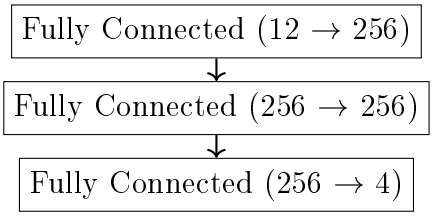

Table 3.3: Action (a) representation

Action
Execute locally F_m
Offload to F_j
Offload to neighbor F_j
Drop / Reject task

3.1.4 Data collection Procedure

According to Table 3.2, we choice TD3 to simulate the proposed algorithm. It works for decentralized training and execution, continuous action space and handle stochastic network environments. TD3 use two critic network, two target network, actor network and actor target network. The network model uses 256 × 256 neurons for actor and critic network. We use replay buffer batch size of 64 for each 100 episode to generate the pyTorch model (actor_agent0_ep100.pth, critic1_agent0_ep100.pth and critic2_agent0_ep100.pth). TD3 sample collection process follows these steps:

- The agent interacts with the fog environment to store and append each transition to replay buffer $\mathcal{D} \cup (s_t, a_t, s_{t+1}, d_t)$.
- This sampling process starts when the $\mathcal{D}$ has enough samples, i.e., when the buffer size is greater than the batch size.
- Randomly select mini-batch $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$ uniformly without replacement
- Use batch for critic and actor update such that the critic minimize TD3 error as defined by Equation (3.3).

$$y = r + \gamma(1 - d) \min_{i=1,2} Q_{\phi_i}^{\text{target}}(s', \tilde{a}') \tag{3.3}$$

  where γ, $Q_{\phi_i}^{\text{target}}$, and $\tilde{a}'$, are the discount factor, i-th target critic network, and target policy action with noise, respectively.

The problem in RL, these consecutive samples are highly correlated because the agent collects in a sequential step of action from the fog environments. However, a Deep Neural

Network (DNN) assumes the training data is independent and identically distributed. Therefore, the network uses a replay buffer to reshuffle the experience, breaking the correlation and resolving the overfitting problem of recent experience. In addition, the mini-batch is a small batch of experience randomly sampled at each update step from the replay buffer. This mini-batch stabilizes the gradient update and reduces computational cost since the average over multiple experiences is used instead of the entire replay buffer. The mini-batch sampled at each 256 transition of the update steps compared with the batch size of network model.

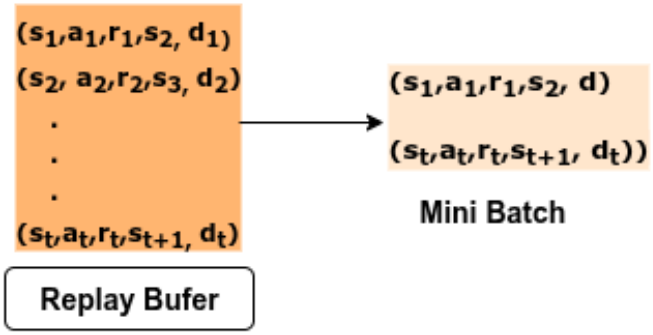

Figure 3.4: Replay buffer and mini-batch data frame

Table 3.4: Sample data collection frame for TD3-based fog simulation

epi_id	step	agent_id	state	action	reward	ne_state	done	latency	energy
12	3	agent3	[0.2, 0.5, 0.7]	[0.3, 0.4]	0.36	[0.25, 0.45, 0.65]	0	0.45	0.12

At each episode in the training stores metrics presented as episode_number, total_reward, average_latency, average_energy, and successful_tasks as presented in table 3.5.

Table 3.5: Sample training metrics

episode_number	total_reward	average_latency	average_energy	successful_tasks
1	0.36	0.17	0.80	125

The sample evaluation data frame trajectory without exploration can be represented in table 3.6

Table 3.6: Sample evaluation metrics

time_step	state	action	reward	latency	energy	done
10	$[0.2, 0.5, 0.7]$	$[0.3, 0.4]$	0.35	0.2	0.91	true or false

Chapter 4

PROPOSED MULTI-AGENT DRL-BASED PARTIAL TASK OFFLOADING MODEL IN FOG-TO-FOG NETWORKS

4.1 Introduction

In this chapter, we present our proposed multi-agent fully cooperative partial task offloading and resource allocation in fog computing (MAFCPTORA), which addresses the challenges mentioned 1-4 in Section 1.8. The proposed system aims to minimize the overall system latency and energy consumption, improve response time, and optimize energy efficiency. This minimization problem is designed into three stages: modeling the DRL problem in MDP tuple, designing offloading decision, and resource allocation algorithms.

We consider the resource and workload status of fog nodes to optimize offloading and resource allocation decisions in stochastic fog environments. In this context, a continuous action space is considered to compute the task, indicating that the task execution approach follows a partially offloaded approach where the offloading action is between 0 and 1, such that the probability of offloading P_j sub-task ($k_{n,j}$), defined in the range ($0 < p_j < 1$) and executed locally as ($1 - \sum_{j=1}^{n} k_{n,j}$). The objective function is defined in terms of computation and communication model with minimizing task execution delay and energy consumption to complete particular tasks K_n and the sum of sub-tasks as $k_1, k_2, \ldots K_n$. Mainly, the task execution delay and transmission energy consumption are formulated for the local execution of the task at F_m and offloading to F_j from F_m to F_j, as well as the retransmission delay of the processed task ($k_{n,j}$) from F_j back to F_m, is considered. In addition, the offloading decision is based on continuous action

where the task computation decisions $k_(m, j)$ at time t are determined by probability ratio.

In Section 4.2, we describe the details of our proposed two-layer network model, task model, communication model, computation model, and formulation of the problem. For the purpose of this study, we assume reliable fog nodes and communication links. Then, in Section 4.2.1, introduce an illustrative class of task offloading mechanisms that our proposed mechanism approach follows. Next, we present the task model in Section 4.3, dynamic task partitioning in Section 4.4 discussed.

In this chapter, (Section 4.6) such that local computation delay (T_l), and energy consumption (E_l) as presented in Eq. 4.11 and 4.12, respectively. Furthermore, the offloading model considers communication and energy consumption in (Section 4.5, execution, and queue delays along with energy consumption in (Section 4.7), as modeled in Eq.4.17 and 4.19, respectively. After that, Algorithms of benchmark mechanisms are discussed in Section 4.10 along with properties discussed. To address this research problem, this dissertation develops the following solution: a multi-agent DRL algorithm for task offloading decisions and resource allocation, considering latency and energy consumption in fog computing environments. Furthermore, to represent the stochastic nature of the task and the fog resource, we model the problem as a MDP tuples (s, a, r, and s_{t+1}), defined as state, action, reward, and next state.

In summary, task offloading in F2F scenario is designed as a probability-based partial offloading. Communication latency, computation latency, and energy consumption were modeled as a multi-objective minimization problem. Perform performance analysis and comparison with the current offloading decision and resource allocation DRL baselines are conducted. We present Table 4.1 that shows the lists of notation used throughout this dissertation.

Table 4.1: List of key parameters.

Notation	Description
F_n	Number of fog nodes
U_n	Number of user devices
$M_{f,n}$	Available memory at each fog node
C_f	Available CPU at each fog node
$\tau_{m,j}$	Task processing delay
$\tau^o_{m,j}$	Task offloading delay
β	Transmission bandwidth
E_k	Energy consumption of task k_n
ζ	Battery life time
H_c	Channel condition
φ	Number of channel available
k_n	Task to be executed
V_k	Total size of the task k_n
H_i	Channel transmission power
τ_{max}	Maximum tolerable delay
Q	Waiting tasks in each fog node
S_i	State of each agent
$O(t)$	Observation of environment at time t
a_i	Individual agent action
X_n	Offloading decision
$R_{f,k}(t)$	Resource allocation decision at time t
μ and $\mu\hat{*}$	Optimal and target policy
r_t and $\tilde{r}_t$	Individual and collective average reward

4.2 Network Model

We model a multi-user, multi-fog environment (as illustrated in Figure 4.2) to handle diverse service demands. The network features multiple heterogeneous and resource-constraint user devices (U_n), where $n = 1, 2, \ldots, N$) connected wirelessly to a hierarchical fog

layer (F_n). There is also a set of $F = F_1, F_2, \cdots, F_n$ heterogeneous fog nodes, where $M \leq N$. Each fog node maintains its resource status and shares updates with its neighbors, enabling collaborative task offloading and resource management. Moreover, their processing capacity increases from the bottom to the top layer, whereas energy consumption and latency also increase.

Our system model incorporates a hybrid two-stage task execution process: User-to-Fog (U2F) and Fog-to-Fog (F2F). In this model, task execution proceeds in two phases: In the U2F phase, a task originating from a user device (U_n) is offloaded to a fog node F_m (the task-owning node, $m \in 1, 2, \ldots, N$). F_m is considered to minimize task transmission delay and response time. We refer to the task execution on F_m as local computation. The second phase, the F2F phase, is invoked in the event that the execution of a task locally at F_m may not guarantee QoS requirements. In this case, we employ a horizontal distributed F2F cooperation mechanism following the approach in Tran-Dang and Kim (2023) depicted in Figure 4.1. In this case, tasks are redistributed from F_m to an adjacent node F_j ($j \in 1, 2, \ldots, N$) for further processing or collaborative execution. Figure 4.2 illustrates the task offloading and execution scenario.

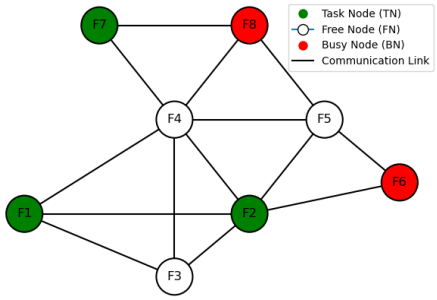

Figure 4.1: F2F cooperation topology

Figure 4.1 shows the cooperation of $F_{m,j}$ in the form of task node (TN), free node (FN), and busy node (BN) states; each state indicated as three TNs (F1, F2, F7), two BN(F6, F8), and three FN (F3, F4, F5) adopted from Tran-Dang and Kim (2023). In this case, all fog nodes are assumed to have the same protocol running independently to update their status.

In this fog layer, task allocation to adjacent fog nodes (F_j) is governed by an offloading probability ratio. This ratio is dynamically determined by factors such as the available resources of each F_j and the requirements of the user's task, including its computation deadline and priority. Within this cooperative peer-to-peer (F2F) framework, each fog node (F_m and its neighbors F_j) can make independent offloading decisions. Consequently, the set of tasks to be offloaded to a specific F_j is denoted as ($K_{n,j} = 1, 2, \ldots, n$). To analyze the offloading decision, the time step (decision episodes) is presented as discrete time ($T = 1, 2, 3, \ldots, n$).

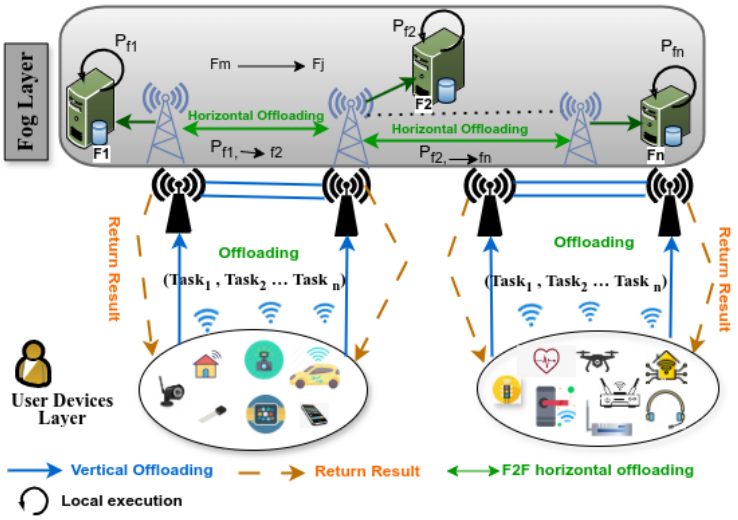

Figure 4.2: User device-Fog Architecture.

4.2.1 Illustration of task offloading decision procedure

The task offloading flowchart for a fog computing environment with numerous user devices and fog nodes is shown in Figure 4.3. Multiple user devices U_n start the process, each of which generates a set of tasks $K_1, K_2, \ldots, K_n$. The user device U_n offloads each task K_n to a specific fog node F_m so that it can be executed. The fog node F_m first determines if it has enough resources to complete K_n by the maximum permitted deadline $T_{\max}$ after receiving a task. The task is carried out at F_m and the outcome is sent back to the original user device U_n if both requirements are met. The system considers different execution strategies in the process that F_m does not have enough resources or is unable to meet the deadline constraint. It first determines whether a nearby fog node F_j is available and capable of completing task K_n by the deadline. The task is fully offloaded to F_j if such a node is present, and the outcome is returned to F_m before being sent to the user device U_n.

In cases where no single adjacent fog node can meet the requirements, the task K_n is split into sub-tasks (partial offloading), which are then distributed to multiple fog nodes $F_1, F_2, \ldots, F_p$ for parallel execution. Each fog node executes its assigned sub-task, and the results are aggregated and returned to the original fog node F_m, which then forwards the final output to the user device U_n.

If no suitable fog nodes are found and the system cannot meet the deadline or resource constraints through either full or partial offloading, the task K_n is dropped. The flowchart also includes feedback loops indicating dynamic task reassessment or fallback to earlier stages based on execution outcomes. The hierarchical and adaptive task offloading approach depicted in this flowchart is ideal for dynamic fog computing environments, as it enables distributed partial offloading, fallback delegation, and direct execution. Furthermore, by monitoring system states, resource availability, task deadlines, and

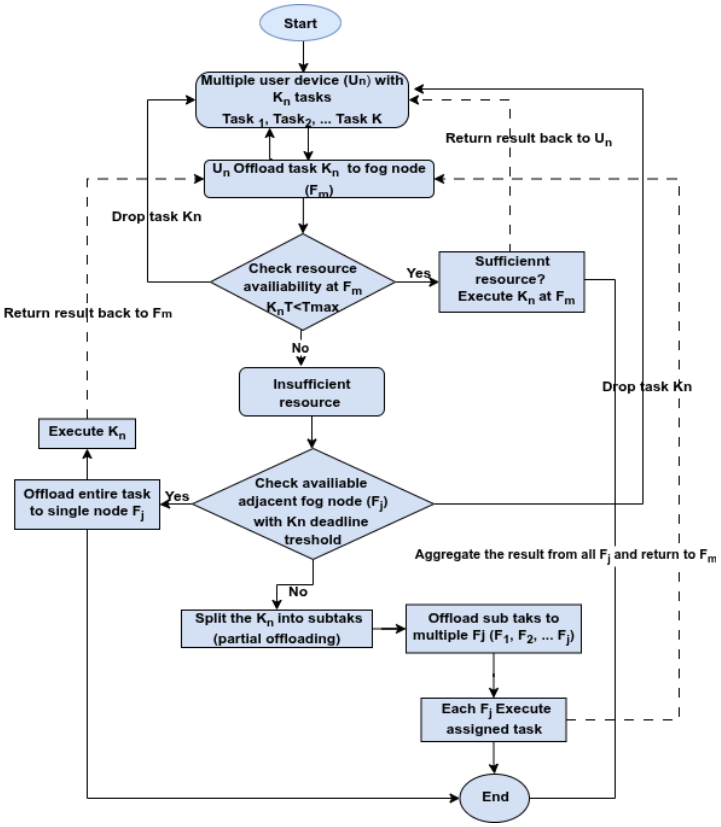

Figure 4.3: Task offloading flow chart

performance results over time, agents can learn optimal strategies for DRL-based task offloading.

From a decision-making perspective, user devices typically begin by selecting the nearest or least-loaded fog node for task offloading. The selected fog node then allocates available resources to execute the task and returns the result to the user device. If the fog node lacks sufficient computational capacity or is overloaded, it forwards the task to the cloud for execution. This multi-tier processing model requires intelligent, adaptive decision-making mechanisms capable of learning from the environment to optimize offloading paths and scheduling. As such, the design of learning algorithms for task offloading and resource allocation must be tightly coupled with the architectural setup. For instance, reinforcement learning agents must be trained with awareness of the hierarchical structure (fog vs. cloud), available bandwidth, node availability, and task urgency. The dynamic interplay between architecture and learning model requires that offloading decisions consider factors such as network latency, fog node capacity, task deadlines, and energy constraints.

4.3 Task Model

We consider multiple tasks initiated from U_ns denoted as K_n, where $\{n = 1, 2, \ldots, N\}$. The tasks can be represented as a directed acyclic graph (DAG) shown in Figure 4.4.

The task profile of the nth user application can be defined in Equation (4.1):

$$K_n = \{V_k, C_n, \tau_{max}\} \tag{4.1}$$

where V_k, C_n, and τ_{max} denote the task data size, the CPU cycles required to compute the task, and the maximum tolerable deadline (expired time) of the task K_n, respectively. The task computation (K_n) at time (t) is defined by the total size of the task (V_k) and the intensity of the device computation $b_u(t)$ at time (t_i).

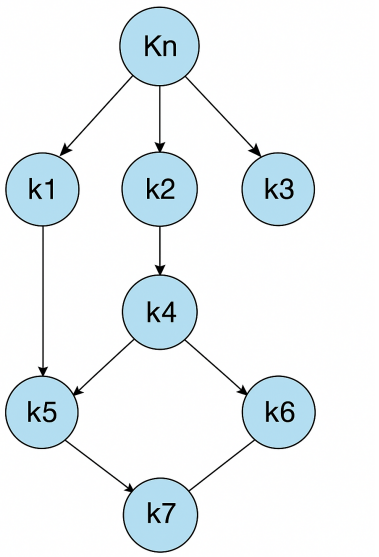

Figure 4.4: Directed acyclic graph task model.

The DAG task in Figure 4.4 demonstrates parallel task execution enabled by task dependencies. Taking task K_n, composed of sub-tasks $k_1, k_2, k_3, k_4, k_5, k_6, k_7$, as an example, k_1, k_2, and k_3 can run in parallel, k_4 wait for execution of k_2, then k_5 and k_6 waits for execution of k_4, while k_7 is dependent on k_5 and k_6 and executes only after both are finished.

4.4 Dynamic Task Partitioning

In our F2F task offloading mechanism, dynamic task partitioning is employed. A DRL agent learns to partition tasks effectively by observing real-time system conditions such as resource availability, task requirements, network bandwidth, and computational cost. For partial offloading, where tasks are distributed across multiple adjacent fog nodes, each fog node F_j is assigned an offloading probability ratio p_j, where $(0 < p_j < 1)$. The complete probability distribution for a task (k_n) being offloaded from F_m to its neighbors is represented by the vector $P = (p_1, p_2, \ldots, p_n + 1)$. Where $p_n(t) = p_n$ and $p_n(t) = 0$ if node f_m processes all the arrived tasks k_n locally, while $F_m(t) = p_n + 1$ and $p_n(t)\#0$ implies that the node F_m offloads the number $p_n(t)$ of k_n to adjacent f_j. This probability

vector guides the decision-making process for distributing portions of the task to available F_j nodes, considering their computational capabilities, the network latency between F_m and each F_j, and their current resource availability. Equation (4.2), presents the task offloading probability ratio for each F_j:

$$p_j = \frac{k_{n,j}}{\sum_{j=1}^{n} k_{n,j}} \tag{4.2}$$

where $\sum_{j=1}^{n} k_{n,j}$ defines the sum of tasks offloaded to adjacent F_j and Equation (4.3), for task execution by f_j.

$$k_{n,j} = p_j \times k_n \tag{4.3}$$

For the local task execution, the remaining partition to be executed at the source (f_m) is defined as in Equation (4.4):

$$P_m = 1 - \sum_{j=1}^{n} p_j \tag{4.4}$$

where k_n indicates the total number of tasks K being offloaded and defined as ($k_n = \sum_{j=1}^{n} k_{n,j}$)). With this, the total probability ratio of all F_j must ensure Equation (4.5):

$$\sum_{j=1}^{n} p_m + p_j = 1 \tag{4.5}$$

This ensures that the sum of all partial offloading ratios forms a valid probability distribution in a continuous action space, where each $x_{m,j} \in (0, 1)$.

Following this partitioning scheme, task offloading proceeds dynamically among horizontally connected fog nodes. To meet task execution requirements under fluctuating conditions, an overloaded fog node (F_m) transfers tasks to less utilized or idle nodes (F_j). Figure 4.2 illustrates this dynamic task partitioning and offloading process.

4.5 Communication Model

The communication model in our proposed network architecture encompasses two types of communication: User-to-Fog (U2F) involving continuous communication between user devices ($U2F$) and fog nodes (F_m), and Fog-to-Fog (F2F), specifically between fog nodes F_m and F_j. This section details these communication aspects.

4.5.1 U2F Communication

Following the model in Zhang et al. (2023), wireless communication between user devices (U_n) and a fog node (F_m) is subject to multi-path fading and path loss. The probability of path loss ($PL(d_{uf})$) considers both line-of-sight (LoS) and non-line-of-sight (NLoS)

conditions. The path loss PL between U_n and the neighboring fog node F_m is adopted from Tran-Dang and Kim (2023), and Hao et al. (2024). To aid offloading decisions, the channel status (H_c) is classified as busy (1) or idle (0). Therefore, the path loss (η) is presented as follows at Equation (4.6) and (4.7).

$$PL^t_{(d_{uf})} = \sigma(d_{uf}) \times PL_{LoS}(d_{uf})(1 - \sigma(d_{uf})) \times PL_{NLoS}(d_{uf}) \tag{4.6}$$

where $\sigma(d_{uf})$, $PL_{LoS}(d_{uf})$, and $PL_{NLoS}(d_{uf})$ represent LoS probability, LoS path loss $\eta 1$, and NLoS path loss $\eta 2$, respectively.

$$P_L = 20\log(d_{\mathrm{km}}) + 20\log(B_{\mathrm{kHz}}) + 32.45\,(\mathrm{dB}) \tag{4.7}$$

The task transmission delay τ^t_{tr} is measured based on the bandwidth of the channel, the size of the offloaded task K_n, the transmission power of the user device's channel, and the distance to the F_m.

$$\begin{aligned} T^t_{off_k} &= \tfrac{K^t_n d_{m,f}}{\tau^t_{tr}}; \\ \tau^t_{tr} &= \beta^t_{u,f} \cdot \log\left(1 + \tfrac{h^t_{u,f} \cdot \zeta^t_{u,f}}{\sigma}\right) \end{aligned} \tag{4.8}$$

We model the channel transmission rate τ^t_{tr} from node f_m to node f_j according to Sellami et al. (2022), where $\beta^t_{u,f}$, $h^t_{u,f}$, $\zeta^t_{u,f}$, and σ are the bandwidth, the channel power gain, the transmission power, and the standard deviation of the noise power gain, respectively. Therefore, the sum of tasks offloaded to adjacent F_j, at the time, is guided by Equation (4.8). We assume that the maximum available transmission power at a fog node corresponds to the power required for offloading the maximum number of tasks as presented in Baek and Kaddoum (2021a). Accordingly, the node is capable of transmitting p^t_j tasks within a single time slot, such that the transmission energy consumption does not exceed the maximum permissible value $\zeta^t_{f,(tr)} \leq \zeta^{(max)}_{f,(tr)}$. The signal-to-noise ratio (SNR) is described as $SNR = \frac{P_{signal}}{P_{noise}}$. The noise power ratio and channel capacity correlations are designed according to Shannon's Law by Equation (4.9) to present the capacity-dependent relationship in the wireless network.

$$C = \beta log_2(1 + (S/N)) \tag{4.9}$$

where C, β, S, N are the channel capacity, channel bandwidth, average received signal power, and average noise power, respectively.

4.5.2 F2F Communication

Fog-to-Fog (F2F) communication is triggered when the originating fog node F_m cannot satisfy the quality of service (QoS) demands of a task from user U_n. Subsequently,

a direct offloading process occurs between F_m and neighboring fog nodes F_j to enable parallel execution of the partitioned sub-task on F_j. Consequently, the transmission and computation delays within these fog layers are considered in our analysis. The communication network connecting fog nodes F_m and F_j is modeled as a flat, wired, fiber, optic Ethernet-based infrastructure. Therefore, the communication cost is evaluated based on transmission delay and propagation delay, both of which depend on the distance between the nodes and the speed of light. The propagation delay between F_m and F_j is calculated as $P_{m,j} = \frac{d_{m,j}}{C_l}$, where $d_{m,j}$ represents the distance between F_m and F_j, and C_l is the speed of light. For task offloading between fog nodes, we assume a negligible noise ratio.

The energy consumed during the transmission of task portions K_n between fog nodes (F_m to F_j) in partial offloading accounts for both transmission and reception energy as indicated Equation (4.10). This consumption is influenced by the channel transmission power, the distance between the nodes, the size of the transferred data, and the energy model of the communication interface.

$$E^{tr}_{m\to j}(t) = E^{trans}_{m\to j} + E^{rec}_{m\to j} \tag{4.10}$$

where $E^{trans}_{m\to j}$ is $\zeta_{trans} \times T_{trans}$ and $E^{rec}_{m\to j}$ is $\zeta_{rec} \times T_{rec}$, where ζ and T are power and time consumed during transmission and receiving data, respectively. The transmission power utilized by fog node f is calculated based on Equation (4.11) and (4.12) as follows

$$\zeta^t_{n,(tr)} = \frac{\beta^t_{m,j^t_i} \cdot \sigma^2}{\eta 1 d_{m,j^t_i} - \eta 2} \left(\frac{k_n p^t_j}{2^{T\beta(m,f^t_n)}} - 1 \right) \tag{4.11}$$

Moreover, the total data transmission rate d_{k_n} for the task k_n over a set of multiple orthogonal channels from F_m is calculated with Shannon's capacity formula (Shannon (1948)) as follows.

$$\tau^t_{tr} total = \sum_{t=1}^{T} \sum_{n=1}^{N} \beta_n(t) \log_2 \left(1 + \frac{\zeta^t_{n,(tr)} h_n(t)}{\sigma} \right) \tag{4.12}$$

where $\beta_n(t)$, $h_n(t)$, $\zeta^t_{n,(tr)}$, and σ represent the nth channel bandwidth, channel gain, allocated transmission power, and noise power spectral density, respectively.

The transmission and communication delay between (F_m) and (F_j), for each individual sub-task, can be calculated Equation (4.13)

$$\tau^o_{m,j} = \frac{k_{n,j} \times \tau^t_{tr} total}{\beta_{m,j}} + P_d \tag{4.13}$$

where P_d and $\beta_{m,j}$ are the propagation delay and the achievable data transmission rate between F_m and F_j, respectively. Therefore, the total sub-task transmission latency can

be computed using Equation (4.14) as the average of $T^o_{m,j}$ such that

$$\tau^o_{total} = \frac{1}{n}\sum_{t=1}^{T}\sum_{j=1}^{N}\tau^o_{m,j} \tag{4.14}$$

where, n is the number of sub-tasks $k_{n,j}$ offloaded to f_j.

4.6 Local Computations

Assume $T^l_{F_m}$, V^l_k, and C_n denote the local computing delay to compute the task portion of K_n, the task size at F_m, and the number of CPU cores, respectively. Therefore, the delay of tasks executed locally at f_m and offloaded from U_n can be expressed by Equation (4.15)

$$\tau^l_{f_m} = \sum_{t=1}^{T}\sum_{m=1}^{N}\frac{V^l_k}{C_n}\cdot p_m \tag{4.15}$$

where C_n depends on the total task size V_k and $b_u(t)$ to measure the required intensity of computational effect at the single node to complete its assigned portion. Energy is not a major concern in the fog layer; energy consumption is expressed in Equation (4.16) measured per CPU cycle ζ_n, and computing tasks K_n are considered to calculate energy as follows.

$$E^l_{F_m} = \sum_{t=1}^{T}\sum_{m=1}^{N}(\zeta_n \times K^e_n) \tag{4.16}$$

4.7 Offloading Computation Delay

As highlighted before, a fog node (F_m) accepts full tasks from U_n until its maximum tolerable capacity and stops accepting when its workload queue is full. Then, F_m offloads the task to the cooperating adjacent (F_j) node when it is unable to achieve the objective of the task's maximum tolerable thresholds. The computation delay for each F_j is computed as follows in Equation (4.17):

$$\tau^o_{c,j} = \frac{k_{n,j} \times V^o_{k_n}}{C_j} \tag{4.17}$$

where V_{k_n} represents the total size of task k_n and C_j denotes the maximum number of CPU cores available on fog node F_j. Following the parallel task execution model presented in Tran-Dang and Kim (2023), we consider scenarios where multiple sub-tasks are executed concurrently across several fog nodes. Therefore, the overall task computation delay is determined by the longest execution time among all its sub-tasks computed by Equation (4.18).

$$\tau^o_{C,max} = max(\tau^o_{C,j_1}, \tau^o_{C,j_2}, \ldots, \tau^o_{C,j_n}) \tag{4.18}$$

In this case, Equation (4.19), the total delay $\tau^t_{m,j}$in F_m and F_j is the total transmission latency τ^o_{total} plus task execution latency $\tau^o_{c,max}$ and local computation delay $\tau^l_{f_m}$.

$$\tau^t_{m,j} = \tau^o_{total} + \tau^o_{c,max} + \tau^l_{f_m} \tag{4.19}$$

We note that our work primarily focuses on task offloading between fog nodes, assuming that the user device has already offloaded the task to an initial fog node F_m. Therefore, we assume that tasks in the fog node queue are handled on a first-come, first-served (FCFS) manner without accounting for contention dynamics or task prioritization.

4.8 Energy Consumption

The energy consumption for each F_j can be calculated as follows using Equation (4.20).

$$E^o_{f_j} = \zeta_n \times k^e_{n,j} \tag{4.20}$$

Meanwhile, the total task execution energy consumption when tasks are executed in multiple F_js is given by Equation (4.21):

$$E_{j,\text{total}} = \sum_{t=1}^{T} \sum_{j=1}^{N} E^o_{f_j} \tag{4.21}$$

Since the partial offloading model is applied in this work, the total system-wise task computation delay T_{total} of the task considers both F_m and F_j as indicated in Equation (4.22).

$$\tau_{total} = \tau^l_{F_m} + \tau^o_{F_j} \tag{4.22}$$

Moreover, in the partial task offloading approach, the total energy consumption E_{total} considers both task execution at local F_m and adjacent F_j as well as energy consumption at the transmission of task K_n. Therefore, E_{total} is defined at Equation (4.23) as follows:

$$E^t_{m,j} = E^l_{F_m} + E_{j,\text{total}} + E^{tr}_{m,j} \tag{4.23}$$

4.9 Multi-objective problem formulation

We now formulate the optimization problem with the aim of minimizing the overall system latency and energy consumption in Equation (4.24a), thereby improving response time and optimizing energy efficiency as follows:

$$\min_{p_n} \lim_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} \sum_{j=1}^{N} \left(\omega_t \tau^t_{m,j} + \omega_e E^t_{m,j}\right) \tag{4.24a}$$

subject to

$$C1 : p_m + \sum_{j=1}^{n} p_j = 1 \tag{4.24b}$$

$$C2 : 0 \leq p_m, \left\{ \sum_{j=1}^{n} p_{j,1}, \ldots p_{j,n} \right\} \leq 1, \quad \forall m, j \in F \tag{4.24c}$$

$$C3 : \quad \sum_{t=1}^{T} \sum_{n=1}^{N} p_j \times k_{n,j} \leq C_j, \quad \forall j \in F, \forall k \in K \tag{4.24d}$$

$$C4 : \quad \tau_{total} \leq \tau_{max} K_n, \quad \forall n \in N \tag{4.24e}$$

$$C5 : \quad E_{m,j}^{exc} \times K_n \leq E_{threshold}, \quad \forall n \in N \tag{4.24f}$$

$$C6 : \quad \sum_{k=1}^{N} \sum_{p=0}^{1} K_n \leq K_{n,\max}, \quad \forall m \in M \tag{4.24g}$$

where N, T, (ω_t), and (ω_e) are the number of fog nodes, time steps, and weight coefficients of computation delay and energy consumption for both local and offloading cases. The values of (ω_t) and (ω_e) are within the interval $[0, 1]$ and satisfy the constraint $(\omega_t) + (\omega_e) = 1$. The objective function is formulated to ensure that a user device offloads a task to a specific fog node in a particular time step t. In this optimization problem the constraints are presented:

- $C1$ (4.24b) defines the total task execution ratio between f_m and multiple f_js should not exceed the maximum (i.e., $\leq$1).
- $C2$ (4.24c) defines the probability of offloading ratio ranges from (0) up to 1
- $C3$ (4.24d) ensures that the size of the offloaded task to a fog node should not exceed the node's maximum capacity.
- $C4$ (4.24e) shows the latency constraints concerning the maximum tolerable delay threshold (τ_{max}) of offloaded tasks in terms of transmission and execution deadlines,
- $C5$ 4.24f) verifies that the available energy in the fog node remains above a specified threshold.
- $C6$ 4.24g) the sum of the offloaded tasks should not exceed the maximum task.

This problem is in the class of NP-hard problems Ghanavati et al. (2020a). Therefore, the discussion of the proposed approach presented in Section 4.10.

From network historical data to train the DL-based model and replay buffer ($\mathcal{D}$) data for the actor-critic RL network to predict resources, such as bandwidth, channel condition, task queue, memory, and others, for the future state prediction. This MADRL model recognizes the availability and utilization of resources as a complex optimization problem

with dependencies and interactions among multiple variables. The dynamic network and resource context create a problem for the agent to converge under optimal conditions. According to Al-Khafajiy et al. (2019a), in F2F cooperation, they share their updated state and maintain a dynamically updated list of the best nodes, as framed in Table 4.2.

Table 4.2: Cooperative $F2F$ sample extracted knowledge frame.

FID	**Status**	**CPUCore**	**CPU(%)**	**Memory(%)**	β	**Queue**	$\#K_n$	**Next Hop**
F_1	Busy	8	70%	50%	10Mbps	$[K_1, K_2, K_3]$	10	F_2

The agent partially observes local states such as workload, available resources, energy status, latency to other fog nodes, QoS metrics, task nature, node status (active, idle, or sleep), and performance metrics from historical data from each fog node. According to Tran-Dang and Kim (2023), this time-varying context is reflected in the topology designed in Figure 4.1 for interactions of the user device, and the status of the fog node can be characterized in terms of distance, resources, and task computation ability. Therefore, MAFCPTORA is designed to maximize the overall return accumulated over time, rather than the immediate reward associated with a particular decision.

4.10 Multi-agent TD3 based task offloading architecture

The task offloading problem in Equation (4.24a) is fundamentally modeled as a Markov decision process (MDP) as presented in Hao et al. (2024), implying that the next state depends only on the current state and the chosen action. In more realistic scenarios where the agent only has a limited view, the problem is framed as a Partially Observable MDP (POMDP). This context affects the model's decisions and long-term performance because it focuses on the immediate maximum value. To address this scenario, the decentralized partially observable MDPs (Dec-POMDP) approach was applied to a multi-agent setting as depicted in Figure 4.5. For this, a multi-agent system task offloading decision-making process involves independent TD3 agents deployed in each fog node. These agents observe the environment and make their decisions based on their observations. The environment's state includes dynamic factors relevant to decision-making. We apply decentralized training to allow agents to learn their policies in a distributed manner, reducing the burden on a single central entity and often enhancing privacy and scalability.

Each agent typically only partially observes local states, which include information like workload, available resources, energy status, latency to other fog nodes, QoS metrics, task nature, node status (active, idle, or sleep), performance metrics from historical data, channel condition, location, and distance between fog nodes.

In cooperative multi-agent environments, policies aim to optimize a joint objective,

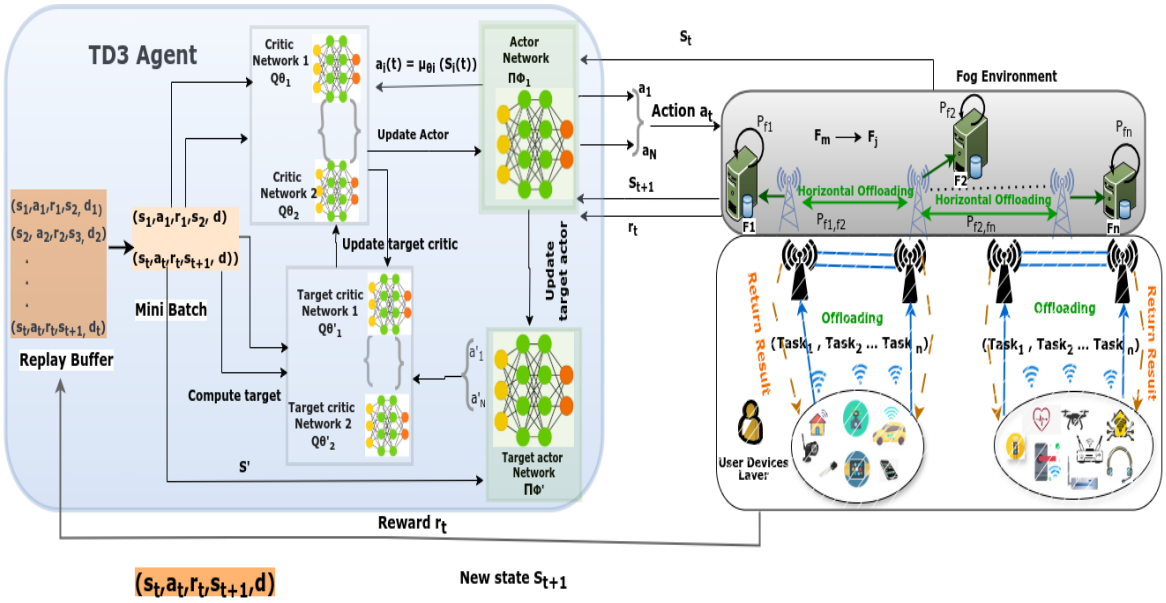

Figure 4.5: MATD3-based horizontal F2F task offloading architecture.

often framed under Dec-POMDPs. These systems pose challenges such as an exponentially growing joint state-action space as the number of agents increases and non-stationarity due to dynamic network and resource availability Raju et al. (2024). These dynamics are formally framed with the MDP tuples (s, a, s', r, d) where:

- State Space(s): In this task offloading problem, referring to the state of the fog (F) at time step (t) observed by the agent from the environment. It includes the task node (F_m), resource status, and channel condition, modeled as a set of $s_t = \{F_t, RS_t\}$, $\forall s_t \in S$, where F_t represents a set of fog nodes and RS_t a set of resources at time step t.

- Action Space (a): the MATD3 agents observe the state and make a decision of action (a_t) based on partial offloading. The agent's decisions are based on partial observability of the system state. We consider a continuous action space where some tasks are executed at F_m, and the remaining are offloaded to an adjacent fog node F_j, $(F_m \rightarrow F_j)$ as a set of actions $a_t \in A$ is the action in time step t. The set of actions at each time step t is defined as $(p_0^0, p_n^1, \ldots, p_0^R), \ldots (p_n^0, p_n^1, \ldots, p_n^R)$, where p_n^R represents the ratio of the task n allocated to available resource R.

- Reward (r): $r(s, a, s')$ is the reward when action a is taken in state s, and the next state is s'. Where the reward is determined with the state transition function $(s'|s, a)$ with the probability that state (s') happens after action (a) is taken in state (s).

- Done (d): Whether the TD3 agent in the fog environment reaches the terminal state or not is communicated by this element. In this environment, the agent knows that

the termination occurs after a hundred offloading decisions or if an offloading action causes F_n resources to enter a busy state.

Figure 4.6 illustrates the dynamics of fog nodes' state and action status with a sample probability ratio that shows the stochastic nature of the network and its resources. A multi-user (U_n) and multi-fog ($F_{m,j}$) network environment is considered to simulate

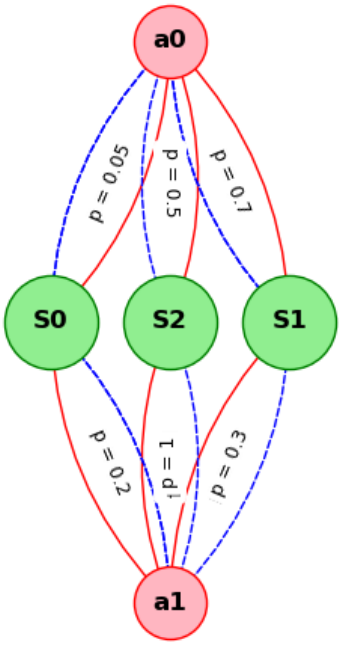

Figure 4.6: The dynamics of fog node state and action status

decentralized training, and the agent computes tasks in a distributed manner. In the MADRL setting, the agent explores the environment and stores the replay buffer ($\mathcal{D}$) experience as an MDP table (s_t, a_t, r_t, s_{t+1}) to train the target network. The current state of the agent is updated with real-time data observed from the environment. After that, the individual agent $F_{m,j}$ can make independent task offloading decisions for the collective result. Depending on the reward design, policies can be trained using either local or collective feedback. Practically, the agent learns the task offloading policy from the network and observes the real-time task and resource information to make an online decision. A decentralized training and execution are used so that each agent updates its actor based on observation for joint actions of agents to enhance collective reward.

The agent's primary goal is to maximize the overall return accumulated over time, rather than focusing solely on immediate rewards. The collective reward function is often structured to penalize (use negative values for) metrics like latency and energy consumption. In this case, to deal with the instability of the data set, the Min-Max

normalization is applied to stabilize reward variation as illustrated by Equation (4.25).

$$x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{4.25}$$

Among them, x_{norm} is the normalized data, x is the original data, and x_{max} and x_{min} are the maximum and the minimum values in the replay buffer data set ($\mathcal{D}$), respectively. The TD3 agent is utilized as an off-policy algorithm with a deterministic policy $\mu(s;\theta)$ and twin Q-value functions $Q_1(s,a;w_1)$ and $Q_2(s,a;w_2)$ to estimate the expected return. The goal of the reward design is to guide agents toward minimizing normalized latency and energy consumption during task offloading. This is represented using Equation (4.26) as follows

$$r_{m,j} = -(\omega_t \cdot T_{normal} + \omega_e \cdot E_{\text{normal}}) + r_{\text{penalty}} \tag{4.26}$$

In a Markov decision process (MDP) setting, Equation (4.27) is used to calculate the cumulative reward at time (t) as:

$$r_t = \sum_{k=0}^{N} \gamma^k r_{t+k+1} \tag{4.27}$$

where ($\gamma \in [0,1]$) is the discount factor that determines the influence of future rewards. Due to the stochastic nature of the environment, a shaped reward is computed in Equation (4.28) using an exponential moving average to ensure stable learning and consistent offloading decisions:

$$\tilde{r}_t = (1-\alpha) \cdot \tilde{r}_{t-1} + \alpha \cdot r_t \tag{4.28}$$

In decentralized multi-agent settings, this shaping is done separately for each fog agent i, allowing agents to adapt independently to local dynamics. We adapted MADRL scenarios from Baek and Kaddoum (2021a), where agents are designed to cooperate to achieve a common goal, typically by sharing a single reward function that reflects the performance of the entire system. This means all agents work towards maximizing the same objective of minimizing total system delay and energy consumption as a set common objective for multiple agents. Agents may share their local rewards as feedback Chen et al. (2022). This approach reduces communication overhead compared to sharing observations and actions. The learning mechanism in such environments relies on DRL principles. However, in this MDP-based learning process, the agent doesn't know the direction in which to proceed while gathering the maximum reward. This is where the Bellman Eq 4.5 applies in RL-learning to enable the agent with the memory from the transition in the MDP process. The Bellman Equation (2.7) helps approximate the action-value function $Q(s,a)$ during value iteration as follows:

$$Q(s,a) \leftarrow r + \gamma \cdot \max_{a'} Q(s',a') \tag{4.29}$$

Therefore, Eq 4.29 essentially states that the optimal policy for a given state (s) is the action that maximizes the sum of the immediate reward and the discounted value of the next state, considering all possible next states (s') and their associated transition probabilities (p). Applying Eq. 4.29 recursively for each state in the environment to obtain the optimal policy for the entire environment.

In our case, we utilize the TD3-based MAFCPTORA algorithm to identify the optimal deterministic policy from stochastic environments. The TD3 method is particularly suited for continuous action spaces Wakgra et al. (2024a), where the actor network learns a policy ($\mu(s;\theta)$ and the critic evaluates its quality using twin Q-value networks:

$$\mu : \mathcal{S} \rightarrow \mathcal{A}$$

4.10.1 TD3 agent

The TD3 agent presented in figure 4.5 is designed in a decentralized manner as depicted in algorithm 2 and 3 where the agent is deployed at each fog node to make independent offloading decisions. Taking the collective offloading action to maximize agents' expected reward from the continuous action space. In the horizontal interaction of F2F environments, there are N fog nodes presented as the local node F_m and the adjacent node F_j to execute the task $(K_1, K_2, \ldots, K_n)$ received from the user device. In this architecture N actions are defined $\alpha_1, \alpha_2, \ldots, \alpha_n$, where they combined to form an action a time t, such that $\alpha_t = (\alpha_{t,1}, \alpha_{t,2}, \ldots, \alpha_{t,n})$. Since continuous action space is considered, each action sums up to 1. For this case, to enable the fog node to take action for themselves independently, a decentralized policy is used, then combines their actions to achieve collective reward.

For that reason, N actor policy networks $\pi_{\phi 1}, \pi_{\phi 2}, \ldots, \pi_{\phi n}$ with parameter $\phi_1, \phi_2, \ldots \phi_n$ and actor target $\phi'_1, \phi'_2, \ldots \phi'_n$ employ in a parallel manner to find optimal offloading decisions. In addition, two critic networks Q_{ϕ_1} and Q_{ϕ_2} with their parameter ϕ_1, ϕ_2 and the critic target network $Q_{\phi'_1}$, $Q_{\phi'_2}$ with parameter ϕ'_1, ϕ'_2 were used to mitigate Q-value overestimation.

To optimize and stabilize the deep neural network, the replay buffer $\mathcal{D}$ is used to store the experience as a mini-batch. The decentralized TD3 agent observes the state s_t and takes action α_t at each time step t using the actor policy networks (Equation 2.6) and (Equation 4.30).

$$\alpha_t(s_t) = [\alpha_{t,1}(s_t), \alpha_{t,2}(s_t), \ldots, \alpha_{t,n}(s_t)] \tag{4.30}$$

At each action of the agent in fog environments, each observation is stored in the replay buffer $\mathcal{D}$ in a single transition of the future state $s'_{i,t+1}$ with respect to previous action a_t, reward $r_{i,t}$ and done flag $d_{i,t}$ presented as $(s_{i,t}, a_{i,t}, s'_{i,t+1}, r_{i,t}, d_{i,t})$. After the replay buffer $\mathcal{D}$ obtained from experience, the individual agent then updates its neural network from the random mini-batch transition $(s_{i,t}, a_{i,t}, s'_{i,t}, r_{i,t}, d_{i,t})$. The actor target policy used to

take the next action a'_t for the next state s'_t and the Gaussian noise clipping to avoid policy stagnation and overestimation (2.6).

4.11 Algorithm Design

To address the aforementioned task offloading problem, we employ a decentralized, multi-agent fully cooperative partial task offloading and resource allocation (MAFCPTORA) algorithm based on the actor-critic network. The MAFCPTORA algorithm is designed in two steps: task offloading decision and resource allocation.

4.11.1 Parallel MAFCPTORA-Algorithm

We designed a novel multi-agent fully cooperative partial task offloading and resource allocation (MAFCPTORA) algorithm based on probability using the TD3 approach. A decentralized TD3 agent is deployed at each F_n and is responsible for their own independent decision to find the optimal offloading policy. The TD3 agent is utilized as an off-policy algorithm with a deterministic policy $\mu(s;\theta)$ and twin Q-value functions $Q_1(s,a;w_1)$ and $Q_2(s,a;w_2)$ to estimate the expected return. For this task offloading problem, DDPG was used in Seid et al. (2023); however, overestimating Q-values poses a challenge for policy breaking due to exploiting errors in the Q-function. We employed TD3, which uses two Q-value functions, Q_{ϕ_1} and Q_{ϕ_2}. These functions are trained by minimizing the Mean Squared Bellman Error (MSBE), ensuring stable value estimation, as presented in Equation(2.7). The clipped double-Q learning is defined where both of the Q-functions use the single target to find a smaller target value using the two Q functions in Equation 2.7. In this case, the fog agent uses both Q-functions learn by regressing to the target as Equation (2.8) and (2.9). The algorithm applies the parameter ($\mu\theta(s)$) to update the agent's knowledge, then perform the gradient descent of the performance measure $Q(\phi 1)$. Parallelization within the algorithm is a practical approach to address resource limitations in fog environments by dividing tasks into smaller, manageable components Lim (2022). This MAFCPTORA algorithm is decomposed into two sequential stages, as outlined in Algorithm 2 and Algorithm 3, which correspond to task offloading and resource allocation $R_{f,k}(t)$, respectively. The most important point in these parallel algorithms are enhancing task execution performance while optimizing resource allocation decision.

In the algorithm the state information, action selection algorithm, the training algorithm, and training and evaluation environments are defined. Here, the initial parameters and maximum episodes iterations are configured. For the decision of each episode, it runs for τ time steps, as written in line 5 to 14 of the Algorithm 2. For each decision steps the action selection method is invoked, then the reward are computed, as well the experience

is stored as a replay buffer $\mathcal{D}$. Finally, after the training iteration of each step concluded, evaluation continues using the same evaluation environments for each DRL models to observe the performance improvement.

Algorithm 2: Multi-agent fully cooperative partial task offloading

Input: Initial policy parameter θ, Q-function parameters ϕ_1, ϕ_2, number of fog nodes F_n, learning rate α, discount factor γ, number of training episodes e, agent id i, time horizon τ, replay buffer size $\mathcal{D}$, exploration rate ϵ

Output: Optimal policy π

Initialize: $\theta_{\text{targ}} \leftarrow \theta$, $\phi_{\text{targ},1} \leftarrow \phi_1$, $\phi_{\text{targ},2} \leftarrow \phi_2$;

foreach $F_m \in \{F_1, \ldots, F_n\}$ **do**

  Initialize F_m, μ_{θ_m}, $Q_{\phi_{1,m}}, Q_{\phi_{2,m}}$, and targets $\hat{\mu}_{\theta_m}, \hat{Q}_{\phi_{1,m}}, \hat{Q}_{\phi_{2,m}}$;

  Observe initial state $S_m(0)$;

foreach *decision step* $t = 0, \ldots, \tau - 1$ **do**

  foreach *fog node* F_m ***in parallel*** **do**

    Generate exploration noise: $a_m(t) = \mu_{\theta_m}(S_m(t)) + \mathcal{N}(0, \sigma)$;

    Clip action: $a_m(t) \leftarrow \text{clip}(a_m(t), a_{low}, a_{high})$;

    With probability P_m, select random $a_m(t)$ (exploration);

    Compute offloading ratio: $x_{m,j} \sim \pi(x_{m,j}|s_m(t))$;

    Allocate task portions $K_{x_{m,j}}$ to neighbor fog nodes F_j;

    Execute offloaded tasks at F_j and remaining $x_m = 1 - \sum_j x_{m,j}$ locally;

    Observe $s_{m,j}(t+1)$ and compute reward using Equation (4.28);

    Store state transition $(s_{m,j}(t), a_{m,j}(t), r_{m,j}(t), s_{m,j}(t+1), d)$ in $\mathcal{D}$;

  foreach *fog node* F_j ***in parallel*** **do**

    if *enough samples in* $\mathcal{D}$ **then**

      Obtain sample mini-batch $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$;

      Compute target actions: $a' = \mu_{\theta_{targ}}(s') + \mathcal{N}(0, \sigma)$;

      Compute target Q-value using Equation (2.8);

      Update critics ϕ_1, ϕ_2 via gradient descent:

$$\phi_i \leftarrow \phi_i - \alpha \nabla_{\phi_i} \mathbb{E}_B[(Q_{\phi_i}(s, a) - y)^2]$$

      ;

      Update actor θ via deterministic policy gradient ascent using Equation (4.31);

      Update target networks using Equation (4.32);

A deterministic policy gradient to update the two critics ϕ_1, ϕ_2, Equation (4.31) and

target networks $\phi_{targ,i}$ with Equation (4.32).

$$\theta \leftarrow \theta + \alpha \nabla_\theta \mathbb{E}_{s \in B}[Q_{\phi_1}(s, \mu_\theta(s))] \tag{4.31}$$

$$\phi_{targ,i} \leftarrow \rho \phi_{targ,i} + (1-\rho)\phi_i \theta_{targ} \leftarrow \rho \theta_{targ} + (1-\rho)\theta \tag{4.32}$$

In the TD3 algorithm, a deterministic policy $\mu(s;\theta)$ is trained to map the observed state $s(t)$ to an action $a_i(t)$. The policy is learned to maximize the Q-value estimated by the critic network Q_{ϕ_1} guided by Equation (4.33):

$$\max_\theta \mathbb{E}_{s \sim D}\left[Q_{\phi_1}(s, \mu_\theta(s))\right] \tag{4.33}$$

In this work, maximizing the cumulative reward is the objective of each agent in this decentralized multi-agent system. Furthermore, the cooperation of each TD3 agent (fog node) with the others follows its respective trained policy, denoted as $\mu = (\mu_1, \mu_2, \ldots, \mu_n)$. In addition, the expectation ($\mathbb{E}$) is defined with the sum over time and the discounted reward by γ at each time step t. Then, the TD3 agent $F_{m,j}$'s value function under policy μ_i at state s is defined by Equation (4.34):

$$V^{\mu_i}(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R_i(s_t, a_t, s_{t+1})\right], \tag{4.34}$$

where $R_i(s_t, a_t, s_{t+1})$ is the reward function at time step t.

The value of agent $F_{m,j}$ in this multi-agent system depends on the collective policy μ rather than just its own policy μ_i. The relationship is given by Equation (4.35) and (4.36):

$$V_i^{\mu_i^*, \mu_{-i}^*}(s, w) \geq V_i^{\mu_i, \mu_{-i}^*}(s), \ \forall s \in \mathcal{S} \text{ and } \mu_i, \tag{4.35}$$

where μ_{-i}^* represents the set of policies of all other agents except π_i.

Additionally, the extended ϵ-Nash equilibrium is defined by the following general formula:

$$V_i^{\mu_i^*, \mu_{-i}^*}(s) \geq V_i^{\mu_i, \mu_{-i}^*}(s) - \epsilon, \ \forall s \in \mathcal{S} \text{ and } \mu_i. \tag{4.36}$$

Here, μ_i^* is the best response of agent $F_{m,j}$ to the policies μ_{-i}^* of other agents.

The reward function presented uses iterative computation of the optimal value function, which represents the maximum expected cumulative reward that an agent can receive from a given state s under an optimal policy $\pi^*(s)$. So, it is used to draw the value iteration (V_i) in each state. It starts by initializing the value function $V^{(0)}(s)$ to 0 for all states s. Then, for each iteration i, the algorithm updates the value function $V_{(i+1)}(s)$ for each state s by taking the maximum expected cumulative reward over all possible actions that can be taken from that state s. In the reward function, a hyperparameter α and γ are used to measure how far the agent reaches its goal and to govern the algorithm's

pace for updating the value parameters, respectively.

Resource allocation is the critical context in dynamic and heterogeneous fog environment for latency sensitive applications.

Algorithm 3: Multi-agent fully cooperative resource allocation

Input: Trained actor networks μ_{θ_i} for all agents $F_{m,j}$

Initialize task resource requirements: V_k, θ_t, H, and $F_{m,j}$ resources (C_f, memory, β);
Initialize μ_{θ_i} and Q_{ϕ_i} for each agent $F_{m,j}$ with random weights θ_i, ϕ_i;
Initialize target networks π'_{θ_i} and Q'_{ϕ_i} with weights $\theta'_i \leftarrow \theta_i, \phi'_i \leftarrow \phi_i$;
for *episode* $e = 0, 1, \ldots, E-1$ **do**
    foreach *decision step* $t = 0, 1, \ldots, \tau - 1$ **do**
        `// Parallelize the following for each agent` $F_{m,j}$
        foreach $F_{m,j}$ *in parallel* **do**
            Use trained actor network to decide resource allocation by Equation (4.37);
            Allocate C_f, memory, β for task processing based on $a_i(t)$;
            Monitor latency, energy consumption, and resource utilization by Equations (4.24d)–(4.24f);
    Compute feedback metrics using Equation (4.27);
    Update task states for the next decision step using Equation (2.8);

Algorithm 3 outlines the proposed multi-agent fully cooperative resource allocation process. Each fog node (agent $F_{m,j}$) uses its trained actor network μ_{θ_i} to determine optimal resource allocation decisions for task execution. Initially, the algorithm sets up task requirements and resource capacities, followed by the initialization of actor–critic networks and their target network. During each episode, agents operate in parallel to allocate CPU, memory, and bandwidth according to the actions generated by their policies. After execution of latency, energy consumption, and resource utilization are computed as a performance metrics, the task states are updated for subsequent decision steps. In this F2F cooperative mechanism allows to enables decentralized yet coordinated optimization of resource usage across fog nodes.

4.11.2 Gaussian Noise

In this TD3-based task offloading, Gaussian noise is added to the stochastic policy to encourage exploration. In F2F partial offloading model where the task offloaded to multiple fog nodes the vector is represent the workload distribution of $\in [0, 1]$ with their sum ≤ 1. The action selection formula uses off policy with Gaussian noise for exploration:

$$a_i(t) = \mu_{\theta_i}(S_i(t)) + \mathcal{N}(0, \sigma) \tag{4.37}$$

- $a_i(t)$: The action chosen by agent $F_{m,j}$ at time t.
- $\mu_{\theta_i}(S_i(t))$: The output of the actor network parameterized by θ_i, given the state $S_i(t)$.
- $\mathcal{N}(0, \sigma)$: illustrates the Gaussian noise in terms of mean (0) and standard deviation σ Equation (3.2).

The addition of Gaussian noise ensures that the agent explores actions around the deterministic policy μ_{θ_i} and avoids premature convergence to suboptimal solutions. Moreover, the update critic Q_{ϕ_i} is done by minimizing the loss function.

$$L(\phi_i) = \mathbb{E}[(Q_{\phi_i}(S, A) - y)^2] \tag{4.38}$$

where $y = r_i + \gamma Q_{\hat{\phi}_i}(S', A')$

In this case, $L(\phi_i)$ presents the loss function for the critic network (Q_{ϕ_i}) parameterized by ϕ. The loss measures the mean squared error (MSE) between the predicted Q value and the target Q value y. The target Q-value y combines the immediate reward r_i and the discounted future value of the next state-action pair, estimated using the target critic network $Q_{\hat{\phi}_i}$. We use this MSE update to train the critic to better predict the expected cumulative reward for each pair of state actions. The smallest mean square error (MSE) between the predicted reward of the critic model and the target reward is then calculated, and this error is used to update the twin critic models using delayed backpropagation. Minimizing this loss improves the accuracy of value estimates, which aids the actor in policy optimization.

4.11.3 Model Sensitivity Analysis

To evaluate the robustness of the proposed MATD3-based offloading model, a sensitivity analysis was conducted by varying key environment parameters such as task arrival rate, network delay, and energy coefficients. The analysis revealed that the model's performance is moderately sensitive to network delay, suggesting that accurate modeling of communication dynamics is crucial for realistic performance estimation. In Algorithms 2 and 3, we update the actor policy ϕ_i at each time step (t), corresponding to delayed policy update by estimating the Q-value of state and action using Equations (4.31) and (4.32). The computational complexity of the TD3-based MAFCPTORA algorithm is influenced by the number of agents (N), state and action dimensions (d_s and d_a), batch size (B), and hidden layer width (H). For each agent, the actor and critic networks contribute a forward pass complexity of $O(H(d_s + d_a) + H^2)$. During training, the critic updates dominate the time complexity with $O(2B \cdot (H(d_s + d_a) + H^2))$ per agent. Including

the delayed actor updates, the total training step per agent scales as $O(B(d_s+d_a+H)H)$. Hence, for N agents:

$$O(N \cdot B \cdot (d_s + d_a + H)H)$$

Dynamic task partitioning involves computing and normalizing task distribution vectors across n neighbors, leading to a per-agent cost of $O(n)$ and a total of $O(N \cdot n)$ per time step. In decentralized environments, communication overhead arises due to state sharing among neighbors, costing $O(N \cdot n \cdot d_s)$ per step. MATD3 demonstrates faster convergence compared to benchmark solutions, including MAIDDPG, MASAC, and MAPPO. Therefore, our MAFCPTORA algorithm maintains scalable performance with linear growth in agents and manageable quadratic growth in network size.

Chapter 5

RESULTS AND DISCUSSIONS

In this section, we evaluate the performance of the proposed MAFCPTORA approach. We also compare it with several existing approaches. Specifically, we used the MAIDDPG Lillicrap et al. (2015), MATD3 Fujimoto et al. (2018), MASAC Haarnoja et al. (2018), and MAPPO Schulman et al. (2017) DRL baseline algorithms. These algorithms are suitable for decentralized training and execution in a continuous action space, where each action represents the proportion of a task to be offloaded to neighboring fog nodes, modeled as a probability between 0 and 1. This implementation represents the state as task state (task size), resource status, channel conditions of connected nodes (allocated or free), and neighboring nodes. These algorithms follow the actor-critic network, where the network trains using exploration and exploitation, updating the network based on the rewards received. Meanwhile, the critic observes the global states and actions of all individual autonomous agents $F_{m,j}$ and evaluates the combined actions of the agents. The assumption is that each user device may connect to multiple fog nodes and select multiple nodes to offload its sub-task at a particular time slot (t). This selection process is guided by the collective reward function presented in Equation (4.24a) comparing task completion time to the delay threshold.

5.1 Simulation Settings

In this section, the simulation experimental setting adopted for simulation is described. We consider a scenario of a multi-user (U_n) and multi-fog nodes (F_n) environment. To reflect the validity and transferability to real-world logic the model parameters extracted from literature with real-world trace and introduced Gaussian noise and uncertainty. Each (U_n) is equipped with a single-core CPU with a frequency between 1 GHz and 2 GHz, while ($F_{m,j}$) features multi-core CPUs with 2 to 8 cores and frequencies ranging from 1.6 GHz to 3 GHz Fu et al. (2021). The computing and communication capabilities of $F_{m,j}$ are assumed to be heterogeneous. The bandwidth between U_n and F_m is 250 Kbps

to 54 Mbps, depending on the size of the task and the data transmission rate between $F_{m,j}$ can ranges up to 100 Mbps Al-Khafajiy et al. (2019a). The offloading decision is affected by the task arrival rate (λ) packets per second, as indicated by Wakgra et al. (2024a) to measure the workload intensity of $F_{m,j}$. The task execution rate (μ) at $F_{m,j}$ is measured in terms of the total task executed per episode duration. A custom fog environment is designed based on OpenAI Gym Brockman et al. (2016) standards. Table 5.1 presents the simulation parameters along with their values.

Table 5.1: Simulation parameters.

Parameters	**Quantity**
Number of User devices	100
Number of fog nodes	6
Distance between fog nodes ($Dist_{m,j}$)	1 km to 5 km
Task arrival rate λ at $F_{m,j}$	3×10^2 packets per second
Available CPU core $\nu_{f_{m,j}}$	[2–4] number of cores ν
Bandwidth between fog nodes β	50 MHz
Channel condition H	$[0 \leq H \leq 1]$
Computational capacity $U_i^{\min}$	[100–1000] MIPS
Computational capacity $F_{m,j}^{\max}$	50 MIPS
CPU frequency range $F_{m,j}{}_{min}^{max}$	$[200 \times 10^6 - 15 \times 10^8]$ Hz
Path loss exponent $PL(d_{uf})$	2.7 dB
User device minimum channel transmission power $\tau_{t,r}^u(t)$	10 dBm
User device maximum channel transmission power $\tau_{t,r}^u(t)$	30 dBm
Number of episodes	10000
Observation steps per episode	100
Learning rate α	0.001 and 0.0001
Discount factor γ	0.99
Weight coefficients ω_t and ω_e	0.7 and 0.3
Number of hidden layer	2
Number of neurons in hidden layer	256
Batch size	64

5.1.1 Hyperpararameter Selection

The hyperparameters used in this study are learning rate (α), discount factor (γ), replay buffer memory ($\mathcal{D}$), batch size ($\mathcal{B}$), target delay, weight coefficient for latency (ω_t) and energy (ω_e). In DRL algorithms selecting the value of these hyperparameters tunings are challenging and no defined rules Gebrekidan et al. (2024). For learning rate (α) the value obtained using the principle from Gulde et al. (2020) and the training process experience. Here, (α) manages the training progress to determines the steps in episode size in each gradient decent with iteration updates. The agents cumulative weighted long-term reward of Q-values for latency-energy trade-off are guided the discount factor value, which is γ value ranges between 0.98 and 0.995. In our fog-to-fog environment the stable convergence (γ) value at 0.99. The weight coefficient of (ω_t) and (ω_e) is 0.5 initially, that is equal weight for both energy and latency. However, their performance are not stable, so that the training episode guides the value of (ω_t) and (ω_e) as 0.7 and 0.3 indicated in Table 5.1, respectively. The remaining hyperparameters are selected based on training experience from the simulation results: $\mathcal{D}$ size 10000, ($\mathcal{B}$) size 64, and observation step 100.

5.2 Simulation Results and Performance Analysis

The simulation results were structured to address key research objectives: improving offloading decisions and minimizing overall task execution system latency and energy consumption in stochastic horizontal fog environments. In comparing these algorithms, we conducted the training based on Table 5.1 for 10,000 episodes. Our MAFCPTORA algorithms integrated with multi-agent PPO, IDDPG, SAC and TD3 in parallel task execution approach. Each model is able to deal with this stochastic network and shows good learning progress in training. We train all models with common F2F customized environments designed using PyTorch python framework. All hyperparameters and fog nodes resources remain the same for each model training to maintain consistence for validation. In this dissertation MATD3 based model shows persistence and improved result for each training episode as depicted Figure 5.1.

In this dissertation, we acknowledge the result of this simulation is highly dependent on assumption and lacks real-world scenarios such as hardware heterogeneity, network fluctuations, and environmental noise. This logic may lead the model perform well but fail to generalize in real world deployment where results can be biased or misleading. Therefore, the network model is designed with real-world model parameters to minimize factors pose by the simulation.

In the Figure 5.1 presents the MATD3 training result with respect to episode rewards, episode latencies, episode energy and episode successful task completion. From these

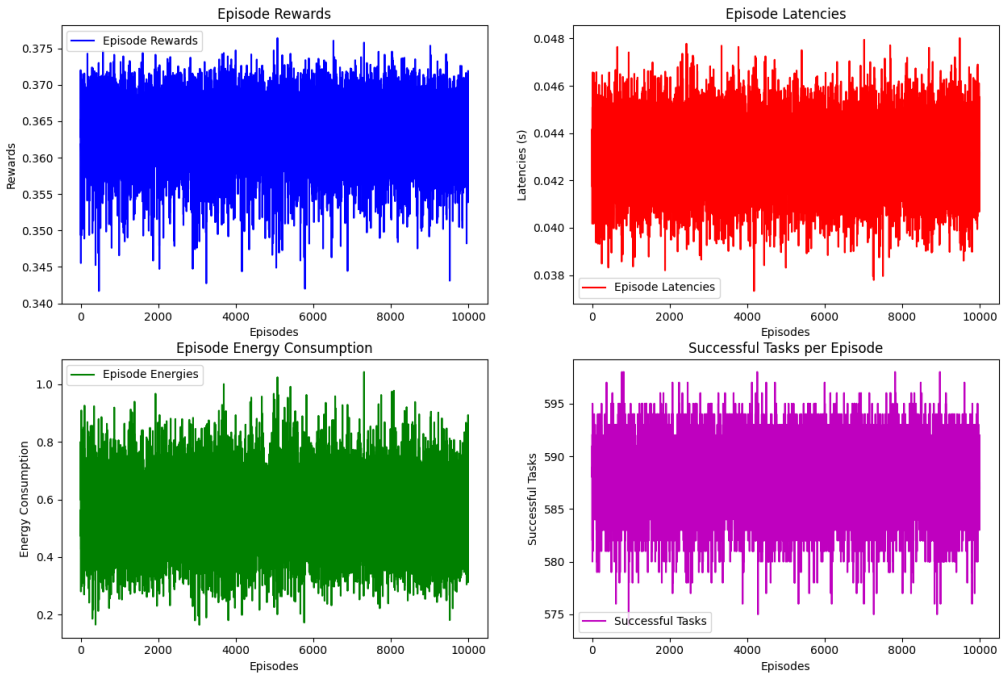

Figure 5.1: MATD3 training result for reward, latency, energy consumption, and task completion rate per episodes

perspectives, MATD3 outperforms the other by return a stable, progressive, and long-term cumulative reward formulated in Equation (4.24a). The training metrics result of MATD3 for episode rewards ranges from 0.34 to 0.75, episode energy consumption from 0.2 to 1.0, episode latencies from 0.02 to 0.048 and successful task completion from 575 to 595 per each 100000 episode. These training result were stochastic average standard values, the smoothing conducted with mean values as shows in figure 5.1. The overall system-wide evaluation metrics are defined based on the average reward, latency, and energy per 100 episodes to get a more stable and reliable performance estimation. Therefore, the average reward, the average latency, and the average energy consumption per episode are recorded every 100 steps.

Figure 5.2 (**a**) illustrates episode-wise evaluation of reward, (**b**) average latency evaluation per episode, (**c**) and average energy evaluation per episode performance of the MAFCPTORA algorithms. Figure 5.2 (**a**) presents the evaluation reward comparison for the MAPPO, MASAC, MAIDDPG, and MATD3 models. Figure 5.2 (**b**) presents the latency evaluation, where MATD3 quickly converges and maintains a consistently low response time. In contrast, MASAC struggles with higher latency, exhibiting a significantly longer response time compared to the other algorithms. Figure 5.2 (**c**) presents the evaluation of energy consumption per episode, highlighting MATD3's efficiency with notably low energy usage. All evaluation results demonstrate progressive learning for each model. According to

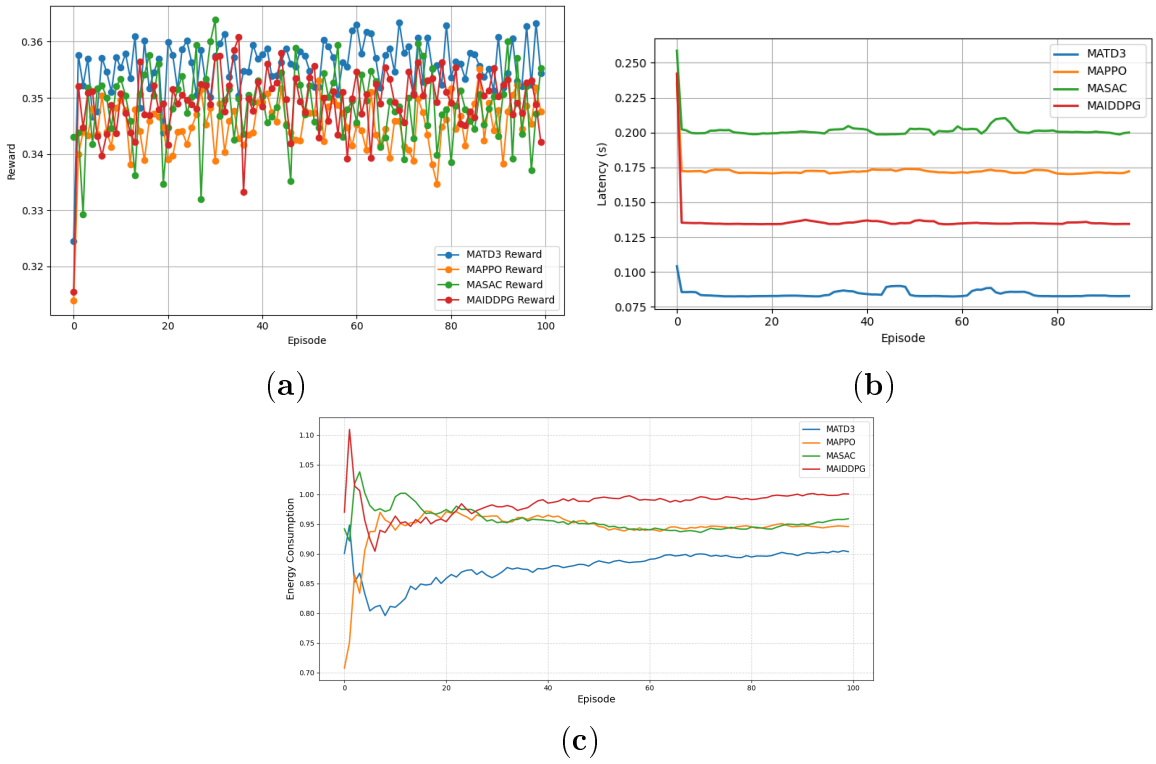

Figure 5.2: Performance evaluation of MAPPO, MAIDDPG, MASAC, and MATD3 for average reward, latency, and energy consumption per episode.

Figure 5.2, the MATD3 algorithm delivers a consistent performance across several metrics and manages the trade-off between offloading decisions, response time, and energy consumption quite well. The result of evaluation per episode reward within the ranges from 0.30 to 0.37 for each model, where MATD3 outperform other models. The performance of episode latency also showing a promising result falls between 0.075 to 0.250 with MATD3 shows excellent response time. Finally, energy consumption per episode varies from 0.7 to 1.4 where MATD3 still demonstrate very acceptable performance.

Figure 5.3 illustrates the performance of MATD3 based on individual metrics: average reward, latency, and energy consumption evaluated per episode. The average reward, latency, and energy performances of MATD3 are stable in convergence with progressive learning. Here in Figure 5.3 (**a**) MATD3: average reward per episode, (**b**) MATD3: average latency per episode, (**c**) MATD3: average energy per episode. Figure 5.3 (**a**) illustrates the average reward evaluation results, which stays stable in a range between 0.32 and 0.36 across the episodes. Figure 5.3 (**b**) shows promising latency, mostly ranging between 0.08 and 0.18s, suggesting relatively best response times. Figure 5.3 (**c**) shows the energy consumption laid between 0.6 and 1.4, indicating a tolerable balance between performance and efficiency.

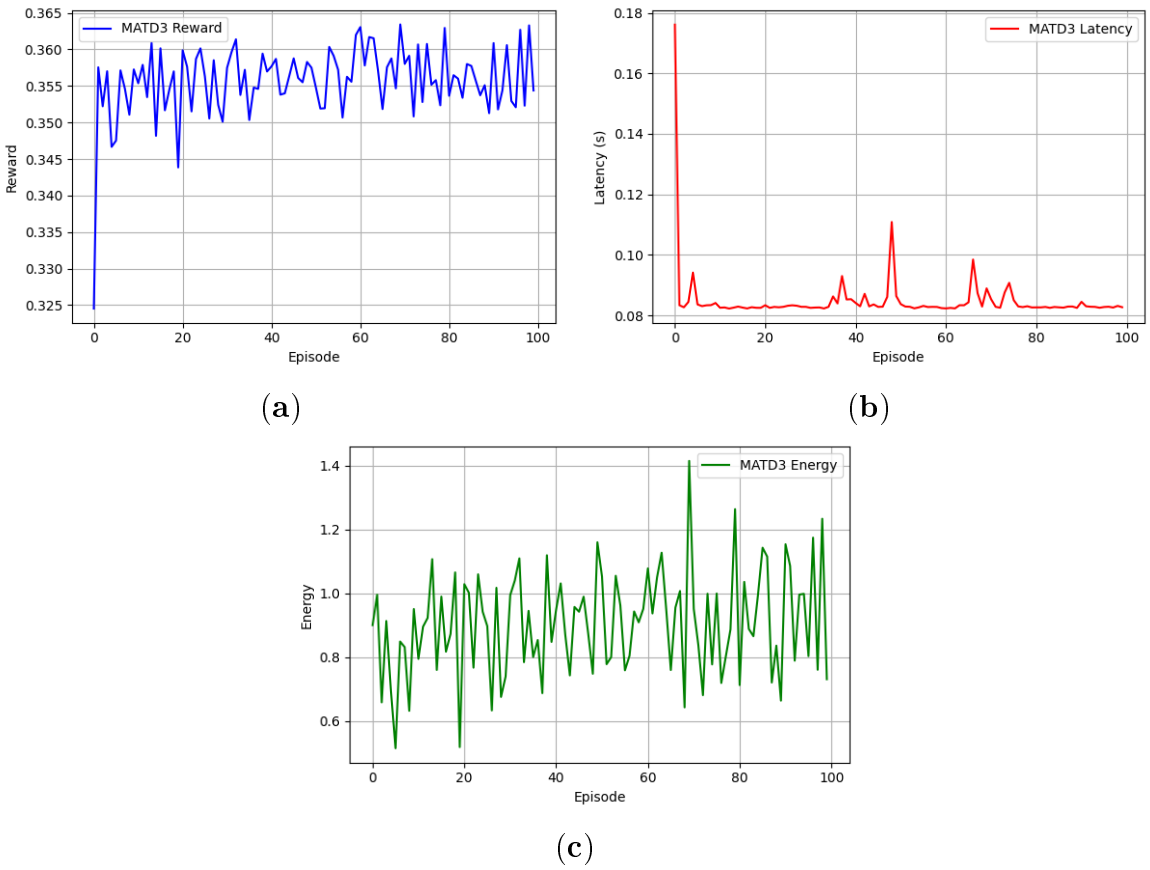

Figure 5.3: Performance evaluation of MATD3 for average reward, latency, and energy consumption per episode.

5.2.1 Ablation Study

The ablation study is conducted for MATD3, our main evaluation target, to examine the performance sensitivity with a change in the weighting parameter (ω). Comparative methods are used as baselines without extensive parameter tuning, consistent with their original configurations. Figure 5.4 shows the ablation study for the effects of the weight coefficient on average reward, latency, and energy consumption. In this study, the base weight coefficients ω_t and ω_e are set to 0.5, ensuring that latency and energy consumption are treated with equal importance in the joint optimization objective. The ablation performance with change in weighting parameter (ω) conducted for on MATD3 model. The result of the base weight coefficients shows a balanced trade-off. Hence, we reduced ω_e to prioritize for minimizing task execution latency over energy consumption. The ablation study indicates that prioritizing latency significantly reduces average latency while energy consumption increases, but it maintains an acceptable range. The best trade-off was observed at $\omega_t = 0.7$, achieving both low latency and minimal energy consumption. Moreover, the average reward exhibits an overall increasing trend; however, a slight decline is observed at the final stage, likely due to the influence of minimized latency.

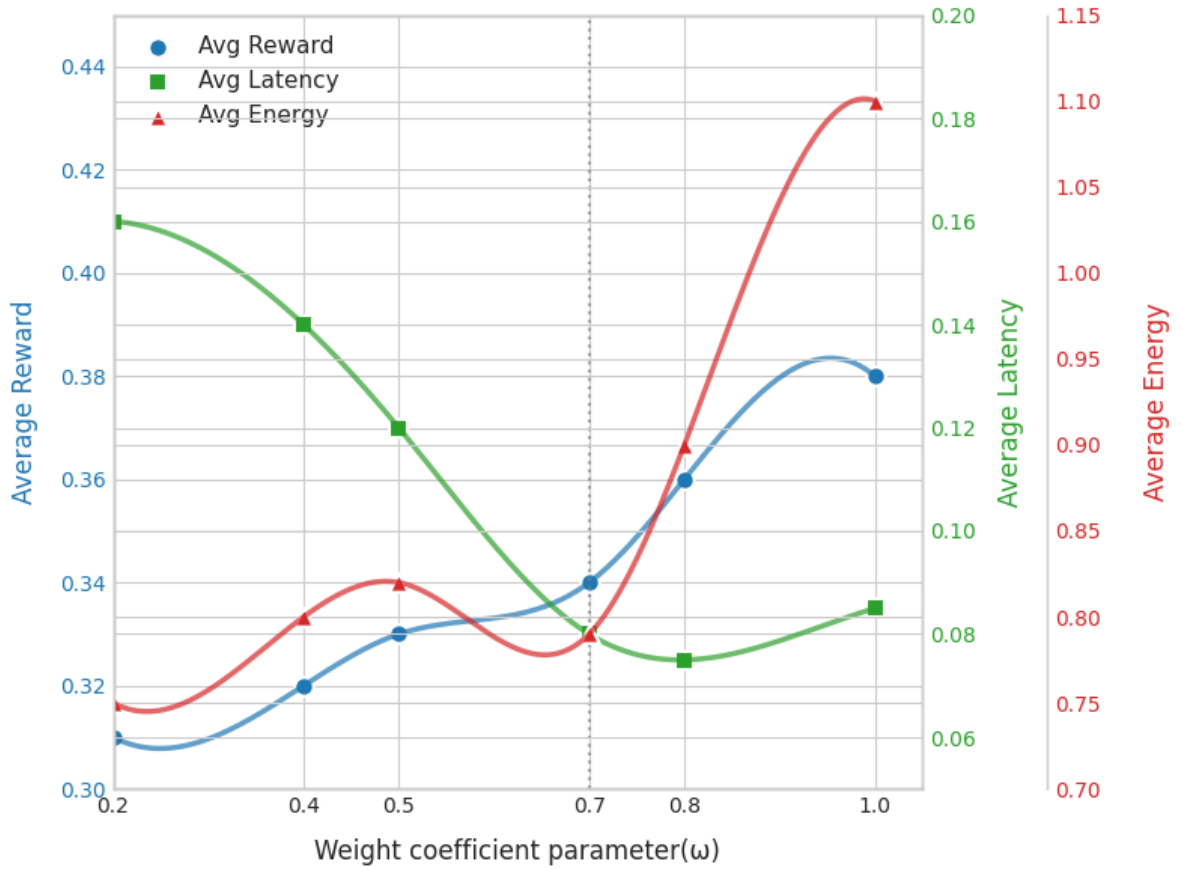

Figure 5.4: Ablation of MATD3 for average reward, latency, and energy consumption per episode.

5.2.2 Performance Evaluation

The evaluation findings of MAPPO, MASAC, MAIDDPG, and MATD3 are presented in Table 5.2 according to the average reward, latency, and energy consumption. Among the models compared, MATD3 shows superior performance in all three metrics. It achieves the highest average reward of (0.36 $\pm$ 0.01), indicating more effective task offloading decisions. Furthermore, it significantly reduces the average latency to (0.08 $\pm$ 0.01), which is considerably lower than the other three models, suggesting faster task execution. Energy consumption is also minimized at (0.76 $\pm$ 0.14), showcasing the model's efficiency in resource usage.

In contrast, MAPPO and MASAC show relatively competitive rewards despite having higher latency and energy consumption. They achieve a nearly similar reward level 0.34 $\pm$ 0.01) and (0.35 $\pm$ 0.01), respectively. The highest average latency is observed in MASAC, with a score of (0.2 $\pm$ 0.03), indicating a slower response time relative to the other models. Among the four models, MAIDDPG demonstrates the highest average energy consumption, measured at 0.999 $\pm$ 0.187. The convergence of these models is observed to be stable during the training period, with low standard deviations across

metrics, affirming the reliability of the learned policies. Our experimental results further validate the effectiveness of this approach, where parallel F2F task execution enabled the best-performing model. It demonstrate suitability for real-time, and distributed applications.

Table 5.2: Summary evaluation results for the four models.

Model	**Ave Reward**	**Ave Latency**	**Ave Energy Consumption**
MAPPO	0.34 ± 0.01	0.17 ± 0.02	0.96 ± 0.15
MASAC	0.35 ± 0.01	0.20 ± 0.03	0.92 ± 0.14
MAIDDPG	0.32 ± 0.01	0.143 ± 0.014	0.999 ± 0.187
MATD3	**0.36 ± 0.01**	**0.08 ± 0.01**	**0.76 ± 0.14**

[1]

To check the stability of our MATD3 model performance, the training was conducted with 0.5, 0.6, 0.7, and 0.8 latency weight factors. The results presented in table 5.3 shows a more stable and consistency.

Table 5.3: The ablation summary evaluation results for the MATD3 model with different latency weight factors.

Latency weight coefficient (ω)	**Ave Reward**	**Ave Latency**	**Ave Energy Consumption**
0.5	0.33 ± 0.011	0.13 ± 0.00	0.78 ± 0.010
0.6	0.34 ± 0.01	0.09 ± 0.011	0.81 ± 0.11
0.7	**0.36 ± 0.01**	**0.08 ± 0.01**	**0.76 ± 0.14**
0.8	0.36 ± 0.01	0.10 ± 0.001	0.79 ± 0.12

[1]**Note:** Bold values in table 5.2 and 5.3 indicate the best performance achieved by the MATD3 algorithm for each metric.

Chapter 6

CONCLUSION AND FUTURE WORK

6.1 Conclusion

The fog computing paradigm is designed to address the computational challenges of delay-sensitive applications generated by the user devices. For such user devices, tasks are offloaded to fog nodes to solve the resource constraints problem to meet their performance requirements. Task offloading happens when local task computation models can not address the user device's QoS requirements. Therefore, offloading to distributed fog nodes, depending on the goals of the applications, becomes a solution. A complete survey is presented on task offloading strategies that maintain task QoS requirements for real-time applications in the user device-to-fog computing environment. Plus, the application of the offloading model to different application domains in a real-time manner is investigated. As a result, various task offloading strategies in fog using centralized, decentralized (distributed), and federated architectural design approaches are presented. Specifically, techniques are presented for optimizing QoS requirements in task offloading for dynamic and static topological fog architectures. Moreover, different task offloading approaches emphasize the convergence problem, as the search space grows exponentially and increases run-time processing. Furthermore, applying continuous updating is challenging in these approaches because they focus on immediate results, which leads to overall performance degradation in the long run. Recently, DRL techniques were utilized; therefore, this work focuses on the application of DRL techniques to enhance task offloading and resource allocation decisions in the continuous search space of the fog.

In this dissertation, we addressed the challenges of task offloading in dynamic fog computing environments, particularly under scenarios with high computational demands. We proposed a MAFCPTORA, a decentralized multi-agent deep reinforcement learning approach for partial task offloading in a horizontal fog-to-fog (F2F) architecture. This

approach allows fog nodes to make autonomous and coordinated decisions, optimizing sub-task offloading based on real-time resource status, workload, and network conditions. The proposed MAFCPTORA algorithm enhances task execution efficiency by reducing latency and energy consumption while maintaining the quality of service (QoS) for end-user applications. We evaluated the performance of our method against various state-of-the-art MADRL algorithms, including MATD3, MASAC, MAIDDPG, and MAPPO. MATD3 demonstrated superior performance, achieving the highest average reward of 0.36 $\pm$ 0.01, the lowest average latency of 0.08 $\pm$ 0.01, and the lowest energy consumption of 0.76$\pm$0.14. These results validate the effectiveness of our decentralized approach for scalable and efficient task offloading in fog computing environments.

6.2 Future work and recommendation

In future work, we plan to consider the task success rate for dynamic resource requirements while optimizing response time. We design the tasks to be split into sub-tasks so that dependency-heavy tasks are planned to be addressed in the extension of this work. Implementing the proposed algorithm in practical application scenarios for deployment is also important, as it provides valuable insights. The absence of a multi-objective optimization framework to jointly balance different QoS requirements in uncertain network conditions is a major gap, and plan to address in the future. Therefore, the next plan is to jointly balance latency, energy usage, bandwidth constraints, and battery health under dynamic and uncertain network conditions. Also, we plan to extend our work with a fault tolerance approach when a fog node fails. A fallback mechanism is important in mission-critical tasks.

BIBLIOGRAPHY

Abdelghany, H. M. (2025). Dynamic resource management and task offloading framework for fog computing. *Journal of Grid Computing*, 23.

Al-Khafajiy, M., Baker, T., Al-Libawy, H., Maamar, Z., Aloqaily, M., and Jararweh, Y. (2019a). Improving fog computing performance via fog-2-fog collaboration. *Future Generation Computer Systems*, 100:266–280.

Al-Khafajiy, M., Baker, T., Waraich, A., Al-Jumeily, D., and Hussain, A. (2019b). Iot-fog optimal workload via fog offloading. *Proc. - 11th IEEE/ACM Int. Conf. Util. Cloud Comput. Companion, UCC Companion 2018*, pages 349–352.

Alfakih, T., Hassan, M. M., Gumaei, A., Savaglio, C., and Fortino, G. (2020). Task offloading and resource allocation for mobile edge computing by deep reinforcement learning based on SARSA. *IEEE Access*, 8:54074–54084.

Ali, E. M., Lemma, F., Srinivasagan, R., and Abawajy, J. (2025). Efficient delay-sensitive task offloading to fog computing with multi-agent twin delayed deep deterministic policy gradient. *Electronics*, 14(11).

Aqib, M., Kumar, D., and Tripathi, S. (2023). Machine Learning for Fog Computing: Review, Opportunities and a Fog Application Classifier and Scheduler. *Wirel. Pers. Commun.*, 129(2):853–880.

Baek, J. and Kaddoum, G. (2021a). Heterogeneous Task Offloading and Resource Allocations via Deep Recurrent Reinforcement Learning in Partial Observable Multifog Networks. *IEEE Internet Things J.*, 8(2):1041–1056.

Baek, J. and Kaddoum, G. (2021b). Heterogeneous task offloading and resource allocations via deep recurrent reinforcement learning in partial observable multifog networks. *IEEE Internet of Things Journal*, 8(2):1041–1056.

Baek, J. Y., Kaddoum, G., Garg, S., Kaur, K., and Gravel, V. (2019). Managing Fog Networks using Reinforcement Learning Based Load Balancing Algorithm. *IEEE Wirel. Commun. Netw. Conf. WCNC*, 2019-April:1–7.

Bai, Y., Li, X., Wu, X., and Zhou, Z. (2023). Dynamic Computation Offloading with Deep Reinforcement Learning in Edge Network. *Appl. Sci.*, 13(3).

Benrazek, A. E., Farou, B., Kouahla, Z., Ferrag, M. A., and Seridi, H. (2023). IoVT-based efficient solution for optimal active smart camera selection in a tracking mission. *Internet of Things (Netherlands)*, 24(July).

Benrazek, A. E., Kouahla, Z., Farou, B., Ferrag, M. A., Seridi, H., and Kurulay, M. (2020). An efficient indexing for Internet of Things massive data based on cloud-fog computing. *Trans. Emerg. Telecommun. Technol.*, 31(3):1–21.

Bi, S., Huang, L., Wang, H., and Zhang, Y. J. A. (2021). Lyapunov-guided deep reinforcement learning for stable online computation offloading in mobile-edge computing networks. *IEEE Trans. Wirel. Commun.*, 20(11):7519–7537.

Bonomi, F., Milito, R., Zhu, J., and Addepalli, S. (2012). Fog computing and its role in the internet of things. *MCC'12 - Proc. 1st ACM Mob. Cloud Comput. Work.*, pages 13–15.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). Openai gym. *arXiv preprint arXiv:1606.01540*.

Canese, L., Farinelli, A., and Lippi, M. (2021). Multi-agent deep reinforcement learning: A review of challenges and applications. *Applied Sciences*, 11(11):4948.

Cao, H. and Cai, J. (2018). Distributed multiuser computation offloading for cloudlet-based mobile cloud computing: A game-theoretic machine learning approach. *IEEE Trans. Veh. Technol.*, 67(1):752–764.

Cao, Z., Zhou, P., Li, R., Huang, S., and Wu, D. (2020). Multiagent Deep Reinforcement Learning for Joint Multichannel Access and Task Offloading of Mobile-Edge Computing in Industry 4.0. *IEEE Internet Things J.*, 7(7):6201–6213.

Cavazzuti, M. (2013). Optimization methods: From theory to design. In *Optimization Theory*, chapter Design of Experiments. Springer.

Chen, J., Chen, P., Niu, X., Wu, Z., Xiong, L., and Shi, C. (2022). Task offloading in hybrid-decision-based multi-cloud computing network: a cooperative multi-agent deep reinforcement learning. *J. Cloud Comput.*, 11(1).

Cheng, X., Lyu, F., Quan, W., Zhou, C., He, H., Shi, W., and Shen, X. (2019). Space/Aerial-Assisted Computing Offloading for IoT Applications: A Learning-Based Approach. *IEEE J. Sel. Areas Commun.*, 37(5):1117–1129.

Cho, B. and Xiao, Y. (2021). Learning-based decentralized offloading decision making in an adversarial environment. *IEEE Trans. Veh. Technol.*, 70(11):11308–11323.

Davoli, G., Cerroni, W., Borsatti, D., Valieri, M., Tarchi, D., and Raffaelli, C. (2022). A fog computing orchestrator architecture with service model awareness. *IEEE Transactions on Network and Service Management*, 19(3):2131–2147.

Dong, C. and Wen, W. (2019). Joint optimization for task offloading in edge computing: An evolutionary game approach. *Sensors (Switzerland)*, 19(3).

Fahimullah, M., Ahvar, S., Agarwal, M., and Trocan, M. (2023). Machine learning-based solutions for resource management in fog computing. *Multimed. Tools Appl.*

Foerster, J., Farquhar, G., Afouras, T., Nardelli, N., and Whiteson, S. (2018). Counterfactual multi-agent policy gradients. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32.

Fu, X., Tang, B., Guo, F., and Kang, L. (2021). Priority and dependency-based dag tasks offloading in fog/edge collaborative environment. In *2021 IEEE 24th international conference on computer supported cooperative work in design (CSCWD)*, pages 440–445. IEEE.

Fujimoto, S., Hoof, H., and Meger, D. (2018). Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pages 1587–1596. PMLR.

Gao, Z., Yang, L., and Dai, Y. (2022). Fast Adaptive Task Offloading and Resource Allocation via Multi-agent Reinforcement Learning in Heterogeneous Vehicular Fog Computing. *IEEE Internet Things J.*, 10(8):6818–6835.

Gebrekidan, T. Z., Stein, S., and Norman, T. J. (2024). Combinatorial Client-Master Multiagent Deep Reinforcement Learning for Task Offloading in Mobile Edge Computing. *Proc. Int. Jt. Conf. Auton. Agents Multiagent Syst. AAMAS*, 2024-May:2273–2275.

Ghanavati, S., Abawajy, J., and Izadi, D. (2020a). An Energy Aware Task Scheduling Model Using Ant-Mating Optimization in Fog Computing Environment. *IEEE Trans. Serv. Comput.*, pages 1–10.

Ghanavati, S., Abawajy, J., and Izadi, D. (2020b). Automata-based Dynamic Fault Tolerant Task Scheduling Approach in Fog Computing. *IEEE*, 6750(c).

Ghanavati, S., Abawajy, J., and Izadi, D. (2022). An energy aware task scheduling model using ant-mating optimization in fog computing environment. *IEEE Transactions on Services Computing*, 15(4):2007–2017.

Gulde, R., Tuscher, M., Csiszar, A., Riedel, O., and Verl, A. (2020). Deep reinforcement learning using cyclical learning rates. In *2020 Third International Conference on Artificial Intelligence for Industries (AI4I)*, pages 32–35. IEEE.

Guo, M., Mukherjee, M., Liang, G., and Zhang, J. (2020). Computation Offloading for Machine Learning in Industrial Environments. *IECON Proc. (Industrial Electron. Conf.*, 2020-Octob:4465–4470.

Guo, Y., Ning, Z., and Kwok, R. (2019). Deep Reinforcement Learning Based Traffic Offloading Scheme for Vehicular Networks. *2019 IEEE 5th Int. Conf. Comput. Commun. ICCC 2019*, pages 81–85.

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

Hao, H., Xu, C., Zhang, W., Yang, S., and Muntean, G.-M. (2024). Joint task offloading, resource allocation, and trajectory design for multi-uav cooperative edge computing with task priority. *IEEE Transactions on Mobile Computing*, 23(9):8649–8663.

Hazra, A., Rana, P., Adhikari, M., and Amgoth, T. (2023). Fog computing for next-generation Internet of Things: Fundamental, state-of-the-art and research challenges. *Comput. Sci. Rev.*, 48.

He, J., Wei, J., Chen, K., Tang, Z., Zhou, Y., and Zhang, Y. (2018). Multitier Fog Computing With Large-Scale IoT Data Analytics for Smart Cities. *IEEE Internet Things J.*, 5(2):677–686.

He, X., Lu, H., Huang, H., Mao, Y., Wang, K., and Guo, S. (2020). QoE-based Cooperative Task Offloading with Deep Reinforcement Learning in Mobile Edge Networks. *IEEE Wirel. Commun.*, pages 2–8.

Hong, C. H. and Varghese, B. (2019). Resource management in fog/Edge computing: A survey on architectures, infrastructure, and algorithms. *ACM Comput. Surv.*, 52(5).

Hortelano, D., de Miguel, I., Barroso, R. J., Aguado, J. C., Merayo, N., Ruiz, L., Asensio, A., Masip-Bruin, X., Fernández, P., Lorenzo, R. M., and Abril, E. J. (2023). A comprehensive survey on reinforcement-learning-based computation offloading techniques in Edge Computing Systems. *J. Netw. Comput. Appl.*, 216(October 2022):103669.

Hou, X., Ren, Z., Wang, J., Zheng, S., Cheng, W., and Zhang, H. (2020). Distributed Fog Computing for Latency and Reliability Guaranteed Swarm of Drones. *IEEE Access*, 8:7117–7130.

Huang, L., Feng, X., Feng, A., Huang, Y., and Qian, L. P. (2018). Distributed Deep Learning-based Offloading for Mobile Edge Computing Networks. *Mob. Networks Appl.*

Huang, W., Han, Z., Zhao, L., Xu, H., Li, Z., and Wang, Z. (2021). Resource allocation for intelligent reflecting surfaces assisted federated learning system with imperfect csi. *Algorithms*, 14(12):1–13.

Hussein, M. K. and Mousa, M. H. (2020). Efficient task offloading for IoT-Based applications in fog computing using ant colony optimization. *IEEE Access*, 8:37191–37201.

Ibrar, M., Akbar, A., Jan, S. R. U., Jan, M. A., Wang, L., Song, H., and Shah, N. (2022). ARTNet: Ai-Based Resource Allocation and Task Offloading in a Reconfigurable Internet of Vehicular Networks. *IEEE Trans. Netw. Sci. Eng.*, 9(1):67–77.

Jalali Khalil Abadi, Z., Mansouri, N., and Khalouie, M. (2023). Task scheduling in fog environment — challenges, tools and methodologies: A review. *Computer Science Review*, 48:100550.

Jamil, B., Ijaz, H., Shojafar, M., and Munir, K. (2023). Irats: A drl-based intelligent priority and deadline-aware online resource allocation and task scheduling algorithm in a vehicular fog network. *Ad hoc networks*, 141:103090.

Jan, S. U., Ahmed, S., Shakhov, V., and Koo, I. (2019). Toward a Lightweight Intrusion Detection System for the Internet of Things. *IEEE Access*, 7(c):42450–42471.

Javeed, D., Saeed, M. S., Ahmad, I., Kumar, P., Jolfaei, A., and Tahir, M. (2023). An Intelligent Intrusion Detection System for Smart Consumer Electronics Network. *IEEE Trans. Consum. Electron.*, PP:1.

Jiang, Q., Xu, X., He, Q., Zhang, X., Dai, F., Qi, L., and Dou, W. (2021). Game theory-based task offloading and resource allocation for vehicular networks in edge-cloud computing. In *2021 IEEE International Conference on Web Services (ICWS)*, pages 341–346. IEEE.

Jošilo, S. and Dán, G. (2018). Decentralized algorithm for randomized task allocation in fog computing systems. *IEEE/ACM Trans. Netw.*, 27(1):85–97.

Kar, B., Yahya, W., Lin, Y. D., and Ali, A. (2023). Offloading Using Traditional Optimization and Machine Learning in Federated Cloud-Edge-Fog Systems: A Survey. *IEEE Commun. Surv. Tutorials*, 25(2):1199–1226.

Ke, H., Wang, J., Wang, H., and Ge, Y. (2019). Joint Optimization of Data Offloading and Resource Allocation with Renewable Energy Aware for IoT Devices: A Deep Reinforcement Learning Approach. *IEEE Access*, 7:179349–179363.

Khan, I., Tao, X., Shafiqur Rahman, G. M., Rehman, W. U., and Salam, T. (2020). Advanced Energy-Efficient Computation Offloading Using Deep Reinforcement Learning in MTC Edge Computing. *IEEE Access*, 8:82867–82875.

Lahmar, I. B. and Boukadi, K. (2020). Resource Allocation in Fog Computing: A Systematic Mapping Study. *2020 5th Int. Conf. Fog Mob. Edge Comput. FMEC 2020*, pages 86–93.

Lan, D., Taherkordi, A., Eliassen, F., and Horn, G. (2019). A survey on fog programming: Concepts, state-of-the-art, and research challenges. *DFSD 2019 - Proc. 2019 2nd Int. Work. Distrib. Fog Serv. Des. Part Middlew. 2019*, pages 1–6.

Laroui, M., Nour, B., Moungla, H., Cherif, M. A., Afifi, H., and Guizani, M. (2021). Edge and fog computing for iot: A survey on current research activities & future directions. *Computer Communications*, 180:210–231.

Le, M., Huynh-The, T., Do-Duy, T., Vu, T.-H., Hwang, W.-J., and Pham, Q.-V. (2024). Applications of distributed machine learning for the internet-of-things: A comprehensive survey. *IEEE Communications Surveys & Tutorials*.

Lee, K., Lee, J., Lim, H., Park, S., and Kim, Y. (2021). Partially observable multi-agent reinforcement learning: Methods and applications. In *2021 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE.

Lee, S. S. and Lee, S. (2020). Resource Allocation for Vehicular Fog Computing Using Reinforcement Learning Combined with Heuristic Information. *IEEE Internet Things J.*, 7(10):10450–10464.

Li, L., Ota, K., and Dong, M. (2018). Deep Learning for Smart Industry: Efficient Manufacture Inspection System with Fog Computing. *IEEE Trans. Ind. Informatics*, 14(10):4665–4673.

Li, Y., Liang, L., Fu, J., and Wang, J. (2022). Multiagent Reinforcement Learning for Task Offloading of Space/Aerial-Assisted Edge Computing. *Secur. Commun. Networks*, 2022.

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

Lim, J. (2022). Latency-aware task scheduling for iot applications based on artificial intelligence with partitioning in small-scale fog computing environments. *Sensor*.

Liu, M. and Liu, Y. (2018). Price-Based Distributed Offloading for Mobile-Edge Computing with Computation Capacity Constraints. *IEEE Wirel. Commun. Lett.*, 7(3):420–423.

Liu, Y., Richard Yu, F., Li, X., Ji, H., and Leung, V. C. (2018). Distributed Resource Allocation and Computation Offloading in Fog and Cloud Networks with Non-Orthogonal Multiple Access. *IEEE Trans. Veh. Technol.*, 67(12):12137–12151.

Lu, R., Li, Y. C., Li, Y., Jiang, J., and Ding, Y. (2020). Multi-agent deep reinforcement learning based demand response for discrete manufacturing systems energy management. *Appl. Energy*, 276(June):115473.

Luong, N. C., Jiao, Y., Wang, P., Niyato, D., Kim, D. I., and Han, Z. (2020). A Machine-Learning-Based Auction for Resource Trading in Fog Computing. *IEEE Commun. Mag.*, 58(3):82–88.

Markus, A. and Kertesz, A. (2020). A survey and taxonomy of simulation environments modelling fog computing. *Simul. Model. Pract. Theory*, 101(December 2019):102042.

Mebrek, A. and Yassine, A. (2021). Intelligent resource allocation and task offloading model for iot applications in fog networks: a game-theoretic approach. *IEEE Transactions on Emerging Topics in Computational Intelligence*.

Min, M., Wan, X., Xiao, L., Chen, Y., Xia, M., Wu, D., and Dai, H. (2019a). Learning-based privacy-aware offloading for healthcare IoT with energy harvesting. *IEEE Internet Things J.*, 6(3):4307–4316.

Min, M., Xiao, L., Chen, Y., Cheng, P., Wu, D., and Zhuang, W. (2019b). Learning-Based Computation Offloading for IoT Devices with Energy Harvesting. *IEEE Trans. Veh. Technol.*, 68(2):1930–1941.

Mouradian, C., Naboulsi, D., Yangui, S., Glitho, R. H., Morrow, M. J., and Polakos, P. A. (2018). A Comprehensive Survey on Fog Computing: State-of-the-Art and Research Challenges. *IEEE Commun. Surv. Tutorials*, 20(1):416–464.

Mukherjee, M., Shu, L., and Wang, D. (2018). Survey of fog computing: Fundamental, network applications, and research challenges. *IEEE Commun. Surv. Tutorials*, 20(3):1826–1857.

Nguyen, T. T., Nguyen, N. D., and Nahavandi, S. (2020). Deep reinforcement learning for multi-agent systems: A review of challenges, solutions and applications. *IEEE Transactions on Cybernetics*, 50(9):3826–3839.

Ning, Z., Dong, P., Wang, X., Guo, L., Rodrigues, J. J., Kong, X., Huang, J., and Kwok, R. Y. (2019). Deep Reinforcement Learning for Intelligent Internet of Vehicles: An Energy-Efficient Computational Offloading Scheme. *IEEE Trans. Cogn. Commun. Netw.*, 5(4):1060–1072.

Oroojlooy, A. and Hajinezhad, D. (2023). A review of cooperative multi-agent deep reinforcement learning. *Autonomous Agents and Multi-Agent Systems*, 37(1):1–54.

Pu, L., Chen, X., Xu, J., and Fu, X. (2016). D2D Fogging: An Energy-Efficient and Incentive-Aware Task Offloading Framework via Network-Assisted D2D Collaboration. *IEEE J. Sel. Areas Commun.*, 34(12):3887–39014.

Puliafito, C., Mingozzi, E., Longo, F., Puliafito, A., and Rana, O. (2019). Fog computing for the Internet of Things: A survey. *ACM Trans. Internet Technol.*, 19(2).

Qi, J., Zhou, Q., Lei, L., and Zheng, K. (2021). Federated reinforcement learning: techniques, applications, and open challenges . *Intell. Robot.*, pages 1–35.

Qi, Q., Wang, J., Ma, Z., Sun, H., Cao, Y., Zhang, L., and Liao, J. (2019). Knowledge-Driven Service Offloading Decision for Vehicular Edge Computing: A Deep Reinforcement Learning Approach. *IEEE Trans. Veh. Technol.*, 68(5):4192–4203.

Qiu, X., Zhang, W., Chen, W., and Zheng, Z. (2021). Distributed and Collective Deep Reinforcement Learning for Computation Offloading: A Practical Perspective. *IEEE Trans. Parallel Distrib. Syst.*, 32(5):1085–1101.

Rahman, G. S., Dang, T., and Ahmed, M. (2020). Deep reinforcement learning based computation offloading and resource allocation for low-latency fog radio access networks. *Intelligent and Converged Networks*, 1(3):243–257.

Raju, M. R., Mothku, S. K., and Somesula, M. K. (2024). Dmits: Dependency and mobility-aware intelligent task scheduling in socially-enabled vfc based on federated drl approach. *IEEE Transactions on Intelligent Transportation Systems*.

Rani, D. S. and Pounambal, M. (2021). Deep learning based dynamic task offloading in mobile cloudlet environments. *Evol. Intell.*, 14(2):499–507.

Rapoport, A. (2012). *Game theory as a theory of conflict resolution*, volume 2. Springer Science & Business Media.

Rejiba, Z., Masip-Bruin, X., and Marín-Tordera, E. (2019). A survey on mobility-induced service migration in the fog, edge, and related computing paradigms. *ACM Comput. Surv.*, 52(5).

Ren, J., Zhang, D., He, S., Zhang, Y., and Li, T. (2020a). A Survey on End-Edge-Cloud Orchestrated Network Computing Paradigms. *ACM Comput. Surv.*, 52(6):1–36.

Ren, Y., Sun, Y., and Peng, M. (2020b). Deep Reinforcement Learning Based Computation Offloading in Fog Enabled Industrial Internet of Things. *IEEE Trans. Ind. Informatics*, 3203(c):1–1.

Rihan, M., Elwekeil, M., Yang, Y., Huang, L., Xu, C., and Selim, M. M. (2020). Deep-VFog: When Artificial Intelligence Meets Fog Computing in V2X. *IEEE Syst. J.*, 15(3):3492–3505.

Rodrigues, T. K., Suto, K., Nishiyama, H., Liu, J., and Kato, N. (2020). Machine Learning Meets Computation and Communication Control in Evolving Edge and Cloud: Challenges and Future Perspective. *IEEE Commun. Surv. Tutorials*, 22(1):38–67.

Salaht, F. A., Desprez, F., and Lebre, A. (2020). An Overview of Service Placement Problem in Fog and Edge Computing. *ACM Comput. Surv.*, 53(3):1–35.

Sami, H., Mourad, A., Otrok, H., and Bentahar, J. (2021). Demand-Driven Deep Reinforcement Learning for Scalable Fog and Service Placement. *IEEE Trans. Serv. Comput.*, 15.

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

Seid, A. M., Boateng, G. O., Mareri, B., Sun, G., and Jiang, W. (2021). Multi-Agent DRL for Task Offloading and Resource Allocation in Multi-UAV Enabled IoT Edge Network. *IEEE Trans. Netw. Serv. Manag.*, 18(4):4531–4547.

Seid, A. M., Erbad, A., Abishu, H. N., Albaseer, A., Abdallah, M., and Guizani, M. (2023). Multi-agent Federated Reinforcement Learning for Resource Allocation in UAV-enabled Internet of Medical Things Networks. *IEEE Internet Things J.*, PP:1.

Sellami, B., Hakiri, A., Yahia, S. B., and Berthou, P. (2022). Energy-aware task scheduling and offloading using deep reinforcement learning in SDN-enabled IoT network. *Comput. Networks*, 210.

Shah-Mansouri, H. and Wong, V. W. (2018). Hierarchical fog-cloud computing for IoT systems: A computation offloading game. *IEEE Internet Things J.*, 5(4):3246–3257.

Shahryari, O.-K., Pedram, H., Khajehvand, V., and TakhtFooladi, M. D. (2021). Energy and task completion time trade-off for task offloading in fog-enabled iot networks. *Pervasive and Mobile Computing*, 74:101395.

Shakarami, A., Ghobaei-Arani, M., and Shahidinejad, A. (2020). A survey on the computation offloading approaches in mobile edge computing: A machine learning-based perspective. *Comput. Networks*, 182(August):107496.

Shannon, C. E. (1948). A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423.

Shi, J., Du, J., Wang, J., Wang, J., and Yuan, J. (2020). Priority-Aware Task Offloading in Vehicular Fog Computing Based on Deep Reinforcement Learning. *IEEE Trans. Veh. Technol.*, 69(12):16067–16081.

Shi, J., Du, J., Wang, J., and Yuan, J. (2021). Deep reinforcement learning-based v2v partial computation offloading in vehicular fog computing. In *2021 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1–6. IEEE.

Shi, W., Chen, L., and Zhu, X. (2023). Task offloading decision-making algorithm for vehicular edge computing: a deep-reinforcement-learning-based approach. *Sensors*, 23(17):7595.

Singh, S. P., Nayyar, A., Kumar, R., and Sharma, A. (2019). Fog computing: from architecture to edge computing and big data processing. *The Journal of Supercomputing*, 75:2070–2105.

Songhorabadi, M., Rahimi, M., MoghadamFarid, A., and Kashani, M. H. (2023). Fog computing approaches in iot-enabled smart cities. *Journal of Network and Computer Applications*, 211:103557.

Suzuki, A., Kobayashi, M., and Oki, E. (2023). Multi-Agent Deep Reinforcement Learning for Cooperative Computing Offloading and Route Optimization in Multi Cloud-Edge Networks. *IEEE Trans. Netw. Serv. Manag.*, pages 3660–3666.

Tadele, S. B., Yahya, W., Kar, B., Lin, Y.-D., Lai, Y.-C., and Wakgra, F. G. (2024). Optimizing the ratio-based offloading in federated cloud-edge systems: A madrl approach. *IEEE Transactions on Network Science and Engineering*.

Tan, Y., Wang, K., Yang, Y., and Zhou, M. T. (2019). Delay-Optimal Task Offloading for Dynamic Fog Networks. *IEEE Int. Conf. Commun.*, 2019-May:1–6.

Tang, Q., Xie, R., Yu, F. R., Huang, T., and Liu, Y. (2020). Decentralized computation offloading in IoT fog computing system with energy harvesting: A dec-POMDP approach. *IEEE Internet Things J.*, 7(6):4898–4911.

Tong, Z., Li, Z., Gendia, A., and Muta, O. (2024). Deep reinforcement learning based computing resource allocation in fog radio access networks. In *2024 IEEE 100th Vehicular Technology Conference (VTC2024-Fall)*, pages 1–5. IEEE.

Tran-Dang and Kim (2023). DISCO: Distributed computation offloading framework for fog computing networks. *Journal of Communications and Networks*, 25(1):121–131.

Tran-Dang, H., Bhardwaj, S., Rahim, T., Musaddiq, A., and Kim, D.-S. (2022). Reinforcement learning based resource management for fog computing environment: Literature review, challenges, and open issues. *J. Commun. Networks*, 24(1):83–98.

Tuli, S., Basumatary, N., and Buyya, R. (2019). EdgeLens: Deep learning based object detection in integrated IoT, fog and cloud computing environments. *IEEE*, pages 496–502.

Tuli, S., Basumatary, N., Gill, S. S., Kahani, M., Arya, R. C., Wander, G. S., and Buyya, R. (2020). HealthFog: An ensemble deep learning based Smart Healthcare System for Automatic Diagnosis of Heart Diseases in integrated IoT and fog computing environments. *Futur. Gener. Comput. Syst.*, 104:187–200.

Tuli, S., Mirhakimi, F., Pallewatta, S., Zawad, S., Casale, G., Javadi, B., Yan, F., Buyya, R., and Jennings, N. R. (2023). AI augmented Edge and Fog computing: Trends and challenges. *J. Netw. Comput. Appl.*, 216(October 2022):103648.

Verma, K., Kumar, A., Ul Islam, M. S., Kanwar, T., and Bhushan, M. (2021). Rank based mobility-aware scheduling in Fog computing. *Informatics Med. Unlocked*, 24(April):100619.

Wahab, O. A., Mourad, A., Otrok, H., and Taleb, T. (2021). Federated machine learning: Survey, multi-level classification, desirable criteria and future directions in communication and networking systems. *IEEE Communications Surveys and Tutorials*, 23(2):1342–1397.

Wakgra, F. G., Kar, B., Tadele, S. B., Shen, S.-H., and Khan, A. U. (2024a). Multi-objective offloading optimization in mec and vehicular-fog systems: A distributed-td3 approach. *IEEE Transactions on Intelligent Transportation Systems*.

Wakgra, F. G., Yahya, W., Kar, B., Lai, Y.-C., Lin, Y.-D., and Tadele, S. B. (2024b). Ratio-based offloading optimization for edge and vehicular-fog federated systems: a multi-agent td3 approach. *IEEE Transactions on Vehicular Technology*.

Wang, D., Liu, Z., Wang, X., and Lan, Y. (2019a). Mobility-Aware Task Offloading and Migration Schemes in Fog Computing Networks. *IEEE Access*, 7:43356–43368.

Wang, J., Lv, T., Huang, P., and Mathiopoulos, P. T. (2020). Mobility-aware partial computation offloading in vehicular networks: A deep reinforcement learning based scheme. *China Commun.*, 17(10):31–49.

Wang, J., Wu, W., Liao, Z., Sangaiah, A. K., and Simon Sherratt, R. (2019b). An energy-efficient off-loading scheme for low latency in collaborative edge computing. *IEEE Access*, 7:149182–149190.

Wang, K., Tan, Y., Shao, Z., Ci, S., and Yang, Y. (2019c). Learning-Based Task Offloading for Delay-Sensitive Applications in Dynamic Fog Networks. *IEEE Trans. Veh. Technol.*, 68(11):11399–11403.

Wang, K., Zhou, Y., Yang, Y., Yuan, X., and Luo, X. (2019d). Task offloading in NOMA-based fog computing networks: A deep Q-learning approach. *2019 IEEE Glob. Commun. Conf. GLOBECOM 2019 - Proc.*

Wang, N. and Varghese, B. (2022). Context-aware distribution of fog applications using deep reinforcement learning. *Journal of Network and Computer Applications*, 203:103354.

Wang, S., Hu, Z., Deng, Y., and Hu, L. (2022). Game-theory-based task offloading and resource scheduling in cloud-edge collaborative systems. *Applied Sciences*, 12(12):6154.

Wang, Y., Wang, K., Huang, H., Miyazaki, T., and Guo, S. (2019e). Traffic and Computation Co-Offloading with Reinforcement Learning in Fog Computing for Industrial Applications. *IEEE Trans. Ind. Informatics*, 15(2):976–986.

Waqas, M., Niu, Y., Ahmed, M., Li, Y., Jin, D., and Han, Z. (2019). Mobility-Aware Fog Computing in Dynamic Environments: Understandings and Implementation. *IEEE Access*, 7:38867–38879.

Wei, Y., Yu, F. R., Song, M., and Han, Z. (2019). Joint optimization of caching, computing, and radio resources for fog-enabled IoT using natural actor-critic deep reinforcement learning. *IEEE Internet Things J.*, 6(2):2061–2073.

Wei, Z., Li, B., Zhang, R., Cheng, X., and Yang, L. (2023). Many-to-Many Task Offloading in Vehicular Fog Computing: A Multi-Agent Deep Reinforcement Learning Approach. *IEEE Trans. Mob. Comput.*, XX(Xx):1–16.

Wu, G., Xu, Z., Zhang, H., Shen, S., and Yu, S. (2023). Multi-agent drl for joint completion delay and energy consumption with queuing theory in mec-based iiot. *Journal of Parallel and Distributed Computing*, 176:80–94.

Xu, C. and Zhu, G. (2021). Intelligent manufacturing Lie Group Machine Learning: real-time and efficient inspection system based on fog computing. *J. Intell. Manuf.*, 32(1):237–249.

Xu, X., Shen, B., Ding, S., Srivastava, G., Bilal, M., Khosravi, M. R., Menon, V. G., Jan, M. A., and Wang, M. (2022). Service Offloading with Deep Q-Network for Digital Twinning-Empowered Internet of Vehicles in Edge Computing. *IEEE Trans. Ind. Informatics*, 18(2):1414–1423.

Yang, B., Cao, X., Bassey, J., Li, X., and Qian, L. (2021). Computation Offloading in Multi-Access Edge Computing: A Multi-Task Learning Approach. *IEEE Trans. Mob. Comput.*, 20(9):2745–2762.

Yang, M., Zhu, H., Wang, H., Koucheryavy, Y., Samouylov, K., and Qian, H. (2020). An Online Learning Approach to Computation Offloading in Dynamic Fog Networks. *IEEE Internet Things J.*, 4662(c):1–13.

Yousefpour, A., Patil, A., Ishigaki, G., Kim, I., Wang, X., Cankaya, H. C., Zhang, Q., Xie, W., and Jue, J. P. (2019). FOGPLAN: A lightweight QoS-aware dynamic fog service provisioning framework. *IEEE Internet Things J.*, 6(3):5080–5096.

Yu, S., Chen, X., Zhou, Z., Gong, X., and Wu, D. (2021). When Deep Reinforcement Learning Meets Federated Learning: Intelligent Multitimescale Resource Management for Multiaccess Edge Computing in 5G Ultradense Network. *IEEE Internet Things J.*, 8(4):2238–2251.

Zabihi, Z., Eftekhari Moghadam, A. M., and Rezvani, M. H. (2023). Reinforcement learning methods for computation offloading: A systematic review. *ACM Comput. Surv.*, 56(1).

Zhang, D., Cao, L., Zhu, H., Zhang, T., Du, J., and Jiang, K. (2022a). Task offloading method of edge computing in internet of vehicles based on deep reinforcement learning. *Cluster Comput.*, 9.

Zhang, J., Du, J., Shen, Y., and Wang, J. (2020). Dynamic Computation Offloading with Energy Harvesting Devices: A Hybrid-Decision-Based Deep Reinforcement Learning Approach. *IEEE Internet Things J.*, 7(10):9303–9317.

Zhang, K., Yang, Z., and Başar, T. (2021a). Multi-agent reinforcement learning: A selective overview of theories and algorithms. *Handbook of reinforcement learning and control*, pages 321–384.

Zhang, K., Yang, Z., and Basar, T. (2021b). A survey on multi-agent reinforcement learning: From collaborative to competitive approaches. *IEEE Transactions on Neural Networks and Learning Systems*, 32(2):738–753.

Zhang, K., Yang, Z., and Başar, T. (2021c). Multi-Agent Reinforcement Learning: A Selective Overview of Theories and Algorithms. *Stud. Syst. Decis. Control*, 325:321–384.

Zhang, K., Zhu, Y., Leng, S., He, Y., Maharjan, S., and Zhang, Y. (2019). Deep Learning Empowered Task Offloading for Mobile Edge Computing in Urban Informatics. *IEEE Internet Things J.*, 6(5):7635–7647.

Zhang, L., Jiang, Y., Zheng, F.-C., Bennis, M., and You, X. (2022b). Computation offloading and resource allocation in f-rans: A federated deep reinforcement learning approach. In *2022 IEEE international conference on communications workshops (ICC Workshops)*, pages 97–102. IEEE.

Zhang, L., Zhang, Z. Y., Min, L., Tang, C., Zhang, H. Y., Wang, Y. H., and Cai, P. (2021d). Task Offloading and Trajectory Control for UAV-Assisted Mobile Edge Computing Using Deep Reinforcement Learning. *IEEE Access*, 9.

Zhang, Y., Deng, S., and Guo, S. (2022c). Deep reinforcement learning for task offloading in fog computing: A comprehensive survey. *IEEE Communications Surveys & Tutorials*, 24(2):1342–1376.

Zhang, Y., Wang, J., Zhang, L., Zhang, Y., Li, Q., and Chen, K. C. (2023). Reliable Transmission for NOMA Systems With Randomly Deployed Receivers. *IEEE Trans. Commun.*, 71(2):1179–1192.

Zheng, J., Li, K., Mhaisen, N., Ni, W., Tovar, E., and Guizani, M. (2022). Exploring Deep Reinforcement Learning-Assisted Federated Learning for Online Resource Allocation in EdgeIoT. *arXiv*, pages 1–12.

Zhou, Z. and Abawajy, J. (2025). Reinforcement learning-based edge server placement in the intelligent internet of vehicles environment. *IEEE Transactions on Intelligent Transportation Systems*, pages 1–11.

Zhu, D., Liu, H., Li, T., Sun, J., Liang, J., Zhang, H., Geng, L., and Liu, Y. (2021a). Deep reinforcement learning-based task offloading in satellite-terrestrial edge computing networks. *IEEE Wirel. Commun. Netw. Conf. WCNC*, 2021-March.

Zhu, X., Chen, S., Chen, S., and Yang, G. (2019). Energy and Delay Co-aware Computation Offloading with Deep Learning in Fog Computing Networks. In *2019 IEEE 38th Int. Perform. Comput. Commun. Conf. IPCCC 2019*, pages 1–6. IEEE.

Zhu, X., Luo, Y., Liu, A., Bhuiyan, Z. A., Member, S., and Zhang, S. (2021b). Multiagent Deep Reinforcement Learning for Vehicular Computation Offloading in IoT. *IEE Internet Things*, 8(12):9763–9773.

Chapter A

APPENDIX A: SAMPLE CODE AND FIGURES

The training and performance evaluation sample graphs for each of the baseline algorithms presented in this section. In the simulation, the design have fog-to-fog environments "fogtofog.py", baseline model implementation such as MATD3.py, MAIDDPG.py, MAPPO.py, and MASAC.py. The network model for actor and critic also designed inside network.py. We presented sample "fogtofog.py" sample python code.

```
import numpy as np
import torch

class FogNode:
def __init__(self, task_size, task_deadline, cpu_req, dep_prob, cpu_avail, ram_avail,
offload_succ_rate, bandwidth, net_delay, link_fail_prob, snr, plr, jitter, channel_st
received_power=0.01, noise_power=0.001):
self.task_size = task_size
self.task_deadline = task_deadline
self.cpu_req = cpu_req
self.dep_prob = dep_prob
self.cpu_avail = cpu_avail
self.ram_avail = ram_avail
self.queue_len = queue_len
self.offload_succ_rate = offload_succ_rate
self.bandwidth = bandwidth
self.net_delay = net_delay
self.link_fail_prob = link_fail_prob
self.snr = self.calculate_snr(received_power, noise_power)
self.plr = plr
```

```
self.jitter = jitter
self.channel_status = channel_status  # 1: Free, 0: Busy
self.SPEED_OF_LIGHT_FIBER = 3e8
self.distance = torch.rand(1, device='cpu') * 90 + 10

def calculate_snr(self, received_power, noise_power):
return received_power / noise_power

def calculate_snr_db(self, received_power, noise_power):
return 10 * np.log10(received_power / noise_power)

def can_handle_task(self, task_cpu, task_bandwidth, task_ram):
return (self.cpu_avail >= task_cpu and
self.bandwidth >= task_bandwidth and
self.ram_avail >= task_ram)

def assign_task(self, task_cpu, task_bandwidth, task_ram):
if self.can_handle_task(task_cpu, task_bandwidth, task_ram):
self.cpu_avail -= task_cpu
self.bandwidth -= task_bandwidth
self.ram_avail -= task_ram
return True
return False

def reset(self):
self.cpu_avail = 3.0  # Reset to initial capacity
self.bandwidth = 100   # Reset to initial bandwidth
self.ram_avail = 2.0   # Reset to initial RAM

def calculate_pd(self, distance):
return distance / self.SPEED_OF_LIGHT_FIBER

def calculate_transmission_latency(self, task_size, bandwidth, distance):
pd = self.calculate_pd(distance)
return task_size / bandwidth + pd


class FogToFogEnv:
def __init__(self, num_fog_nodes=6):
```

```
self.num_fog_nodes = num_fog_nodes
self.fog_nodes = [FogNode(0, 0, 0, 0, 3.0, 2.0, 4, 0.8, 100, 30, 0.02, 1.0, 0.01, 1.0
self.task_counter = 0
self.result_ratio = 0.1
self.state = self._generate_initial_state()
self.max_latency = 100  # Placeholder max latency
self.max_energy = 10  # Placeholder max energy
self.max_successful_tasks = 10  # Placeholder max successful tasks
self.smoothed_reward = 0.2

def _generate_initial_state(self):
self.S_channel = np.random.choice([0, 1], size=(self.num_fog_nodes,))
self.S_power = np.random.normal(3.0, 0.5, size=(self.num_fog_nodes,))
self.S_gain = np.random.normal(1.0, 0.2, size=(self.num_fog_nodes,))
self.S_size = np.random.normal(25, 10, size=(self.num_fog_nodes,))
self.S_cycle = np.random.normal(3.0, 0.5, size=(self.num_fog_nodes,))
self.S_ddl = np.random.normal(25, 10, size=(self.num_fog_nodes,))
self.S_res = np.random.normal(2.0, 0.5, size=(self.num_fog_nodes,))
self.S_com = np.random.normal(100, 20, size=(self.num_fog_nodes,))
self.S_epsilon = np.random.normal(0.01, 0.005, size=(self.num_fog_nodes,))
self.S_new_feature1 = np.random.normal(1.0, 0.2, size=(self.num_fog_nodes,))
self.S_new_feature2 = np.random.normal(1.0, 0.2, size=(self.num_fog_nodes,))
self.S_new_feature3 = np.random.normal(1.0, 0.2, size=(self.num_fog_nodes,))
...
```

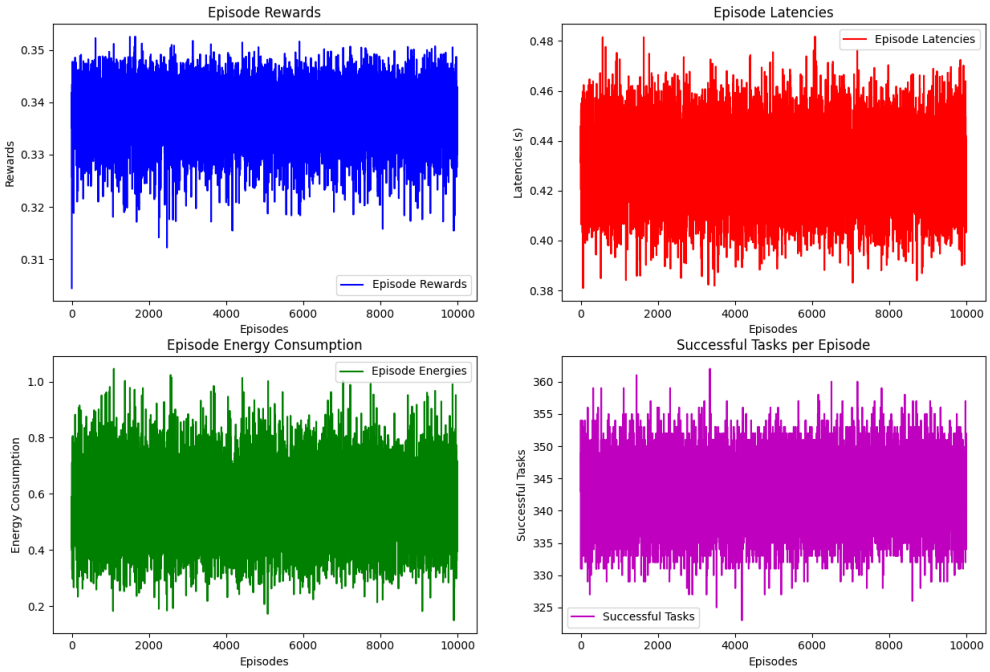

Figure A.1: MAIDDPG Training Result

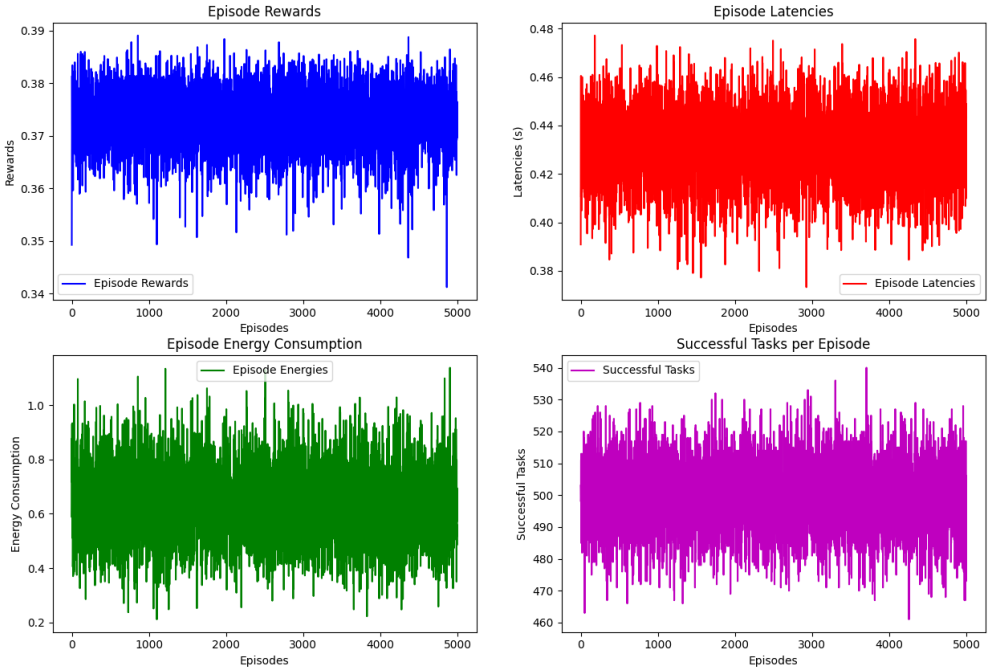

Figure A.2: MAPPO Training Result

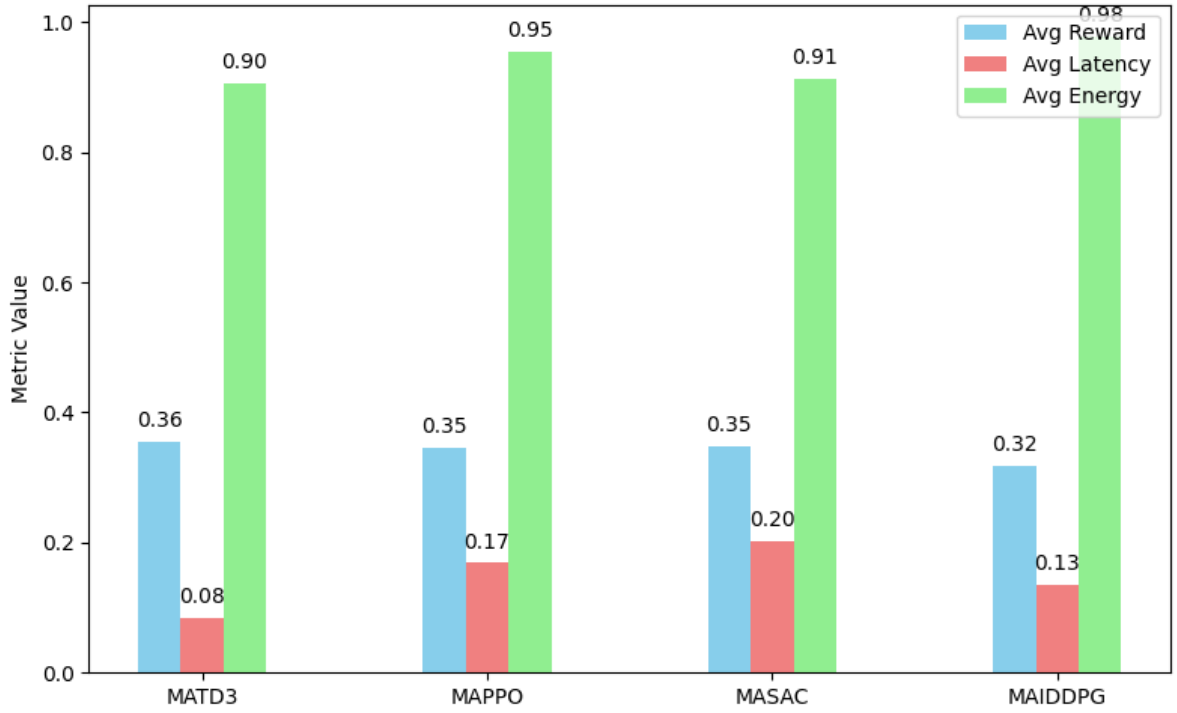

Figure A.3: Baseline algorithm sample comparison

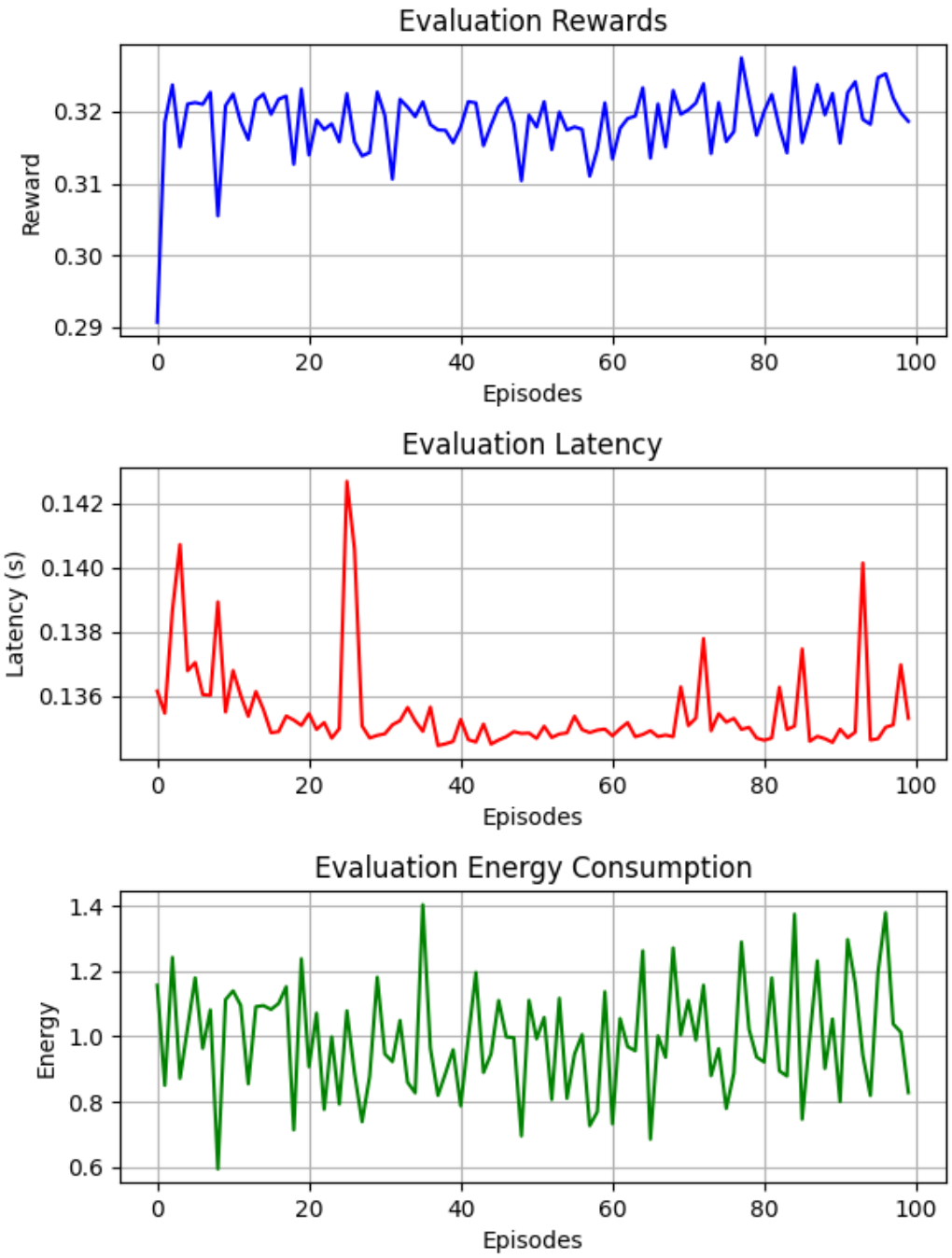

Figure A.4: MAIDDPG Sample evaluation result

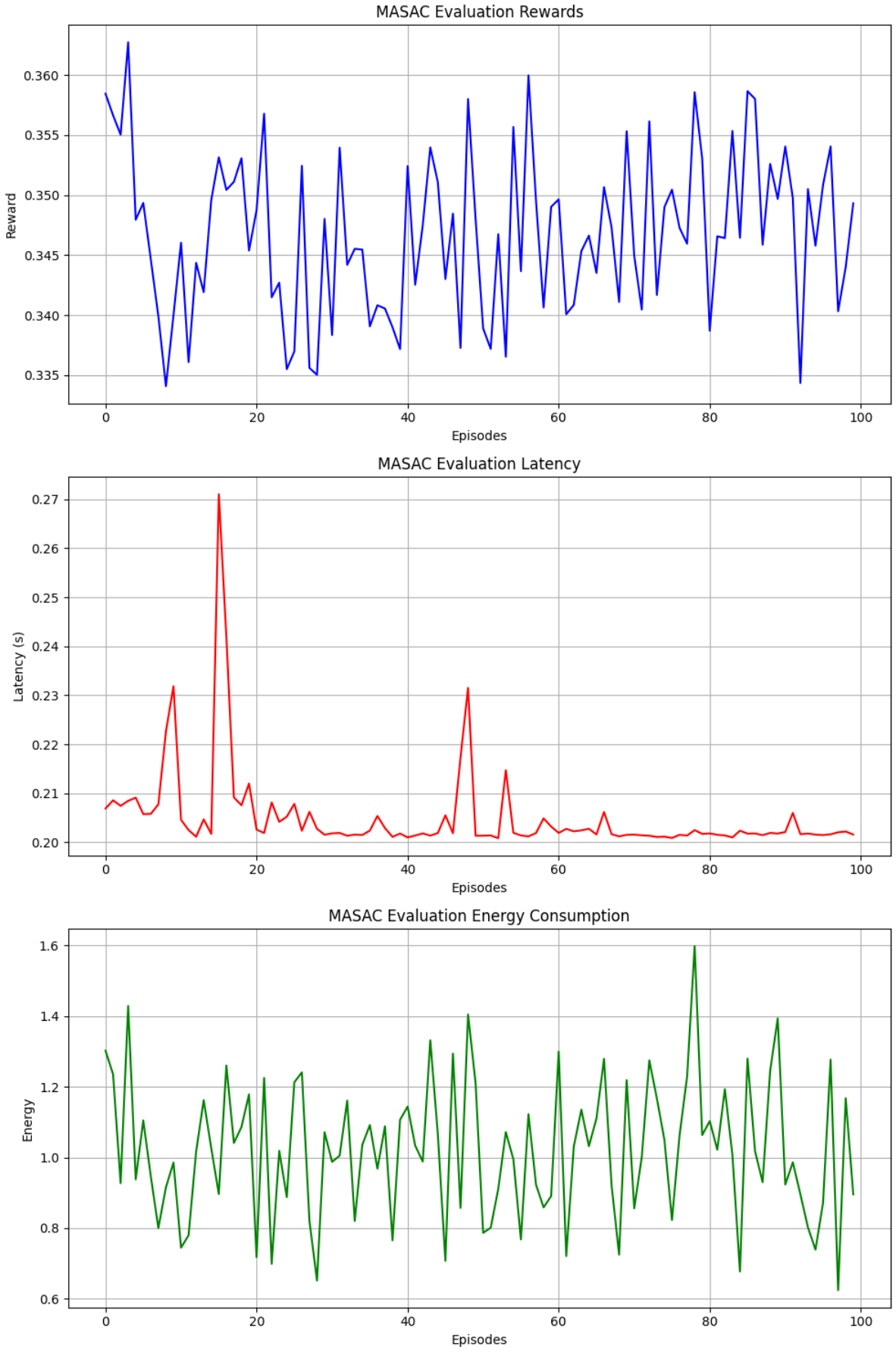

Figure A.5: MASAC Sample evaluation result

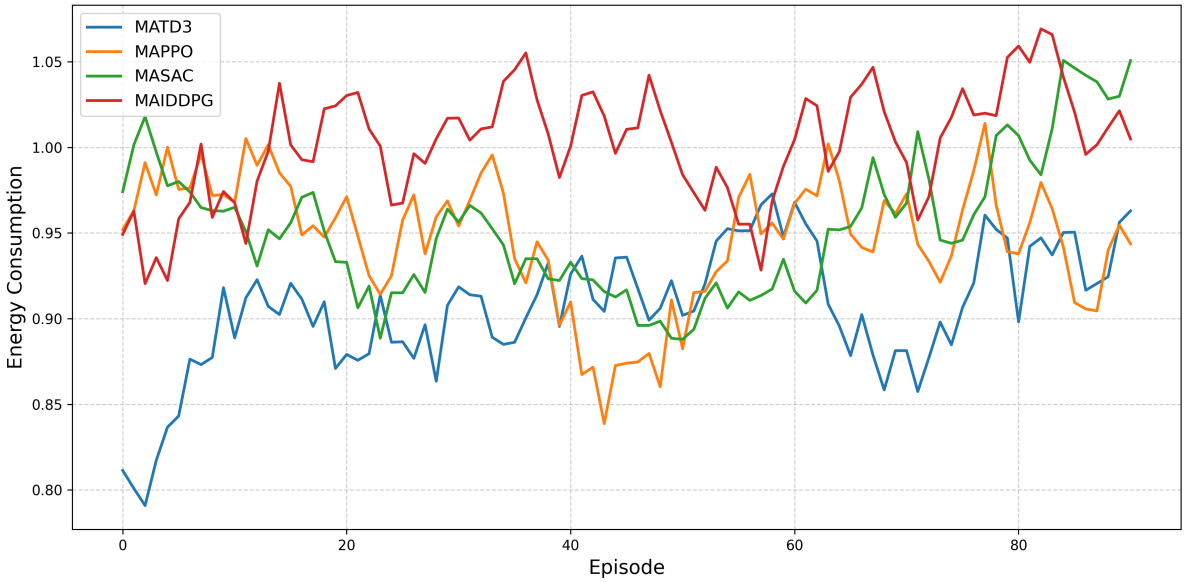

Figure A.6: Smoothed energy consumption graph