Artificial Intelligence (AI)-Driven Resource Management in the IoT-Edge-Cloud Computing Continuum

Zhiyu Wang

Submitted in total fulfilment of the requirements of the degree of

Doctor of Philosophy

School of Computing and Information Systems
THE UNIVERSITY OF MELBOURNE, AUSTRALIA

January 2026

0000-0002-9521-5324

Artificial Intelligence (AI)-Driven Resource Management in the IoT-Edge-Cloud Computing Continuum

Zhiyu Wang
Principal Supervisor: Prof. Rajkumar Buyya
Co-Supervisors: Dr. Mingming Gong and Dr. Mohammad Goudarzi

Abstract

With the exponential proliferation of Internet of Things (IoT) devices and the continuous evolution of wireless connectivity, computing paradigms are undergoing a fundamental shift from solitary cloud centers to the IoT-Edge-Cloud computing continuum. While this continuum significantly expands computational boundaries, its intrinsic multi-dimensional heterogeneity, highly dynamic stochastic workloads, and stringent data privacy constraints impose unprecedented challenges on distributed resource management. Traditional rule-based or static model-driven scheduling approaches struggle to effectively balance the trade-offs between latency, energy consumption, scalability, and security within such massive and complex environments. To address these challenges, this thesis proposes a systematic AI-driven intelligent resource management solution suite, delivering holistic innovations spanning theoretical algorithms, architectural frameworks, and practical deployment to enable adaptive, robust, and scalable resource management across heterogeneous IoT-Edge-Cloud continuum.

The primary research contributions of this thesis are summarized as follows:

1. **Conceptualization of a Systematic Taxonomy:** To resolve the theoretical fragmentation in intelligence resource management in the IoT-Edge-Cloud computing continuum, this thesis establishes a novel orthogonal taxonomy based on control scope and training paradigms. This framework clarifies the design principles and applicability boundaries of DRL architectures, providing a theoretical foundation for the subsequent algorithmic innovations.

2. **Development of Adaptive Optimization for DAG Tasks:** Addressing the stochasticity in edge-fog environments, the DRLIS algorithm is proposed. By employing a customized deep reinforcement learning policy with a composite reward mech-

anism, it achieves an effective trade-off between response time and load balancing for complex DAG-structured IoT applications.

3. **Design of Transformer-Enhanced Distributed Scheduling:** To tackle scalability bottlenecks in large-scale concurrent scenarios, the TF-DDRL technique is engineered. By synergizing attention-based sequence modeling with a high-throughput distributed actor-learner architecture, it effectively captures long-term spatiotemporal dependencies, significantly reducing exploration costs and improving convergence speed in high-throughput environments.

4. **Implementation of a Modular and Extensible Software Platform:** Bridging the gap between algorithmic theory and system deployment, a modular, containerized, and extensible framework, ReinFog, is developed. Furthermore, a novel memetic algorithm (MADCP) is devised to optimize the physical placement of DRL components themselves, minimizing the inherent management overhead of intelligent systems.

5. **Establishment of a Cross-Architecture Collaborative Learning Framework:** To overcome model incompatibility across the Cloud-Edge-IoT continuum, the KD-AFRL framework is proposed. It introduces environment-oriented knowledge distillation to break structural barriers, enabling resource-constrained devices to acquire complex decision-making capabilities through adaptive federated learning.

6. **Construction of a Robust Decentralized Cooperation Protocol:** Addressing data silos and adversarial risks in cross-organizational collaboration, the fully decentralized DeFRiS framework is constructed. Utilizing a dual-track mechanism based on gradient fingerprints and tracking, it guarantees robust convergence and security in Non-IID environments without a central coordinator.

7. **Orchestration of Multi-Agent Coordination in Dynamic Topologies:** Targeting extreme dynamism in aerial computing, the AirFed framework is proposed. It integrates dual-layer dynamic Graph Attention Networks (GATs) with multi-agent reinforcement learning, effectively solving the joint optimization of UAV trajectory control and task offloading under time-varying network topologies.

8. **Synthesis of Key Insights and Future Roadmaps:** Drawing upon the comprehensive investigation across the proposed theoretical models, algorithmic techniques, and system implementations, this thesis synthesizes generalizable design methodologies for the IoT-Edge-Cloud continuum. Furthermore, it identifies pivotal future directions for building ubiquitous, trustworthy, and sustainable next-generation computing infrastructure.

This research establishes a real-world heterogeneous testbed comprising cloud instances (AWS, Azure, Nectar) and various edge and IoT devices to extensively evaluate the proposed frameworks and algorithms. The experimental results validate the efficacy, scalability, and robustness of the proposed AI-driven techniques in handling complex resource management tasks within dynamic environments. This thesis not only achieves innovation at the algorithmic level but also provides a comprehensive system architecture, laying a solid foundation for building the next generation of ubiquitous, intelligent, and trustworthy IoT-Edge-Cloud computing infrastructure.

Declaration

This is to certify that

1. the thesis comprises only my original work towards the PhD,
2. due acknowledgement has been made in the text to all other material used,
3. the thesis is less than 100,000 words in length, exclusive of tables, maps, bibliographies and appendices.

Zhiyu Wang, January 2026

Preface

Main Contributions

This thesis research has been carried out in the Quantum Cloud Computing and Distributed Systems (qCLOUDS) Laboratory, School of Computing and Information Systems, The University of Melbourne. The main contributions of the thesis are discussed in Chapters 2-8 and are based on the following publications:

- **Zhiyu Wang**, Mohammad Goudarzi, Mingming Gong, and Rajkumar Buyya, "Deep Reinforcement Learning for Resource Management in IoT-Edge-Cloud Continuum: A Taxonomy and Future Directions", *ACM Computing Surveys (CSUR)*, 2026 [Under Review].

- **Zhiyu Wang**, Mohammad Goudarzi, Mingming Gong, and Rajkumar Buyya, "Deep Reinforcement Learning-based Scheduling for Optimizing System Load and Response Time in Edge and Fog Computing Environments", *Future Generation Computer Systems (FGCS)*, Volume 152, Pages: 55-69, Elsevier, 2024.

- **Zhiyu Wang**, Mohammad Goudarzi, and Rajkumar Buyya, "TF-DDRL: A Transformer-Enhanced Distributed DRL Technique for Scheduling IoT Applications in Edge and Cloud Computing Environments", *IEEE Transactions on Services Computing (TSC)*, Volume 18, Number 2, Pages: 1039-1053, IEEE, 2025.

- **Zhiyu Wang**, Mohammad Goudarzi, and Rajkumar Buyya, "ReinFog: A Deep Reinforcement Learning Empowered Framework for Resource Management in Edge and Cloud Computing Environments", *Journal of Network and Computer Applica-*

tions (JNCA), Volume 242, Article 104250, Elsevier, 2025.

- **Zhiyu Wang**, Mohammad Goudarzi, Mingming Gong, and Rajkumar Buyya, "A Knowledge Distillation-empowered Adaptive Federated Reinforcement Learning Framework for Multi-Domain IoT Applications Scheduling", *IEEE Transactions on Mobile Computing (TMC)*, 2026 [In Press].

- **Zhiyu Wang**, Mohammad Goudarzi, Mingming Gong, and Rajkumar Buyya, "DeFRiS: Silo-Cooperative IoT Applications Scheduling via Decentralized Federated Reinforcement Learning", *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 2026 [Under Review].

- **Zhiyu Wang**, Suman Raj, and Rajkumar Buyya, "AirFed: Federated Graph-Enhanced Multi-Agent Reinforcement Learning for Multi-UAV Cooperative Mobile Edge Computing", *IEEE Transactions on Mobile Computing (TMC)*, 2026 [Under Review].

Supplementary Contributions

During the PhD candidature, I have also contributed to the following supplemental work (this thesis does not claim those as its main contributions):

- Mohammad Goudarzi, Arash Shaghaghi, **Zhiyu Wang**, and Rajkumar Buyya, "Performance and Security Aware Distributed Service Placement in Fog Computing", *IEEE Transactions on Services Computing (TSC)*, 2026 [Under Review].

- Yulun Huang, **Zhiyu Wang**, and Rajkumar Buyya, "A Risk-Aware UAV-Edge Computing Framework for Wildfire Monitoring and Emergency Response", *IEEE International Conference on Web Services (ICWS)*, 2026 [Under Review].

Acknowledgements

Writing this thesis marks the end of a significant chapter in my life and the beginning of a new journey. This dissertation would not have been possible without the guidance, support, and encouragement of many people who have accompanied me over the past few years.

First and foremost, I would like to express my deepest gratitude to my principal supervisor, *Prof. Rajkumar Buyya*, for offering me the invaluable opportunity to be part of the Quantum Cloud Computing and Distributed Systems (qCLOUDS) Laboratory. His visionary leadership, profound expertise in distributed systems, and trust in my abilities have been the driving force behind this research. I am deeply grateful for the world-class research environment he has cultivated at qCLOUDS, which has allowed me to explore the frontiers of Edge and Cloud computing with freedom and confidence. I am equally indebted to my co-supervisors, *Dr. Mingming Gong* and *Dr. Mohammad Goudarzi*, for their continuous mentorship. I am grateful for their combined insights into Artificial Intelligence methodologies, system design, and practical framework implementation. Their close collaboration, constructive feedback, and availability for discussion provided the solid foundation for this thesis and significantly elevated the quality of this work. I would also like to extend my sincere gratitude to my thesis examiners, *Prof. Carlos Becker Westphall* and *Prof. Leandro Navarro*, for dedicating their time and expertise to the thorough examination of this dissertation and for their generous recognition of this work.

My PhD journey at the qCLOUDS Lab has been enriched by a brilliant and supportive group of colleagues. The daily interactions and technical discussions made the challenges of research enjoyable. I would like to extend my sincere thanks to all past

and current lab members for their friendship and assistance: *Dr. Samodha Pallewatta*, *Dr. Amanda Jayanetti*, *Dr. Anupama Mampage*, *Dr. Tharindu B. Hewage*, *Jie Zhao*, *Ming Chen*, *Siddharth Agarwal*, *Hoa Nguyen*, *Kalyani Pendyala*, *Yulun Huang*, *Duneesha Fernando*, *Thakshila Imiya Mohottige*, *Qifan Deng*, *Tianyu Qi*, *Hootan Zhian*, *Murtaza Rangwala*, *Yifan Sun*, *Prabhjot Singh*, *Haoyu Bai*, *Abhishek Sawaika*, *and Avishka Sandeepa*.

I gratefully acknowledge the financial support provided by the University of Melbourne through the Melbourne Research Scholarship, as well as the support from the Australian Research Council (ARC) grants. I also wish to thank the past and present administrative staff of the School of Computing and Information Systems for their assistance throughout my candidature.

Finally, I dedicate this work to my family. Thank you for your enduring support and for believing in my potential throughout this journey. This achievement belongs to you as much as it does to me.

Zhiyu Wang
Melbourne, Australia
January 2026

Contents

List of Figures

List of Tables

Chapter 1
Introduction

This chapter establishes the foundation for the thesis by introducing the paradigm shift from centralized cloud computing to the heterogeneous IoT-Edge-Cloud computing continuum. It elucidates the intrinsic complexities of resource management within this continuum, which is characterized by extreme dynamism, scale, and data privacy constraints. Furthermore, it motivates the necessity of AI-driven approaches, particularly Deep Reinforcement Learning (DRL), as a superior decision-making mechanism. Subsequently, the chapter identifies seven critical research challenges limiting current solutions, defines the corresponding research objectives, and summarizes the main contributions of the thesis. These contributions range from theoretical taxonomies and optimization algorithms to comprehensive system frameworks. Finally, it outlines the organizational structure of the remainder of the thesis.

1.1 Background and Motivation

With the exponential growth of Internet of Things (IoT) devices globally [1, 2] and the continuous evolution of ubiquitous wireless connectivity, we are witnessing a profound transformation in computing paradigms. This paradigm shift not only fundamentally reshapes the patterns of data generation, transmission, and consumption, but also imposes unprecedented challenges on the underlying distributed resource management mechanisms. This section first elucidates the evolutionary logic of computing architectures shifting from a solitary cloud center toward a ubiquitous continuum. Subsequently, it dissects the intrinsic complexity of resource management within this context. Finally, it argues why data-driven Artificial Intelligence (AI) methods, particularly Deep Reinforcement Learning (DRL), constitute the inevitable scientific choice for addressing these complex challenges.

1.1.1 The Evolution of Computing Paradigms

Over the past few decades, Cloud Computing, with its powerful resource pooling capabilities and economies of scale, has served as the core pillar for processing large-scale data and complex applications [3]. However, as the digitization of the physical world accelerates, billions of heterogeneous terminal devicesranging from smart sensors and wearable devices to autonomous vehicleshave generated massive amounts of data distributed in a streaming fashion. A fundamental contradiction has emerged between the decentralized nature of this data generation and the centralized processing mode of cloud computing. Constrained by the physical limits of the speed of light, the high latency caused by long-distance data transmission cannot satisfy the demands of applications requiring immediate feedback. Simultaneously, the upload of massive raw data places increasing pressure on backbone network bandwidth and exacerbates data privacy risks.

To resolve these contradictions, Edge Computing and Fog Computing have emerged as novel computing paradigms [2, 4]. Following the principle of Data Gravity, these paradigms push computing, storage, and intelligent services from the network core down to the network edge, located in physical proximity to data sources. This architectural shift significantly reduces service response latency, alleviates network congestion, and enhances local data privacy [5].

However, Edge Computing is not a replacement for Cloud Computing; rather, it is an extension and complement in physical space. The current computing environment is evolving into a tightly coupled, multi-tier ecosystem, namely the **IoT-Edge-Cloud Computing Continuum** [6, 7], as illustrated in Figure 1.1. Within this continuum, resources are no longer isolated silos but form a complete spectrum spanning resource-constrained terminals (IoT devices), edge nodes with moderate computing power, and cloud data centers with virtually unlimited capacity. This continuum architecture aims to overcome the resource bottlenecks of any single tier through seamless cross-tier synergy, thereby supporting the stringent requirements of future intelligent applications regarding computing power, latency, and energy efficiency [8].

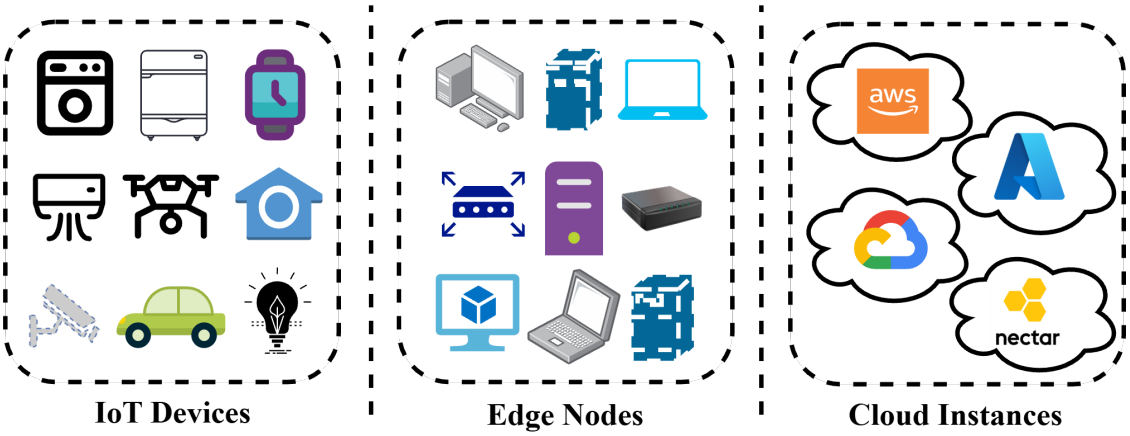


Figure 1.1: The Architecture of the IoT-Edge-Cloud Computing Continuum.

1.1.2 The Complexity Challenges of Resource Management

Although the IoT-Edge-Cloud continuum provides an ideal architectural blueprint for distributed computing, achieving efficient resource managementincluding task offloading, job scheduling, and resource allocationwithin such a massive and complex system faces severe challenges stemming from the intrinsic nature of the system [9, 10]. These challenges are primarily manifested in the following three dimensions:

First, **the complexity of task flows and the diversity of Quality of Service (QoS)**. Modern intelligent applications are no longer simple atomic tasks; they typically manifest as Directed Acyclic Graph (DAG) structures with strict data dependency relationships [11]. The temporal constraints between subtasks require scheduling decisions to possess a global perspective. Simultaneously, the QoS demands of different applications exhibit extreme divergence: mission-critical tasks demand deterministic low latency at the millisecond level, whereas compute-intensive tasks prioritize the long-term energy efficiency ratio of the system [12]. Balancing these conflicting objectives on heterogeneous resources represents a typical multi-objective optimization problem [13].

Second, **multi-dimensional system heterogeneity and scale effects**. The continuum is essentially a highly heterogeneous system [14, 15]. Compute nodes differ vastly in hardware architecture (CPU/GPU/TPU/NPU), processing capability, storage capacity, and energy supply modes. Communication links encompass hybrid networks ranging from wired fiber to various wireless protocols, exhibiting distinct bandwidth fluctua-

tion and packet loss characteristics. Furthermore, as the number of connected devices surges, the expansion of system scale causes the solution space for resource scheduling problems to explode exponentially, rendering the search for a global optimum computationally infeasible (NP-hard) within polynomial time [16].

Finally, **high environmental dynamism and stochasticity**. In real-world continuum environments, uncertainty is the norm rather than the exception [17]. Task arrival patterns often follow complex stochastic distributions. The network topology structure may undergo spatial-temporal evolution at any moment due to node mobility (e.g., mobile edge nodes, UAV relays). Additionally, wireless channel quality fluctuates dramatically due to environmental interference. This non-stationary characteristic implies that any offline resource management strategy based on static model assumptions will rapidly become ineffective [18].

1.1.3 The Necessity of AI-driven Approaches

Faced with the aforementioned systemic complexities, traditional resource management approaches have proven inadequate. Mathematical analytical methods based on Convex Optimization or Queueing Theory typically rely on idealized system models and strict assumptions, which are difficult to uphold in highly dynamic real-world environments [9]. Conversely, Rule-based or Meta-heuristic algorithms, while reducing dependency on models, often lack online learning capabilities. They struggle to make adaptive adjustments to environmental changes within millisecond-level time scales and are prone to getting trapped in local optima.

In this context, Deep Reinforcement Learning (DRL), as an emerging data-driven decision-making paradigm, has demonstrated unique advantages in solving such problems and has become a focal point for both academia and industry [8, 19]:

1. **Model-free Adaptation:** DRL agents do not require the pre-establishment of precise mathematical models of the system, such as state transition probability matrices. Instead, they learn optimal policies through continuous interaction with the environment (trial-and-error mechanisms) [20]. This enables them to naturally adapt to highly non-linear systems that are difficult to model [21].

2. **High-dimensional Perception and Decision-making:** Combining the feature extraction capabilities of Deep Neural Networks (DNNs), DRL can process high-dimensional, continuous, and complex input states, such as heterogeneous resource states and dynamic topologies [21]. Consequently, it can make more precise decisions within immense solution spaces.

3. **Online Continuous Evolution:** DRL possesses the potential for lifelong learning. When network environments, load patterns, or device topologies drift, trained DRL agents can quickly adapt to new environmental distributions through online fine-tuning, maintaining high system robustness and performance [8, 17].

In summary, leveraging AI-driven approaches, particularly DRL and its derivative technologies (such as multi-agent collaboration and federated learning mechanisms), to build an intelligent, adaptive, and scalable resource management system is not only key to resolving current technical bottlenecks but also the inevitable path to unleashing the full potential of the IoT-Edge-Cloud continuum [5, 8].

1.2 Research Challenges

While Deep Reinforcement Learning (DRL) offers a revolutionary paradigm for addressing highly dynamic and complex resource management problems, migrating it from idealized, isolated experimental environments to the realistic IoT-Edge-Cloud continuum is not a simple application transplant. Existing research efforts are predominantly confined to specific, isolated scenarios or predicated on simplified assumptions [8, 21]. However, the real-world continuum environment inherently exhibits multi-dimensional heterogeneity, Non-Independent and Identically Distributed (Non-IID) data, and extreme dynamics [14, 22]. These intrinsic characteristics create a substantial chasm between theoretical models and practical deployment.

Specifically, to achieve ubiquitous, intelligent, and robust resource management, this study identifies **seven core challenges** that must be overcome:

Challenge 1: Theoretical Fragmentation

Despite the proliferation of DRL applications in edge computing, the existing lit-

erature remains highly fragmented [8, 9]. Researchers often struggle to find clear guiding principles for architectural choices, such as Single-Agent vs. Multi-Agent, and Standard vs. Federated Training, when facing specific problems. The absence of a unified, orthogonal taxonomy leads to blindness in algorithm design and difficulties in identifying structural defects and gaps in existing solutions from a system perspective [21, 23].

Challenge 2: Inefficiency in Dynamic Multi-Objective Optimization

In fundamental edge-fog computing environments, task scheduling necessitates striking a delicate balance between system load balancing and application response time [9]. In particular, modern IoT applications typically manifest as Directed Acyclic Graphs (DAGs) with complex dependencies, which drastically increases the complexity of the state space and decision constraints [11]. When handling such structured task flows, generic DRL state representations and reward mechanisms often fail to capture critical path features effectively, resulting in agents struggling to efficiently converge to policies that satisfy weighted-cost optimality within stochastic dynamic environments [24, 25].

Challenge 3: Scalability Constraints in Large-Scale Systems

As the system scale expands to encompass hundreds or thousands of concurrent requests in IoT scenarios, the centralized DRL training paradigm faces severe scalability challenges [26, 27]. On the one hand, massive concurrent data streams lead to the curse of dimensionality, causing exploration costs to skyrocket; on the other hand, the spatiotemporal patterns of task arrivals often possess long-term dependencies, which traditional fully connected neural networks struggle to capture [28]. Ensuring scheduling decision quality while significantly reducing training and inference overhead in large-scale scenarios remains an urgent problem to be solved [8].

Challenge 4: Gap between Algorithm and Deployment

In practical deployment, constructing a unified software framework capable of flexibly supporting diverse DRL paradigmssuch as enabling native algorithm development versus third-party library integration, and switching between central-

ized and distributed architecturesremains an unresolved engineering challenge [29]. More critically, the DRL agent itself is a resource-consuming software component. In the resource-constrained and heterogeneous continuum, the physical placement of DRL components (e.g., the distribution of learners and actors) significantly impacts communication overhead and overall system latency, yet this system-level optimization problem is frequently overlooked in existing studies [30, 31].

Challenge 5: Model Incompatibility due to Device Heterogeneity

When scheduling scenarios extend to multi-domain environments, the heterogeneity of device computational power becomes a primary barrier to collaborative learning [15]. Cloud servers can execute deep neural networks, whereas resource-constrained IoT devices can only host lightweight models. Traditional federated learning methods typically mandate identical model architectures across all participants to facilitate parameter aggregation, which is evidently impractical in the heterogeneous continuum [14]. Breaking the constraint of identical model structures to enable cross-architecture knowledge sharing is a critical challenge for realizing ubiquitous intelligence [22].

Challenge 6: Data Silos and Adversarial Risks

In cross-organizational silo-cooperative scheduling scenarios, data across different management domains often exhibits significant Non-IID characteristics [22, 32]. Direct aggregation of model parameters can lead to severe performance degradation. Simultaneously, decentralized cooperative environments naturally lack a trust basis and are vulnerable to poisoning attacks from malicious nodes or noise interference from faulty nodes [33, 34]. Designing a robust collaboration mechanism that can adapt to Non-IID distributions and withstand adversarial attacks while preserving data privacy is extremely challenging [35].

Challenge 7: Coordination in Extreme Dynamic Topologies

In many mobile edge computing scenarios, the network topology structure changes momentarily with the rapid movement of nodes [36]. This highly dynamic characteristic renders spatial proximity relationships and communication links between nodes extremely unstable. Traditional DRL methods based on fixed vector inputs

struggle to capture these dynamic spatial topological relationships [37]. Furthermore, such networks are typically subject to strict communication bandwidth and energy constraints. How to achieve efficient multi-agent coordination and joint trajectory-task optimization in environments with highly time-varying topologies and constrained communication (such as aerial computing networks) represents a significant and urgent challenge in this field [38, 39].

1.3 Research Objectives

In view of the aforementioned seven core challenges, the primary objective of this thesis is to design and develop a comprehensive solution suite encompassing theoretical frameworks, optimization algorithms, system implementations, and cross-domain collaboration mechanisms, to achieve intelligent, efficient, and robust resource management in the IoT-Edge-Cloud continuum. Specifically, this research aims to achieve the following **seven core objectives**:

Objective 1: Constructing a Systematic Classification Taxonomy for DRL-based Resource Management

To address the theoretical fragmentation (Challenge 1), this thesis aims to conduct an in-depth analysis of existing literature and establish an orthogonal two-dimensional taxonomy [8, 9]. By systematically organizing existing works from the dimensions of control scope (single-agent vs. multi-agent) and training paradigm (standard vs. federated), this objective seeks to clarify the applicability boundaries and design principles of different architectures [21, 23].

Objective 2: Developing Weighted-Cost Optimization Algorithms for Complex Task Flows

To resolve the inefficiency in dynamic multi-objective optimization (Challenge 2), this thesis aims to design an adaptive scheduling algorithm based on improved policy gradients for IoT applications with DAG dependency structures [11]. By constructing composite reward functions and advantage estimation mechanisms [40, 41], the objective is to achieve Pareto optimization between response time and load

balancing in stochastic dynamic environments [24, 25].

Objective 3: Designing Scalable Distributed DRL Techniques with Long-term Dependency Modeling

To overcome scalability bottlenecks in large-scale systems (Challenge 3), this thesis aims to develop advanced DRL techniques combining Transformer architectures with distributed training paradigms for high-concurrency request scenarios. Utilizing attention mechanisms to capture the long-term spatiotemporal dependencies of task arrivals [42, 43], the objective is to significantly enhance sample efficiency and convergence speed in large-scale environments [26].

Objective 4: Implementing a Unified and Modular Intelligent Resource Management Framework

To bridge the gap between algorithm and deployment (Challenge 4), this thesis aims to design and implement a lightweight, container-supported software framework [29, 44]. The framework is required to possess high extensibility, enabling seamless integration of diverse DRL paradigms (native/library-based, centralized/distributed). Simultaneously, intelligent component placement algorithms will be proposed to optimize the physical deployment of DRL learners and actors, minimizing management overhead [30, 31].

Objective 5: Proposing Cross-Architecture Federated Learning Mechanisms Supporting Device Heterogeneity

To solve the model incompatibility issue in multi-domain environments (Challenge 5), this thesis aims to introduce environment-oriented knowledge distillation techniques [45, 46]. By constructing resource-aware adaptive architecture generation mechanisms, the objective is to break the structural barriers between heterogeneous devices (from IoT sensors to cloud servers), enabling cross-architecture knowledge transfer and collaborative training [14, 15].

Objective 6: Establishing Decentralized and Robust Cross-Domain Collaboration Protocols

To address data silos and adversarial risks (Challenge 6), this thesis aims to design

a fully decentralized federated learning architecture [32, 47]. Combining gradient fingerprint detection and gradient tracking technologies [48], the objective is to achieve effective knowledge aggregation under Non-IID data environments, while ensuring system robustness and convergence stability against malicious node attacks [33, 34].

Objective 7: Developing Graph Neural Network-based Methods for High-Dynamic Spatial Coordination

To tackle coordination difficulties in extreme dynamic topologies (Challenge 7), this thesis aims to utilize dual-layer dynamic Graph Attention Networks (GATs) to explicitly model the spatiotemporal evolution features of network topologies for scenarios like UAV-assisted computing [24, 49]. Furthermore, it seeks to develop a dual-actor collaborative decision-making mechanism based on hybrid action spaces to achieve joint optimization of trajectory and task offloading under strict communication and coverage constraints [38, 50].

1.4 Main Contributions

This thesis makes the following seven main contributions to the field of resource management in the IoT-Edge-Cloud continuum:

Contribution 1: A Two-Dimensional Taxonomy for DRL-based Resource Management (Chapter 2)

This thesis conducts a systematic review and analysis of existing literature, proposing a novel **orthogonal two-dimensional taxonomy**. This taxonomy reconstructs existing works from two dimensions: control scope (single-agent vs. multi-agent) and training paradigm (standard vs. federated). It resolves conceptual ambiguities in current surveys and clearly identifies critical research gaps in federated and decentralized architectures, establishing a theoretical foundation for subsequent chapters.

Contribution 2: A Weighted-Cost Optimization Algorithm DRLIS for Edge-Fog Environments (Chapter 3)

To address the trade-off between load balancing and response time in stochastic dynamic environments, this thesis proposes a Deep Reinforcement Learning-based IoT Scheduling algorithm, **DRLIS**. By customizing Proximal Policy Optimization (PPO) techniques and designing a composite reward function, DRLIS effectively optimizes the weighted cost of IoT applications with DAG structures. Practical implementation within the FogBus2 framework demonstrates that this algorithm significantly outperforms traditional meta-heuristic algorithms in convergence speed and decision quality.

Contribution 3: An Enhanced Distributed DRL Technique TF-DDRL (Chapter 4)

Addressing scalability constraints in large-scale heterogeneous environments, this thesis develops a Transformer-combined distributed DRL technique, **TF-DDRL**. Based on the IMPALA architecture, this technique integrates Gated Transformer-XL to capture long-term spatiotemporal dependencies of task flows and introduces Prioritized Experience Replay (PER) mechanisms to enhance sample efficiency. Experimental results prove that TF-DDRL significantly reduces exploration costs and improves scheduling performance when processing massive concurrent requests.

Contribution 4: A Unified Resource Management Software Platform ReinFog (Chapter 5)

To bridge the gap between algorithmic theory and system deployment, this thesis designs and implements a modular, containerized software platform, **ReinFog**. This platform supports the seamless integration of centralized and distributed DRL agents, including native implementations and third-party libraries. Furthermore, this thesis proposes a novel memetic algorithm, **MADCP**, specifically designed to optimize the physical deployment of DRL learner and worker components, thereby minimizing system communication overhead and accelerating training startup.

Contribution 5: A Knowledge Distillation-Empowered Federated Framework KD-AFRL (Chapter 6)

To overcome the heterogeneity challenge where resource-constrained devices cannot run complex models, this thesis proposes the **KD-AFRL** framework. This

framework introduces resource-aware adaptive architecture generation mechanisms and environment-oriented cross-architecture knowledge distillation methods. It enables heterogeneous devices, ranging from IoT sensors to cloud servers, to collaboratively learn optimal policies without sharing raw data, effectively bridging the performance gap between lightweight and large-scale models.

Contribution 6: A Decentralized Robust Federated Learning Framework DeFRiS (Chapter 7)

Targeting Non-IID workloads and adversarial risks in cross-organizational cooperative scheduling, this thesis develops the **DeFRiS** framework. Unlike traditional centralized federated approaches, DeFRiS adopts a fully decentralized peer-to-peer network architecture. It utilizes action-space-agnostic policies to enable knowledge transfer across heterogeneous infrastructures and effectively defends against anomalous node attacks through a dual-track robust aggregation protocol combining gradient fingerprints and gradient tracking, ensuring convergence in adversarial environments.

Contribution 7: A Graph Neural Network-Enhanced Multi-Agent Framework AirFed (Chapter 8)

Addressing extreme dynamic scenarios such as aerial edge computing, this thesis proposes the **AirFed** framework for multi-UAV coordination. This framework integrates dual-layer dynamic Graph Attention Networks (GATs) to explicitly model spatiotemporal dependencies and designs a dual-actor single-critic architecture to jointly optimize continuous trajectory control and discrete task offloading. Additionally, it introduces reputation-based federated mechanisms and adaptive quantization techniques, guaranteeing Quality of Service (QoS) under strictly constrained communication bandwidth.

1.5 Thesis Organization

This thesis is organized into nine chapters. A visual overview of the thesis structure and the interconnections between chapters is depicted in Figure 1.2. The outline of the

remaining chapters is as follows:

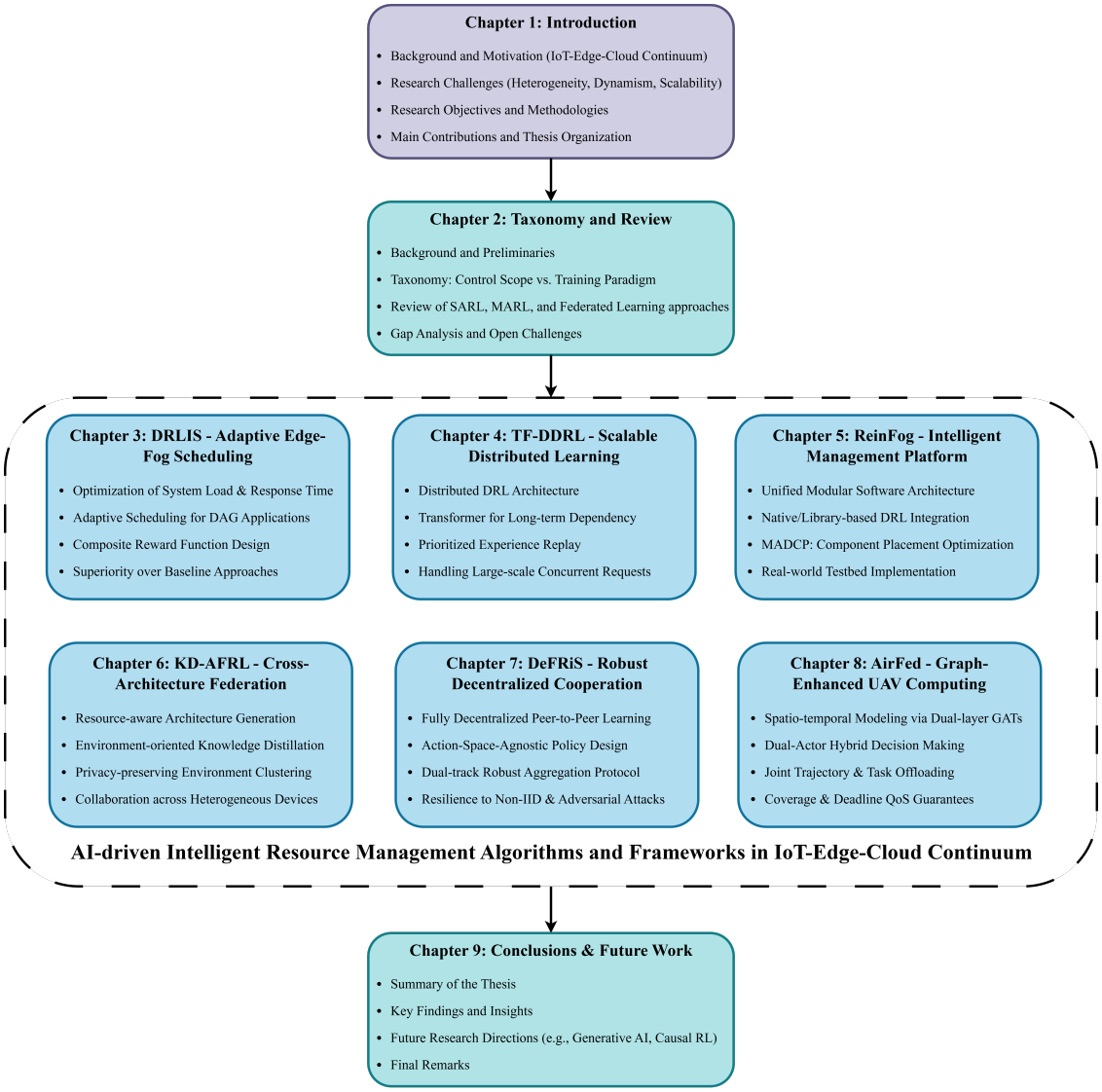


Figure 1.2: Visual overview of the thesis organization structure.

Chapter 2: Deep Reinforcement Learning for Resource Management in IoT-Edge-Cloud Continuum: A Taxonomy and Future Directions

This chapter provides a comprehensive literature review of DRL-based resource management techniques in the IoT-Edge-Cloud continuum. By analyzing existing works, it establishes an orthogonal classification system comprising two dimensions: control scope and training paradigm. Based on this taxonomy, the chapter

deeply analyzes the limitations of current research, particularly the gaps in federated and decentralized architectures, thereby motivating the research presented in this thesis. This chapter is derived from:

Zhiyu Wang, Mohammad Goudarzi, Mingming Gong, and Rajkumar Buyya, "Deep Reinforcement Learning for Resource Management in IoT-Edge-Cloud Continuum: A Taxonomy and Future Directions", *ACM Computing Surveys (CSUR)*, ACM, 2026 [Under Review].

Chapter 3: Deep Reinforcement Learning-based Scheduling for Optimizing System Load and Response Time in Edge and Fog Computing Environments

Focusing on fundamental single-node scheduling scenarios, this chapter proposes the DRLIS algorithm. Targeting IoT applications with DAG structures, it details how to balance response time and load balancing in stochastic dynamic environments through DRL and a composite reward function. The chapter verifies the superiority of DRLIS over meta-heuristic algorithms. This chapter is derived from:

Zhiyu Wang, Mohammad Goudarzi, Mingming Gong, and Rajkumar Buyya, "Deep Reinforcement Learning-based Scheduling for Optimizing System Load and Response Time in Edge and Fog Computing Environments", *Future Generation Computer Systems (FGCS)*, Vol. 152, pp. 55-69, Elsevier, 2024.

Chapter 4: TF-DDRL: A Transformer-enhanced Distributed DRL Technique for Scheduling IoT Applications

To address scalability issues in large-scale scenarios, this chapter introduces the TF-DDRL technique. It demonstrates how to utilize Gated Transformer-XL to capture long-term temporal dependencies of task flows and enhance sampling efficiency through the IMPALA distributed architecture, thereby significantly reducing exploration costs when processing massive concurrent requests. This chapter is derived from:

Zhiyu Wang, Mohammad Goudarzi, and Rajkumar Buyya, "TF-DDRL: A Transformer-Enhanced Distributed DRL Technique for Scheduling IoT Applications in Edge and Cloud Computing Environments", *IEEE Transactions on Services Computing (TSC)*, Vol. 18, No. 2, pp. 1039-1053, IEEE, 2025.

Chapter 5: ReinFog: A Deep Reinforcement Learning Empowered Framework for Resource Management

Shifting from algorithms to system implementation, this chapter details the design and development of the ReinFog framework. It elucidates how the framework supports the seamless integration of multiple DRL paradigms through a modular design and proposes a memetic algorithm-based component placement strategy (MADCP) to optimize the deployment efficiency of the DRL system itself. This chapter is derived from:

Zhiyu Wang, Mohammad Goudarzi, and Rajkumar Buyya, "ReinFog: A Deep Reinforcement Learning Empowered Framework for Resource Management in Edge and Cloud Computing Environments", *Journal of Network and Computer Applications (JNCA)*, Vol. 242, Article 104250, Elsevier, 2025.

Chapter 6: A Knowledge Distillation-empowered Adaptive Federated Reinforcement Learning Framework

Addressing the challenge of device heterogeneity, this chapter proposes the KD-AFRL framework. It details a resource-aware adaptive architecture generation mechanism and an environment-oriented cross-architecture knowledge distillation method, demonstrating how federated learning enables lightweight edge devices to acquire decision-making capabilities comparable to cloud-based large models. This chapter is derived from:

Zhiyu Wang, Mohammad Goudarzi, Mingming Gong, and Rajkumar Buyya, "A Knowledge Distillation-empowered Adaptive Federated Reinforcement Learning Framework for Multi-Domain IoT Applications Scheduling", *IEEE Transactions on Mobile Computing (TMC)*, IEEE, 2026 [In Press].

Chapter 7: DeFRiS: Silo-Cooperative IoT Applications Scheduling via Decentralized Federated Reinforcement Learning

Targeting data silos and security risks in cross-organizational collaboration, this chapter introduces the DeFRiS framework. It explains how to resolve compatibility issues of heterogeneous infrastructures through action-space-agnostic policies and utilizes a dual-track aggregation protocol (gradient fingerprint and tracking)

to achieve robust decentralized collaboration under Non-IID data and adversarial environments. This chapter is derived from:

Zhiyu Wang, Mohammad Goudarzi, Mingming Gong, and Rajkumar Buyya, "DeFRiS: Silo-Cooperative IoT Applications Scheduling via Decentralized Federated Reinforcement Learning", *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, IEEE, 2026 [Under Review].

Chapter 8: AirFed: A Federated Graph-Enhanced Multi-Agent Reinforcement Learning Framework

This chapter explores resource management under extreme dynamic topologies by proposing the AirFed framework. It demonstrates how to model spatial dependencies using dual-layer dynamic Graph Attention Networks (GATs) and jointly optimize UAV trajectory control and task offloading through a dual-actor single-critic architecture to guarantee Quality of Service (QoS) under strict communication constraints. This chapter is derived from:

Zhiyu Wang, Suman Raj, and Rajkumar Buyya, "AirFed: Federated Graph-Enhanced Multi-Agent Reinforcement Learning for Multi-UAV Cooperative Mobile Edge Computing", *IEEE Transactions on Mobile Computing (TMC)*, IEEE, 2026 [Under Review].

Chapter 9: Conclusions and Future Directions

This chapter summarizes the main research findings and contributions of the entire thesis. Based on the limitations of the current work, it identifies potential directions for future research in IoT-Edge-Cloud continuum resource management.

Chapter 2

Deep Reinforcement Learning for Resource Management in Computing Continuum: A Taxonomy and Future Directions

The Internet of Things (IoT)-Edge-Cloud computing continuum demands intelligent resource management to satisfy stringent latency, energy, and quality-of-service requirements. Deep Reinforcement Learning (DRL) has emerged as a promising paradigm for adaptive resource management, capable of learning effective policies without requiring accurate analytical models. However, existing surveys lack a unified framework that systematically organizes approaches across fundamental design dimensions. This chapter proposes a novel two-dimensional taxonomy that orthogonally separates control scope (single-agent versus multi-agent) from training paradigm (standard versus federated), resolving ambiguities in prior work. We provide comprehensive literature review, comparative analysis, and practical design guidelines mapping deployment requirements to algorithmic choices. We further survey advanced enhancement techniques and identify critical open challenges with future research directions toward intelligent, scalable, and privacy-preserving resource management.

2.1 Introduction

The convergence of Internet of Things (IoT) and edge computing has created a paradigm shift in distributed computing architectures. With IoT deployments reaching unprece-

This chapter is derived from:

- **Zhiyu Wang**, Mohammad Goudarzi, Mingming Gong, and Rajkumar Buyya, "Deep Reinforcement Learning for Resource Management in IoT-Edge-Cloud Continuum: A Taxonomy and Future Directions", *ACM Computing Surveys (CSUR)*, 2026 [Under Review].

dented scale worldwide and generating massive data volumes, traditional cloud-centric architectures face fundamental limitations in serving latency-sensitive applications [51]. Emerging use cases such as autonomous vehicles, augmented reality, and industrial control systems require near-instantaneous response times that cannot tolerate the round-trip delays inherent to distant cloud datacenters [52].

The IoT-Edge-Cloud continuum has emerged as the architectural response, forming a three-tier distributed computing hierarchy that bridges resource-constrained IoT devices with resource-abundant cloud datacenters through intermediate edge infrastructure deployed at network edges [53]. By distributing computation across this hierarchy, the continuum enables latency-critical processing at the edge while leveraging cloud resources for compute-intensive analytics, creating a distributed computing fabric spanning billions of heterogeneous devices across global geographical scales [51].

Efficient resource management across this continuum is both critical and challenging. System operators must dynamically allocate computational resources, schedule heterogeneous workloads, and make intelligent task placement decisions under stringent constraints [54]. The challenges are multifaceted: unprecedented scale, extreme dynamics, pervasive heterogeneity spanning device capabilities and network technologies, and strict operational constraints including latency bounds and energy budgets [52]. Moreover, privacy regulations and data sovereignty requirements increasingly mandate that sensitive data remains local to edge nodes, prohibiting centralized data collection for training [55]. Inefficient resource allocation directly translates to quality-of-service violations, excessive energy consumption, and substantial operational costs for large-scale deployments [54, 56].

Deep Reinforcement Learning (DRL) has emerged as a promising paradigm for adaptive resource management in this complex environment. Unlike traditional rule-based heuristics relying on manually designed policies or model-based optimization requiring accurate analytical models, DRL agents learn near-optimal policies through environmental interaction without prior system models [57]. This model-free adaptive learning capability makes DRL particularly suited to the IoT-Edge-Cloud continuum, where workload patterns are unpredictable, system dynamics are non-stationary, and accurate models are difficult to obtain [58]. However, standard DRL training assumes un-

restricted data access, which conflicts with privacy regulations and data sovereignty requirements in IoT/edge environments [59]. Federated learning addresses this challenge by enabling collaborative model training across distributed nodes/regions while keeping data local, making it an essential training paradigm for privacy-preserving resource management [33]. Research in this area has grown rapidly, motivating the need for a comprehensive survey that systematically organizes the expanding literature.

2.1.1 Related Surveys and Research Gap

Resource management in the IoT-Edge-Cloud continuum has generated numerous surveys, but these works exhibit significant limitations in scope definition, methodological organization, and analytical depth. Table 2.1 systematically compares 16 representative surveys published between 2023 and 2025, evaluating them across two dimensions: **coverage** (breadth of methods and architectures discussed) and **analytical components** (depth of analysis provided), revealing substantial gaps in existing literature.

Single-Agent Dominance with Weak Multi-Agent Coverage. Among the 16 surveys, 12 fully cover serial SARL methods (DQN, PPO, SAC), but only 4 partially address parallel SARL (A3C, IMPALA). Multi-agent coverage is extremely weak: only 2 surveys mention independent MARL (IDQN, IPPO), and only Yang et al. [8] covers CTDE methods (QMIX, MADDPG, MAPPO). This gap is critical given that the IoT-Edge-Cloud continuum is inherently a distributed system where multi-agent architectures naturally map to autonomous edge nodes.

Nearly Complete Absence of Federated Training Paradigm. Only 3 surveys provide comprehensive coverage of centralized federated learning, with 6 offering partial treatment. Most critically, no survey covers decentralized federated architectures, despite their crucial trade-offs in convergence speed, scalability, and fault tolerance for large-scale deployments. Furthermore, existing surveys treat federated learning as an isolated technique rather than a training paradigm orthogonally combinable with different control architectures, leaving federated single-agent and federated multi-agent design spaces unexplored.

Insufficient Comparative Analysis with Limited Design Guidance. While all surveys include comparisons, most provide only qualitative descriptions lacking quantitative performance evaluation or systematic deployment trade-offs. Only 2 surveys provide comprehensive design guidelines, with 5 offering partial guidance. The fundamental gap is that no survey provides a systematic comparative framework across the two-dimensional space of control architectures and training paradigms, making it difficult for system designers to map deployment characteristics to appropriate DRL configurations.

Inadequate Coverage of Enhancement Techniques and Emerging Paradigms. Only 5 surveys cover advanced techniques that augment base DRL methods. These techniques are essential for practical deployment, addressing challenges such as resource-constrained devices, rapid adaptation to new environments, and safety constraint satisfaction. However, existing surveys lack systematic analysis of their applicability to IoT/edge/cloud scenarios or practical deployment guidelines. The potential of emerging paradigms to advance resource management capabilities also remains largely unexplored.

As shown in Table 2.1, this chapter is the first to provide comprehensive coverage across both dimensions of control scope (serial/parallel SARL, independent/CTDE MARL) and training paradigms (standard, centralized/decentralized federated), introduces a principled orthogonal taxonomy, provides systematic cross-dimensional comparative analysis with practical deployment guidelines, comprehensively reviews enhancement techniques, and provides in-depth analysis of emerging paradigms. By systematically addressing the fragmentation in existing literature, this chapter establishes a comprehensive foundation for DRL-based resource management in the IoT-Edge-Cloud continuum.

2.1.2 Chapter Contributions and Scope

This chapter makes the following contributions:

(1) Two-Dimensional Taxonomy. We propose a two-dimensional taxonomy that orthogonally separates control scope (single-agent vs. multi-agent) from training paradigm

Table 2.1: Comparison of existing surveys on resource management in IoT/edge/cloud computing environments (Ordered by Year).

Survey	Year	Coverage						Analytical Components			
		Standard				Federated		Comparative	Guidelines	Advanced Techniques	Directions
		SARL		MARL		Centr-alized	Decentr-alized				
		Serial	Parallel	Indep.	CTDE						
Zhang et al. [10]	2023	✓	×	×	×	✓	×	✓	○	×	✓
Chiang et al. [60]	2023	✓	○	×	×	○	×	✓	○	×	✓
Wang et al. [61]	2023	✓	×	×	×	×	×	✓	○	×	✓
Hortelano et al. [23]	2023	✓	○	×	×	○	×	✓	×	×	✓
Zabihi et al. [21]	2024	✓	○	×	×	○	×	✓	×	○	✓
Walia et al. [5]	2024	✓	×	×	×	✓	×	✓	✓	○	✓
Tang et al. [28]	2024	✓	×	○	×	○	×	✓	○	✓	✓
Dong et al. [62]	2024	✓	×	×	×	×	×	✓	×	×	✓
Asghari et al. [63]	2024	○	×	×	×	×	×	✓	×	×	✓
Maia et al. [64]	2024	○	×	×	×	×	×	✓	×	×	✓
Zolghadri et al. [65]	2024	○	×	×	×	×	×	✓	×	×	✓
Taleb et al. [52]	2025	✓	×	×	×	×	×	✓	○	×	✓
Rasouli et al. [12]	2025	○	×	×	×	○	×	✓	×	×	✓
Vetriveeran et al. [66]	2025	✓	×	×	×	×	×	✓	×	×	✓
Yang et al. [8]	2025	✓	○	✓	✓	○	×	✓	✓	○	✓
Wu et al. [67]	2025	✓	×	×	×	✓	×	✓	×	✓	✓
This chapter	**2026**	**✓**	**✓**	**✓**	**✓**	**✓**	**✓**	**✓**	**✓**	**✓**	**✓**

Note: ✓: Fully covered; ○: Partially covered; ×: Not covered. **SARL**: Single-Agent RL; **MARL**: Multi-Agent RL; **CTDE**: Centralized Training with Decentralized Execution.

(standard vs. federated), resolving ambiguities in prior surveys that conflate these independent design dimensions.

(2) Comprehensive Literature Review. We systematically review over 100 papers applying DRL to resource management in IoT-Edge-Cloud continuum (2020-early 2026), organizing them through our two-dimensional taxonomy.

(3) Comparative Analysis and Design Guidelines. We systematically analyze and compare methods within each taxonomy category and provide practical design guidelines mapping deployment requirements to appropriate algorithmic choices.

(4) Advanced Techniques Survey. We survey orthogonal enhancement techniques including specialized network architectures, training optimizations, model compression, and safety mechanisms that apply across taxonomy categories.

(5) Open Challenges and Future Directions. We identify critical research challenges where substantial gaps remain and outline promising directions for future research.

Scope. This chapter covers DRL methods for resource management in the IoT-Edge-Cloud continuum (2020-early 2026). We exclude non-DRL methods and pure network-

ing problems without explicit resource management objectives.

2.1.3 Organization

The rest of this chapter is organized as follows. Section 2.2 provides background on the IoT-Edge-Cloud continuum, DRL fundamentals, and problem formulation. Section 2.3 presents our two-dimensional taxonomy. Sections 2.4 and 2.5 systematically review DRL approaches under standard and federated training paradigms, each organized by control scope (SARL then MARL). Section 2.6 surveys advanced techniques. Section 2.7 discusses open challenges and future directions. Section 2.8 provides the conclusion.

2.2 Background and Preliminaries

This section establishes the foundational concepts for DRL-based resource management in the IoT-Edge-Cloud continuum, covering the computing architecture (Section 2.2.1), reinforcement learning framework (Section 2.2.2), and problem formulation (Section 2.2.3).

2.2.1 IoT-Edge-Cloud Continuum

The IoT-Edge-Cloud continuum is a hierarchical computing paradigm that bridges resource-constrained IoT devices with cloud datacenters through intermediate edge infrastructure [52, 53]. Unlike cloud-centric architectures, this continuum distributes workloads across three tiers based on latency requirements, resource availability, and data locality [54].

As illustrated in Figure 2.1, the *cloud layer* provides virtually unlimited computational resources for intensive analytics and model training [52]. The *edge layer*, comprising base stations, cloudlets, and micro-datacenters, offers moderate resources with low-latency access to end devices [52]. The *IoT layer* encompasses heterogeneous devices with constrained computational capacity, memory, energy budgets, and intermittent connectivity [2, 68]. Many IoT applications exhibit DAG-structured task dependencies requiring precedence-aware scheduling [59, 69].

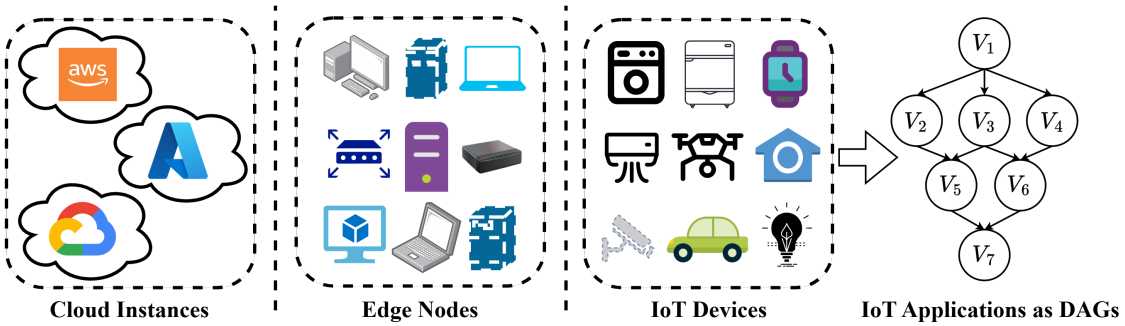


Figure 2.1: The IoT-Edge-Cloud continuum architecture.

This architecture presents three fundamental challenges that motivate learning-based approaches:

Dynamic workloads. IoT applications generate highly variable traffic patterns with temporal dynamics and spatial heterogeneity [54]. User mobility further exacerbates unpredictability [70], while DAG-structured dependencies impose time-varying execution constraints [59]. Static policies cannot adapt to such variability.

Pervasive heterogeneity. The continuum spans devices with vastly different capabilities, network technologies with variable bandwidth and latency, and applications with fundamentally different QoS requirements [54, 59, 71]. Manual policy design for each scenario is infeasible [26].

Model uncertainty. Stochastic task arrivals, network fluctuations, and device failures render accurate system modeling intractable [52, 59]. Optimization approaches assuming known models suffer severe performance degradation under such uncertainty.

These challenges, namely dynamism, heterogeneity, and uncertainty, necessitate adaptive approaches capable of learning effective policies through environmental interaction, motivating the DRL methods formalized next.

2.2.2 Deep Reinforcement Learning Fundamentals

DRL provides a mathematical framework for sequential decision-making under uncertainty, where an agent learns optimal behavior through environmental interaction [20]. Unlike supervised learning requiring labeled examples, DRL agents learn from evaluative feedback in the form of sparse reward signals, making it suited to resource man-

agement scenarios where optimal policies are unknown *a priori* but system performance can be measured.

The problem is formalized as a **Markov Decision Process (MDP)**, defined by the tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$: state space $\mathcal{S}$ describing system configurations, action space $\mathcal{A}$ representing available decisions, transition function $P(s'|s, a)$ specifying state dynamics, reward function $R(s, a)$ providing scalar feedback, and discount factor $\gamma \in [0, 1]$ balancing immediate and future rewards.

A **policy** $\pi : \mathcal{S} \to \Delta(\mathcal{A})$ maps states to action distributions. The agent's objective is to discover a policy maximizing expected cumulative discounted reward:

$$J(\pi) = \mathbb{E}_{\tau \sim \pi} \left[\sum_{t=0}^{\infty} \gamma^t r_t \right], \tag{2.1}$$

where τ denotes a trajectory sampled by executing policy π. The **state value function** $V^\pi(s)$ and **state-action value function** $Q^\pi(s, a)$ represent expected cumulative rewards from a given state or state-action pair, respectively. The optimal policy satisfies $\pi^*(s) = \arg\max_a Q^*(s, a)$.

Policy optimization follows three paradigmatic approaches: *value-based methods* learn Q-functions and derive policies through value maximization; *policy gradient methods* directly optimize parameterized policies; *actor-critic methods* maintain both policy and value representations.

In multi-agent settings, the framework extends to **Decentralized Partially Observable MDPs (Dec-POMDPs)**, where each agent i receives local observations $o_i \in \mathcal{O}_i$, executes actions $a_i \in \mathcal{A}_i$, and receives individual rewards r_i. The joint action $\mathbf{a} = (a_1, \ldots, a_n)$ determines state transitions, creating strategic interactions addressed through independent learning or coordinated training mechanisms.

2.2.3 Resource Management Problem Formulation

Resource management in the IoT-Edge-Cloud continuum can be naturally formulated as MDPs where the system learns allocation, scheduling, and offloading decisions to optimize performance metrics.

State space design. The state s_t typically aggregates three categories: *system state* (resource utilization, energy levels, queue lengths), *workload state* (task characteristics, application priorities, mobility patterns), and *network state* (bandwidth, latency, channel quality, topology).

Action space design. The action a_t specifies resource management decisions that may be discrete (offloading decisions, server selection, cache admission), continuous (resource allocation ratios, power levels, bandwidth shares), or hybrid combinations thereof.

Reward function design. The reward r_t encodes optimization objectives as scalar feedback. Common formulations include single-objective rewards (e.g., $r = -\alpha \cdot \text{latency}$), multi-objective weighted combinations (e.g., $r = -\alpha \cdot \text{latency} - \beta \cdot \text{energy} + \gamma \cdot \text{throughput}$), and constraint penalties for SLA violations or resource overloads.

Several challenges complicate MDP formulations in this domain:

Partial observability. Edge nodes lack complete visibility into global system state, necessitating decentralized multi-agent formulations or recurrent architectures that infer hidden state from observation histories.

Hybrid action spaces. Many problems require simultaneous discrete and continuous decisions. Value-based methods handle discrete actions but struggle with continuous spaces, while policy gradient methods accommodate both at increased training complexity.

Multi-objective optimization. Conflicting objectives, such as latency versus energy consumption, require careful reward engineering or constrained reinforcement learning formulations.

Privacy and data sovereignty. Regulations such as GDPR mandate that sensitive data remains within specific jurisdictions, prohibiting centralized data collection and requiring federated learning paradigms.

These formulation challenges motivate the diverse DRL architectures reviewed in subsequent sections, organized through our two-dimensional taxonomy.

2.3 Taxonomy and Classification Methodology

We propose a two-dimensional taxonomy that orthogonally separates control scope from training paradigm for DRL-based resource management in the IoT-Edge-Cloud continuum. This framework enables systematic comparison of methods based on:

- **Dimension I: Control Scope**. This dimension distinguishes approaches based on the number of autonomous decision-making entities:
  - *Single-Agent Architecture (SARL)*: A unified global policy is learned and deployed to control resource management across the entire IoT-Edge-Cloud system. While training may involve multiple distributed workers (e.g., A3C, IMPALA) to accelerate convergence, all system components ultimately execute the same shared policy.
  - *Multi-Agent Architecture (MARL)*: Multiple autonomous agents are deployed across the system (e.g., one per edge computing node or region), each learning and executing its own policy. Agents may operate independently (e.g., IPPO) or coordinate through cooperation (e.g., MADDPG, QMIX).
- **Dimension II: Training Paradigm**. This dimension categorizes approaches based on whether federated learning is employed:
  - *Standard Training*: No privacy constraints on data sharing. Training data, intermediate results (e.g., gradients, experiences), or model parameters can be freely exchanged between computing nodes/regions.
  - *Federated Training*: Training under strict data locality constraints for privacy preservation. Raw training data remains local at each computing node/region. Nodes exchange only model information (e.g., parameters in FedAvg, gradients in FedSGD) through federated aggregation mechanisms, preserving data sovereignty. Implementation can be through centralized architectures (with a central server) or decentralized architectures (peer-to-peer coordination).

Table 2.2 summarizes the resulting taxonomy framework. The two dimensions are orthogonal: any control scope (SARL or MARL) can be combined with any training paradigm (Standard or Federated), yielding distinct architectural patterns with different deployment requirements and performance characteristics. Detailed taxonomies for each paradigm are presented in Figures 2.2 and 2.3.

Table 2.2: Two-dimensional taxonomy framework for DRL-based resource management in IoT-Edge-Cloud continuum.

Control Scope / **Training Paradigm**	**Single-Agent Architecture (SARL)**	**Multi-Agent Architecture (MARL)**
Standard Training (Fig. 2.2)	2.4.1	2.4.2
Federated Training (Fig. 2.3)	2.5.1, 2.5.2	2.5.1, 2.5.2

2.4 Standard Training Paradigm

The standard training paradigm assumes no privacy constraints on data sharing, enabling free exchange of training data, gradients, experiences, or model parameters between computing nodes. Within this paradigm, we categorize DRL approaches by control scope: *single-agent architectures* employ a unified global policy offering coherent optimization at the cost of potential scalability bottlenecks, while *multi-agent architectures* deploy autonomous agents with individual policies, enabling distributed decision-making that aligns with edge computing infrastructures.

Figure 2.2 illustrates the taxonomy under this paradigm. Section 2.4.1 explores single-agent architectures with serial and parallel training approaches. Section 2.4.2 investigates multi-agent architectures, analyzing independent learning and centralized training with decentralized execution.

2.4.1 Single-Agent Architectures (SARL)

SARL employs a unified global policy to coordinate resource management across the entire IoT-Edge-Cloud continuum. This centralized decision-making paradigm is par-

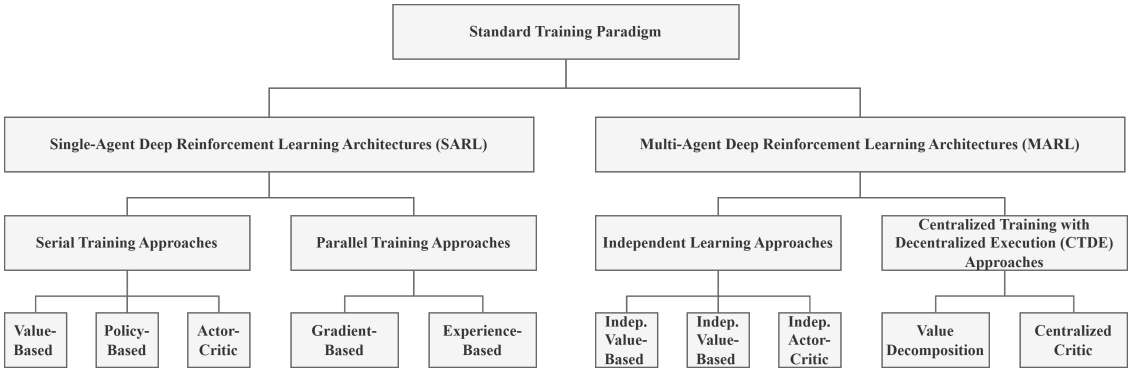


Figure 2.2: Taxonomy of DRL methods under standard training paradigm.

ticularly prevalent in scenarios where: (i) a central controller has visibility into system-wide state, (ii) resource pools can be managed holistically, and (iii) global optimization objectives (e.g., minimizing end-to-end latency, maximizing throughput) outweigh the benefits of distributed autonomy.

We organize SARL methods by their training mechanisms. *Serial Training Approaches* execute learning on a single computational thread, suitable for small to medium-scale deployments. *Parallel Training Approaches* leverage distributed workers to accelerate learning, essential for large-scale IoT environments generating high-volume data streams. This dichotomy reflects a fundamental trade-off between implementation simplicity and training scalability.

Serial Training Approaches

Serial training approaches sequentially collect experiences and update policies on a single computational thread. These approaches remain widely adopted in resource management due to mature implementations, theoretical guarantees, and effectiveness in moderately-sized problem spaces. We organize serial methods by their policy representation: *Value-Based Methods* learn state-action value functions to derive policies through value maximization, *Policy-Based Methods* directly parameterize and optimize policies using gradient ascent, and *Actor-Critic Methods* maintain both explicit policy and value function representations to combine their respective advantages.

Value-Based Methods. Value-based methods learn state-action value functions $Q(s, a)$ that estimate the expected cumulative reward of executing action a in state s, with policies derived through value maximization (e.g., $\pi(s) = \arg\max_a Q(s, a)$). This indirect policy representation is particularly suited to discrete action spaces common in resource allocation decisions.

Deep Q-Network (DQN) [72] pioneered the integration of deep neural networks with Q-learning by introducing experience replay (storing transitions (s, a, r, s') in a replay buffer $\mathcal{D}$ and sampling mini-batches to break temporal correlations) and a separate target network to stabilize training. Subsequent innovations addressed DQN's limitations: Double DQN [73] mitigates overestimation bias by decoupling action selection and evaluation; Dueling DQN [74] decomposes $Q(s, a)$ into state value $V(s)$ and advantage $A(s, a)$ streams, improving learning efficiency when many actions yield similar values; Rainbow [75] synthesizes six orthogonal improvements (Double DQN for overestimation mitigation, dueling architecture for value-advantage decomposition, prioritized experience replay, multi-step learning, categorical value distribution, and noisy networks for exploration), achieving superior sample efficiency.

In IoT-Edge-Cloud resource management, DQN variants have been extensively deployed for discrete decision problems. Xiong et al. [76] applied DQN to minimize the long-term weighted sum of task completion time and resource utilization in IoT edge computing. Liao et al. [77] employed Double DQN to jointly optimize offloading decisions and resource allocation, balancing delay and energy consumption in mobile edge computing. Birhanu Tadele et al. [78] leveraged Dueling DQN to minimize end-to-end Age of Information in vehicular fog systems, improving information freshness for real-time IoT services. Zhang et al. [79] deployed Rainbow DQN to reduce waiting delay and system congestion in Industrial IoT data scheduling.

Despite their effectiveness for discrete decisions, value-based methods face inherent limitations. Action spaces grow exponentially with decision dimensionality (e.g., simultaneously allocating CPU, memory, and bandwidth), causing the curse of dimensionality [20, 80]. Furthermore, these methods cannot directly handle continuous action spaces (e.g., precise power levels, bandwidth percentages) without discretization, which introduces approximation errors and scalability issues [80]. These constraints motivate

policy-based approaches that directly parameterize policies over continuous spaces.

Policy-Based Methods. Policy-based methods directly parameterize and optimize policies $\pi_\theta(a|s)$ without explicitly learning value functions, enabling natural handling of continuous and high-dimensional action spaces. Modern policy gradient methods incorporate value function baselines to reduce variance while maintaining the core policy optimization objective.

Trust Region Policy Optimization (TRPO) [41] ensures monotonic policy improvement by constraining the KL divergence between successive policies, providing theoretical guarantees against catastrophic performance collapse. However, TRPO's second-order optimization requirements impose substantial computational costs, limiting its applicability to large-scale systems. Proximal Policy Optimization (PPO) [40] simplifies TRPO's approach through a clipped surrogate objective that limits the ratio between new and old policies, preventing destructive updates. PPO's first-order implementation achieves comparable performance to TRPO with significantly lower computational overhead, explaining its widespread adoption in practice.

In resource management scenarios, these methods have proven particularly effective for problems with continuous action spaces and complex constraints. Priyadarshni et al. [81] employed TRPO within a meta reinforcement learning framework for task offloading in multi-access edge computing, reducing network traffic and latency through adaptive policy learning across diverse task topologies. Wang et al. [26] applied PPO to optimize scheduling decisions, jointly minimizing system load and response time in edge and fog computing environments, while Wang et al. [82] proposed an approach based on PPO for computation offloading, achieving fast adaptation to dynamic environments with minimal retraining requirements.

The stability of PPO's policy updates is particularly valuable in production deployments where catastrophic performance degradation must be avoided. However, both PPO and TRPO remain fundamentally on-policy, requiring fresh samples for each update and continuous environment interaction, which limits sample efficiency compared to off-policy methods that can reuse historical data. This motivates actor-critic architectures that combine policy optimization with off-policy learning.

Actor-Critic Methods. Actor-critic methods maintain explicit parameterizations of both policy (actor) $\pi_\theta(a|s)$ and value function (critic) $Q_\phi(s,a)$ or $V_\psi(s)$. The critic evaluates actions to guide policy improvement, while the actor focuses on policy representation. Off-policy actor-critic methods further enable experience replay, dramatically improving sample efficiency. This is critical for real-world deployments where environment interactions are expensive.

Deep Deterministic Policy Gradient (DDPG) [80] adapts DQN's experience replay and target networks to continuous control by learning a deterministic policy $\mu_\theta(s)$ guided by a critic $Q_\phi(s,a)$. However, DDPG suffers from overestimation bias and sensitivity to hyperparameters. Twin Delayed DDPG (TD3) [83] addresses these issues through three key innovations: twin critics (taking the minimum to reduce overestimation), delayed policy updates (updating the actor less frequently than critics), and target policy smoothing (adding noise to target actions for regularization). Soft Actor-Critic (SAC) [84] further improves sample efficiency and stability by incorporating maximum entropy reinforcement learning, which encourages exploration by maximizing both expected return and policy entropy. SAC's automatic temperature tuning makes it particularly robust across diverse tasks without extensive hyperparameter search.

These off-policy actor-critic methods have become the choice for continuous control problems in resource management. Ale et al. [85] employed DDPG for task partitioning and offloading with constrained hybrid action spaces in mobile edge computing, effectively handling both continuous and discrete decision variables. Li et al. [86] applied an improved TD3 algorithm for resource allocation optimization in ISAC-aided vehicular edge computing, minimizing total latency for delay-sensitive tasks through prioritized experience sampling. Tang et al. [87] utilized SAC within a collective deep reinforcement learning framework for intelligence sharing across distributed edge nodes, achieving superior learning efficiency through local and collaborative learning mechanisms.

Comparative studies reveal nuanced trade-offs among these methods. Empirical evaluations demonstrate that TD3 typically achieves higher asymptotic performance through its delayed policy updates and target policy smoothing mechanisms [83], while SAC exhibits superior sample efficiency and training stability via maximum entropy reinforcement learning at the cost of increased computational overhead [84]. Both im-

prove upon DDPG in various continuous control benchmarks [83, 84], with the choice between them depending on the relative importance of final performance versus sample efficiency in the specific application.

Parallel Training Approaches

As IoT deployments scale to millions of devices generating continuous data streams, single-threaded serial training struggles to meet learning efficiency demands. Parallel training architectures address this challenge by distributing the learning process across multiple workers, enabling both faster convergence through increased data throughput and improved exploration through diverse parallel trajectories. We distinguish two paradigms based on what workers share: *Gradient-Based Parallelism* where workers compute and communicate gradients, and *Experience-Based Parallelism* where workers collect experiences for centralized learning.

Gradient-Based Parallelism. Gradient-based parallel methods distribute both environment interaction and gradient computation across multiple workers that share a global parameter server. Each worker independently collects experiences, computes gradients, and contributes to global model updates, effectively parallelizing the entire learning pipeline.

Asynchronous Advantage Actor-Critic (A3C) [88] pioneered this paradigm by executing multiple workers asynchronously. Each worker updates shared parameters immediately upon computing gradients without waiting for others. This asynchrony accelerates learning by maximizing hardware utilization and provides implicit exploration diversity as workers explore different parts of the state space simultaneously. The advantage function $A(s,a) = Q(s,a) - V(s)$ reduces gradient variance, stabilizing learning despite asynchronous updates. However, A3C's asynchrony can cause workers to compute gradients based on stale parameters, potentially degrading sample efficiency. Advantage Actor-Critic (A2C), a synchronous variant proposed in the same paper, addresses this by batching gradients from all workers before updating, ensuring stable gradient aggregation and better training consistency while trading off some parallelism for improved stability.

In edge computing scenarios, gradient-based parallelism naturally maps to distributed edge infrastructure. Zou et al. [89] proposed A3C-DO, a regional resource scheduling framework for edge scenarios that addresses device heterogeneity and task complexity through dynamic offloading decisions, achieving improved service quality and resource utilization compared to conventional static strategies. Zhang et al. [27] extended A3C to large-scale Industrial IoTs environments, employing Markov game-based modeling and heuristic policy annealing to enable distributed asynchronous task scheduling across edge clouds, effectively balancing workloads and reducing communication latency. Lu et al. [90] developed A2C-DRL for dynamic scheduling in stochastic edge-cloud environments, leveraging advantage actor-critic learning to optimize online resource allocation decisions under workload uncertainties and time-varying system conditions.

The key advantage of gradient-based parallelism lies in its simplicity and direct correspondence to distributed edge deployments. However, the on-policy nature of A3C and A2C limits sample reuse [20], and communication overhead of frequent gradient exchanges can become problematic in bandwidth-constrained edge environments [91]. These limitations motivate experience-based methods that decouple data collection from learning.

Experience-Based Parallelism. Experience-based parallel methods separate the roles of actors (collecting experiences) and learners (training models), enabling massive parallelization of data collection while centralizing computation-intensive learning. Actors execute policies to gather transitions (s, a, r, s') and send them to a shared replay buffer; learners sample from this buffer to train models off-policy. This decoupling allows actors to use slightly outdated policies, dramatically increasing scalability.

Ape-X DQN [92] scales DQN to hundreds of parallel actors by distributing experience collection while maintaining a single prioritized replay buffer. This architecture significantly reduces communication overhead compared to gradient-based methods, as actors only transmit compact experiences rather than high-dimensional gradients. Actors periodically synchronize with the latest policy and contribute experiences with computed priorities, while a central learner samples high-priority transitions for training. This architecture achieved state-of-the-art performance on Atari benchmarks, demon-

strating the power of massive parallelization. IMPALA (Importance Weighted Actor-Learner Architecture) [93] extends this paradigm to policy-based methods using V-trace off-policy correction, which adjusts for the discrepancy between actor and learner policies. IMPALA's architecture supports massive actor parallelization, making it suitable for planet-scale deployments. R2D2 (Recurrent Replay Distributed DQN) [94] adds recurrent neural networks to handle partial observability, storing and replaying entire episode sequences rather than individual transitions.

In large-scale IoT-Edge-Cloud systems, experience-based parallelism enables learning from massive distributed data sources. Zhang et al. [95] employed Ape-X DQN for building demand response control, balancing thermal comfort and energy consumption across edge-cloud infrastructure. Wang et al. [54] proposed TF-DDRL following IMPALA for IoT application scheduling, jointly minimizing response time, energy consumption, and monetary cost. Wang et al. [31] developed ReinFog integrating IMPALA and R2D2 for adaptive resource management, reducing response time and energy consumption in dynamic environments.

Experience-based methods offer unparalleled scalability, with near-linear speedup as actors increase [92, 93]. However, they require substantial infrastructure (centralized replay buffers, high-bandwidth connections to learners) and introduce complexity in managing actor-learner synchronization [92–94]. The choice between gradient-based and experience-based parallelism depends on deployment constraints: gradient-based methods suit moderate-scale edge deployments with good connectivity, while experience-based methods excel in planetary-scale IoT scenarios where data collection dominates computational costs.

Comparative Analysis and Design Guidelines

Single-agent architectures provide foundational capabilities for IoT-Edge-Cloud resource management, offering coherent global optimization when centralized control is feasible and beneficial. Table 2.3 systematically compares SARL methods across their key characteristics, strengths, and limitations in IoT-Edge-Cloud resource management.

Strengths. SARL architectures offer several fundamental advantages. Unified global policies enable coherent system-wide optimization, avoiding coordination challenges and suboptimal equilibria inherent to multi-agent approaches. Mature theoretical foundations provide convergence guarantees and performance bounds under well-established assumptions. Implementation simplicity reduces development complexity, as practitioners manage a single policy rather than coordinating multiple interacting agents. Direct policy deployment eliminates runtime coordination overhead, as all system components execute the same policy without inter-component communication. Well-established open-source implementations (e.g., Stable Baselines3, RLlib) accelerate development and ensure reproducibility.

Challenges and Limitations. SARL faces fundamental scalability and robustness challenges as system complexity increases. As the number of managed resources grows, global state representation becomes intractable, and centralized policy networks struggle with exponentially expanding state-action spaces. Centralized controllers create single points of failure, where controller unavailability halts resource management system-wide. Disseminating global state and distributing policy updates consume substantial bandwidth, particularly problematic in connectivity-limited edge networks. Processing high-dimensional global state imposes computational demands that may exceed edge controller capabilities. Single global policies struggle to adapt to heterogeneous local conditions when different edge regions exhibit drastically different resource characteristics, workload patterns, or network conditions.

Design Guidelines. Method selection should be driven by action space characteristics and sample efficiency requirements. Value-based methods (DQN family) suit discrete action spaces with manageable cardinality, while policy-based methods (TRPO, PPO) handle continuous control and high-dimensional discrete problems. Off-policy actor-critic methods (DDPG, TD3, SAC) excel when sample efficiency is critical, leveraging experience replay to maximize learning from expensive environment interactions. Training architecture should scale with deployment size: serial training suffices for moderate deployments, gradient-based parallelism (A3C, A2C) suits edge clusters with re-

liable connectivity, and experience-based parallelism (Ape-X, IMPALA, R2D2) enables planetary-scale systems with massive data generation. Deployment considerations include ensuring model size and inference latency align with edge controller capabilities, evaluating robustness to distribution shift through domain randomization or online adaptation, and implementing monitoring with failover mechanisms to detect policy failures and switch to fallback controllers when necessary.

2.4.2 Multi-Agent Architectures (MARL)

While SARL approaches offer coherent global optimization, they face fundamental scalability barriers in large-scale IoT-Edge-Cloud systems. The centralized decision-making paradigm struggles when: (i) the state space grows exponentially with the number of devices, rendering global state representation intractable [20]; (ii) communication latency between edge nodes and central controllers violates real-time constraints [52]; (iii) heterogeneous edge domains (e.g., industrial IoT, smart cities, autonomous vehicles) exhibit conflicting local objectives that cannot be reconciled through a single global policy [96]. MARL addresses these limitations by decomposing the resource management problem across multiple autonomous agents, each responsible for local decision-making within its domain (e.g., an edge server, a network slice, or a geographical region).

We organize MARL approaches based on their coordination mechanisms. *Independent Learning Approaches* (Section 2.4.2) treat other agents as part of the environment, learning without explicit coordination, suitable for scenarios with weak inter-agent dependencies. *Centralized Training with Decentralized Execution (CTDE) Approaches* (Section 2.4.2) leverage global information during training to learn coordinated policies while maintaining distributed execution, the predominant paradigm for edge computing where training can occur in resource-rich cloud environments while execution must be distributed across edge nodes.

Independent Learning Approaches

Independent learning approaches apply single-agent DRL methods to each agent independently, treating other agents' actions and policies as part of stochastic environment

Table 2.3: Systematic Comparison of Single-Agent DRL Methods for IoT-Edge-Cloud Continuum Resource Management

Category	Method	Year	Action Space	Key Characteristics	Literature	Policy Type	Strengths	Limitations
Serial – Value-Based	DQN	2015	Discrete	Experience replay; Target network; Pioneering deep Q-learning	[76]	Off	Stable training; Mature implementations; Convergence guarantees	Discrete actions only; Exponential action space growth; Cannot handle continuous control
	Double DQN	2016	Discrete	Decoupled action selection/evaluation; Reduces overestimation	[77]			
	Dueling DQN	2016	Discrete	Value-advantage decomposition; Efficient with sparse rewards	[78]			
	Rainbow	2018	Discrete	6 orthogonal DQN extensions; Superior sample efficiency	[79]			
Serial – Policy-Based	TRPO	2015	Cont/Disc	KL-constrained updates; Monotonic improvement; Second-order optimization	[81]	On	Handles continuous and high-dimensional actions; Direct policy optimization	On-policy limits sample reuse; Requires continuous environment interaction
	PPO	2017	Cont/Disc	Clipped surrogate objective; First-order efficiency; Production-grade	[26]; [82]			
Serial – Actor-Critic	DDPG	2016	Continuous	Deterministic policy; DQN-style replay for continuous control	[85]	Off	Off-policy sample reuse; Combines policy and value learning	Training complexity; Hyperparameter sensitivity; Requires careful tuning
	TD3	2018	Continuous	Twin critics; Delayed policy updates; Target smoothing	[86]			
	SAC	2018	Continuous	Maximum entropy framework; Auto temperature tuning; Robust	[87]			
Parallel – Gradient-Based	A3C	2016	Cont/Disc	Asynchronous workers; Advantage function; Immediate updates	[89]; [27]	On	Accelerated training; High exploration diversity; Natural distributed architecture	On-policy limits reuse; Gradient communication overhead; Staleness issues
	A2C	2016	Cont/Disc	Synchronous batching; Stable gradient aggregation; Consistency	[90]			
Parallel – Experience-Based	Ape-X	2018	Discrete	Hundreds of actors; Prioritized distributed replay	[95]	Off	Massive scalability; Actor-learner decoupling; Near-linear speedup	Centralized infrastructure needed; High bandwidth demands
	IMPALA	2018	Cont/Disc	Large-scale actors; V-trace off-policy correction	[54]; [31]			
	R2D2	2019	Discrete	Recurrent networks; Episode-level replay; Partial observability	[31]			

dynamics. Each agent i learns its own policy $\pi_i(a_i|o_i)$ based solely on local observations and rewards, without explicit knowledge of other agents. This paradigm offers extreme simplicity and scalability, as agents require no inter-agent communication and can be trained in parallel. We distinguish three categories: *Independent Value-Based Methods*, *Independent Policy-Based Methods*, and *Independent Actor-Critic Methods*.

Independent Value-Based Methods. Independent value-based methods learn separate state-action value functions $Q_i(o_i, a_i)$ for each agent, with policies derived through independent value maximization. Each agent treats other agents' behaviors as stochastic environment dynamics, inheriting the discrete action space suitability of value-based methods. Independent DQN (IDQN) is the predominant approach in this category.

IDQN [97] applies DQN to multi-agent scenarios where each agent i maintains its own Q-network $Q_{\theta_i}(o_i, a_i)$ with independent experience replay buffer $\mathcal{D}_i$. This fully decentralized approach achieves extreme scalability, as adding new agents requires no modification to existing agents' training processes.

In IoT-Edge-Cloud resource management, independent value-based methods have been successfully deployed in scenarios with loose coupling between edge domains. Cui et al. [98] employed IDQN to maximize long-term reward balancing throughput and power consumption in UAV network resource allocation, enabling each UAV to autonomously optimize user selection, power, and subchannel assignment through decentralized learning. Waqar et al. [99] proposed an IDQN-based approach to maximize effective throughput in NOMA-enabled MEC systems, incorporating fingerprint-augmented states to mitigate non-stationarity from concurrent agent learning. Shahid et al. [100] employed IDQN to minimize handover failures for mobile edge connectivity in small-cell networks, incorporating topology-aware prioritized experience replay to enhance learning efficiency across dynamic network topologies.

The effectiveness of independent value-based methods is limited by discrete action spaces. Continuous resource allocation requires discretization, introducing approximation errors [58]. Furthermore, deep Q-functions may insufficiently capture complex state-action relationships in high-dimensional observation spaces [57, 101]. These limitations motivate independent policy gradient methods.

Independent Policy-Based Methods. Independent policy-based methods extend policy gradient methods to multi-agent settings by maintaining separate policies for each agent. Each agent directly optimizes its policy through gradient ascent on expected returns using local trajectories. This approach naturally handles continuous action spaces, suitable for scenarios requiring precise control. Independent PPO (IPPO) is the predominant approach due to its training stability.

IPPO [102] extends PPO's clipped surrogate objective to multi-agent settings. Each agent independently applies PPO updates, maintaining its own policy π_{θ_i}, old policy $\pi_{\theta_{i,old}}$, and critic network $V_{\psi_i}(o_i)$ for advantage estimation, all computed from local trajectories. IPPO inherits PPO's training stability while achieving multi-agent scalability.

Lin et al. [103] employed IPPO to maximize computing energy efficiency in vehicular edge computing networks, enabling each vehicular user equipment to autonomously learn offloading policies through fully distributed training without information sharing while jointly optimizing VEC server selection and power allocation. Liu et al. [104] developed an IPPO-based approach for time-sensitive IoT applications in edge networks, incorporating hybrid action space optimization to jointly handle discrete resource unit selection and continuous power adjustment while guaranteeing differentiated Quality-of-Service (QoS) requirements among heterogeneous IoT devices.

Independent policy gradient methods achieve continuous control but face training instability in strongly-coupled scenarios [105]. Non-stationary environments also cause oscillatory behavior when agents rapidly adjust policies [106]. While effective in weakly-coupled scenarios, these methods lack mechanisms to explicitly model other agents.

Independent Actor-Critic Methods. Independent actor-critic methods maintain separate actor-critic pairs for each agent, where agent i independently learns both a policy $\pi_{\theta_i}(a_i|o_i)$ (actor) and value function $Q_{\psi_i}(o_i, a_i)$ (critic). This approach combines continuous action capability with sample efficiency through off-policy learning and experience replay. Independent DDPG (IDDPG) serves as the primary approach in this category.

IDDPG [105] applies DDPG to multi-agent scenarios where each agent maintains its own actor network, critic network, and corresponding target networks. Each agent independently maintains experience replay buffer $\mathcal{D}_i$ and updates networks through

independent gradient descent. This achieves deterministic policy optimization without inter-agent coordination.

Zheng et al. [107] employed IDDPG to minimize Age of Information in wireless-powered IoT networks, where each mobile node independently learns policies for relay selection, channel allocation, transmission duration, and power control through separate actor-critic pairs with independent experience replay.

The effectiveness of independent actor-critic methods is limited by environment non-stationarity from concurrent policy updates [108]. When agents' policies evolve simultaneously, critic value estimation becomes unreliable as environment dynamics shift, potentially causing training instability or convergence to suboptimal equilibria.

Centralized Training with Decentralized Execution (CTDE) Approaches

Independent learning approaches struggle under strong inter-agent dependencies, as agents treat each other's evolving policies as environment dynamics, leading to non-stationarity and suboptimal coordination [105, 106]. CTDE addresses this by exploiting a key asymmetry in IoT-Edge-Cloud deployments: training leverages global information in resource-rich cloud environments, while execution remains distributed across edge nodes using only local observations. We distinguish two CTDE paradigms: *Value Decomposition Methods* factorize global value functions for decentralized discrete action selection, while *Centralized Critic Methods* employ critics with global information to guide distributed actors for both discrete and continuous control.

Value Decomposition Methods. Value decomposition methods address coordinated discrete action selection while maintaining decentralized execution by factorizing global Q-functions into local components. The core idea is learning a joint action-value function $Q_{tot}(\mathbf{o}, \mathbf{a})$ that decomposes into agent-specific utilities $Q_i(o_i, a_i)$, such that global optimal actions can be obtained through independent local maximization.

Value Decomposition Networks (VDN) [109] achieves this through additive decomposition: $Q_{tot}(\mathbf{o}, \mathbf{a}) = \sum_{i=1}^{n} Q_i(o_i, a_i)$. VDN's simplicity enables efficient training but imposes restrictive assumptions that may fail under complex non-linear agent interactions. QMIX [110] relaxes this constraint through monotonic value factorization using a

mixing network while maintaining $\frac{\partial Q_{tot}}{\partial Q_i} \geq 0$, enabling more expressive value functions through hypernetworks conditioned on global state.

In edge computing scenarios, value decomposition methods have proven particularly effective for discrete resource allocation problems. Raivi and Moh [111] employed VDN combined to minimize energy consumption and delay in UAV-enabled IoT for post-disaster data aggregation and computation offloading. Yu et al. [112] developed a QMIX-based delay minimization algorithm for heterogeneous UAV-swarm-enabled aerial edge computing networks, leveraging monotonic value function factorization to jointly optimize cooperative computation offloading and trajectory planning while balancing coverage and collaborative task distribution among UAVs with diverse capabilities.

Value decomposition methods provide decentralized execution guarantees through the Individual-Global-Max principle [113] and communication-free scalability [109]. However, their structural constraints and limitation to discrete actions restrict applicability in IoT-Edge-Cloud deployments with intricate inter-dependencies [66].

Centralized Critic Methods. Centralized critic methods implement CTDE by employing critics that observe global information during training to guide distributed actors, eliminating non-stationarity by directly modeling other agents' policies. This category supports both discrete and continuous action spaces.

Multi-Agent Deep Deterministic Policy Gradient (MADDPG) [105] pioneered CTDE actor-critic for continuous control. Each agent maintains a deterministic policy $\mu_{\theta_i}(o_i)$ and a centralized critic $Q_{\phi_i}(\mathbf{o}, \mathbf{a})$ observing global state during training, while execution remains decentralized. Counterfactual Multi-Agent (COMA) [114] addresses credit assignment through counterfactual advantages measuring each agent's marginal contribution, particularly effective for discrete action spaces. Multi-Agent Proximal Policy Optimization (MAPPO) [102] extends PPO through centralized value functions for advantage estimation, combining training stability with CTDE coordination.

Applications of centralized critic methods in IoT-Edge-Cloud systems span multiple domains. He et al. [53] employed MADDPG to jointly maximize cache hit ratio and minimize traffic load in cloud-to-edge proactive caching, combining hyperdimensional

computing with Transformer for content prediction and dynamic placement optimization. Subhan et al. [71] developed a COMA-based edge-assisted rate adaptation approach to maximize Quality of Experience (QoE) and fairness across multiple clients in 360 video streaming. Yao et al. [115] proposed a MAPPO-based approach to minimize task processing and transmission delays in crowd-edge computing under partial network observability while balancing loads across heterogeneous computation resources.

Centralized critic methods provide maximum flexibility, supporting both continuous and discrete action spaces through different algorithmic approaches. However, compared to value decomposition methods, they typically exhibit higher sample complexity and greater sensitivity to hyperparameters [106].

Comparative Analysis and Design Guidelines

Multi-agent architectures provide essential capabilities for managing large-scale, heterogeneous IoT-Edge-Cloud deployments, but their effectiveness depends critically on matching coordination mechanisms to application requirements. Table 2.4 systematically compares MARL methods across their key characteristics, strengths, and limitations in IoT-Edge-Cloud resource management.

Strengths. MARL architectures offer several fundamental advantages for distributed resource management. Distributed decision-making scales naturally with system size, avoiding computational and communication bottlenecks inherent to centralized control. The multi-agent architecture aligns naturally with the inherently distributed IoT-Edge-Cloud architecture, where autonomous edge nodes make local decisions with limited global visibility. Parallel learning across multiple agents accelerates exploration and training, as diverse agents encounter different state-action combinations simultaneously. CTDE approaches provide coordinated global optimization while maintaining decentralized execution, eliminating runtime communication overhead.

Challenges and Limitations. MARL introduces unique challenges absent or less severe in SARL. Each agent's environment includes other learning agents whose policies evolve during training, violating the Markov assumption and creating non-stationarity

that persists even when CTDE approaches mitigate it during training. Determining individual agents' contributions to collective outcomes remains challenging when rewards are sparse or global, requiring value decomposition or counterfactual methods that introduce architectural complexity and structural assumptions. Agents must learn compatible policies despite partial observability and limited communication, critical in bandwidth-constrained edge networks. CTDE methods require global information exchange during training, creating bandwidth demands in distributed edge deployments despite decentralized execution.

Design Guidelines. Coordination mechanism selection should be driven by inter-agent dependency strength. Independent learning suits weak dependencies and isolated resource pools, while CTDE approaches are essential for shared resources, cascading effects, or global optimization objectives. System scale dictates feasibility: independent learning scales linearly to thousands of agents, while CTDE methods scale to moderate counts before communication costs dominate. Agent architecture design must balance local state sufficiency with communication overhead, with discrete action spaces favoring value decomposition and continuous control requiring actor-critic methods.

2.4.3 SARL vs MARL

The choice between single-agent and multi-agent architectures depends on system scale, objective alignment, and communication infrastructure. SARL is well-suited when: (i) the number of managed entities (edge servers, IoT clusters, network slices) is moderate, allowing tractable global state representation; (ii) all system components share a coherent optimization objective expressible as a unified reward function (e.g., minimizing system-wide energy consumption, maximizing aggregate throughput); (iii) reliable communication exists between edge nodes and a central controller for policy dissemination. MARL becomes necessary when: (i) system scale grows to hundreds or thousands of entities, rendering global state spaces intractable for centralized policy networks; (ii) heterogeneous edge domains exhibit conflicting local objectives that cannot be reconciled through a single global policy (e.g., cross-operator federations where each operator maximizes its own profit, smart cities where traffic management and energy grids have

Table 2.4: Systematic Comparison of Multi-Agent DRL Methods for IoT-Edge-Cloud Continuum Resource Management

Category	Method	Year	Action Space	Key Characteristics	Literature	Coordination	Strengths	Limitations
Independent – Value-Based	IDQN	2017	Discrete	Per-agent replay; Local observations only; DQN extension	[98]; [99]; [100]	Implicit; Environment as others	Extreme scalability; No comm. overhead; Simple implementation	May converge to suboptimal equilibria; Training instability
Independent – Policy-Based	IPPO	2020	Cont/Disc	Clipped objective; Independent critics; PPO extension	[103]; [104]	Implicit; Environment as others	Training stability; Multi-agent scalability	Non-stationary learning; Suboptimal coordination
Independent – Actor-Critic	IDDPG	2017	Continuous	Separate actor-critic pairs; Independent replay; Deterministic policy	[107]	Implicit; Environment as others	Continuous control; Off-policy efficiency; Deterministic optimization	Non-stationarity; Training instability; Suboptimal equilibria
CTDE – Value Decomp.	VDN	2018	Discrete	Additive factorization; Decentralized argmax; Linear mixing	[111]	Explicit; Value mixing	Decentralized execution consistency; High scalability; Comm-free execution	Requires global state; Discrete actions only; Expressiveness limited by constraints
	QMIX	2018	Discrete	Monotonic mixing; Hypernetworks; Non-linear factorization	[112]			
CTDE – Centralized Critic	MADDPG	2017	Continuous	Deterministic policy; Shared replay buffer; DDPG extension	[53]	Explicit; Centralized critic guidance	Mitigates non-stationarity; Handles complex interactions; No structural constraints	High sample complexity; Hyperparameter sensitive; Requires global state
	COMA	2018	Discrete	Counterfactual advantages; Action enumeration; Explicit credit assignment	[71]			
	MAPPO	2021	Cont/Disc	Clipped objective; Variance reduction; PPO extension	[115]			

fundamentally different criteria); (iii) communication latency or bandwidth constraints prevent timely global state aggregation. Independent learning suits weakly-coupled scenarios where agents have minimal mutual influence, while CTDE methods are essential when strong inter-agent dependencies demand coordinated policies despite distributed execution.

The progression from SARL to MARL reflects the growing scale and heterogeneity of IoT-Edge-Cloud systems. However, all methods discussed in this section assume free access to training data, whether through centralized collection (SARL) or global state access via centralized critics (CTDE). In practice, privacy regulations, communication constraints, and data sovereignty requirements often render such data centralization infeasible. Section 2.5 addresses this challenge by introducing the federated learning paradigm, which enables collaborative training without sharing raw data.

2.5 Federated Training Paradigm

The standard training paradigm assumes unrestricted access to training data. However, this assumption is increasingly untenable in real-world IoT-Edge-Cloud deployments due to three fundamental constraints: (i) *privacy regulations* such as GDPR and HIPAA prohibit raw data transmission across administrative boundaries [33]; (ii) *communication costs* make continuous data uploading from resource-constrained IoT devices economically infeasible [116]; (iii) *data sovereignty requirements* mandate that data remains locally stored within specific jurisdictions [117].

Federated learning addresses these constraints by inverting the traditional paradigm: rather than moving data to models, it moves models to data. Each node maintains local training data and periodically exchanges only model updates with an aggregator, ensuring raw data never leaves its origin while enabling collaborative learning.

Figure 2.3 illustrates the taxonomy under this paradigm. *Centralized Federated Architectures* (Section 2.5.1) employ a central aggregator for weighted aggregation, providing faster convergence at the cost of potential bottlenecks. *Decentralized Federated Architectures* (Section 2.5.2) eliminate central aggregation through peer-to-peer exchange, offering superior fault tolerance at the cost of slower convergence.

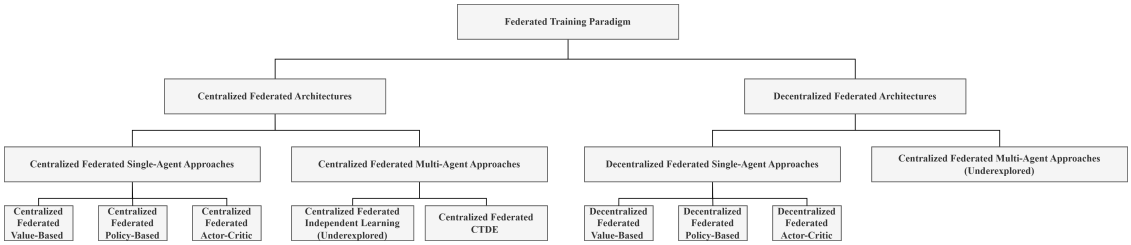


Figure 2.3: Taxonomy of DRL methods under federated training paradigm.

2.5.1 Centralized Federated Architectures

Centralized federated architectures implement a hierarchical topology where a central aggregator coordinates training across distributed computing nodes [117]. This architecture is particularly prevalent in IoT-Edge-Cloud scenarios where: (i) a trusted central entity (e.g., cloud service provider, edge orchestrator) can be designated as the aggregator [14]; (ii) communication infrastructure supports reliable uplink/downlink channels between nodes and the central server [117]; (iii) the number of participating nodes is sufficiently large that the communication overhead of centralized aggregation is outweighed by faster convergence compared to decentralized alternatives [47].

The centralized paradigm offers several practical advantages for resource management applications. First, synchronized global aggregation enables deterministic convergence analysis and theoretical guarantees under standard assumptions (e.g., bounded gradient dissimilarity, Lipschitz-continuous loss functions) [118]. Second, the central aggregator can implement sophisticated client selection strategies, prioritizing updates from nodes with more informative local data or better communication conditions [68]. Third, centralized coordination simplifies the implementation of privacy mechanisms such as secure aggregation and differential privacy, as cryptographic protocols can be executed between each client and a single trusted server [68].

Architecture Overview and Mechanisms

Central Aggregator Architecture. The centralized federated architecture consists of three primary components: (i) a *central aggregator* that maintains the global model and orchestrates training rounds, (ii) multiple *participating nodes* that store local training data

and compute model updates, and (iii) a *communication protocol* defining message exchange and synchronization strategies. In IoT-Edge-Cloud deployments, the aggregator typically resides on a cloud server or regional edge controller, while participating nodes may be edge servers, IoT gateways, or capable IoT devices [119].

The training process proceeds in iterative rounds. At round t, the aggregator broadcasts global model θ^t to selected clients $\mathcal{S}^t \subseteq \mathcal{N}$. Each client $k \in \mathcal{S}^t$ performs local training on private dataset $\mathcal{D}_k$ for E local epochs, computing updated parameters θ_k^{t+1}. Clients upload local updates $\Delta_k^t = \theta_k^{t+1} - \theta^t$ to the aggregator, which produces new global model θ^{t+1} through weighted aggregation and broadcasts it for the next round.

This architecture introduces three key design parameters [117]: *client participation fraction* C balances gradient estimate quality against communication overhead; *local epochs* E reduces communication frequency but may cause client drift under heterogeneous data distributions [14]; *batch size* B stabilizes local training but reduces exploration. These trade-offs require careful calibration based on deployment scenarios.

Centralized Aggregation Algorithms. The aggregation algorithm defines how the central server combines local updates into the global model, directly determining convergence speed and robustness to data heterogeneity.

Federated Averaging (FedAvg) [117] is the foundational aggregation algorithm. The server computes a weighted average $\theta^{t+1} = \sum_{k \in \mathcal{S}^t} \frac{n_k}{n} \theta_k^{t+1}$, where n_k is client k's local dataset size, ensuring clients with more data have proportionally greater influence. FedAvg's simplicity has made it the default baseline [14], though it may converge slowly under severe data heterogeneity [118].

Federated Proximal (FedProx) [118] addresses client drift by adding a proximal term $\frac{\mu}{2}\|\theta - \theta^t\|^2$ to each client's local objective, where $\mu > 0$ penalizes deviation from the global model. This regularization limits local model drift, improving convergence robustness under heterogeneous data distributions.

Adaptive Federated Optimization methods (FedAdam, FedYogi, FedAdagrad) [120] apply adaptive learning rate techniques at the server level. Rather than simple averaging, the server maintains momentum-based statistics of aggregated gradients, tracking first and second moments to dynamically adjust per-parameter learning rates. These

methods accelerate convergence on heterogeneous objectives by automatically adapting to different learning rate requirements across clients [121].

Convergence and Communication Analysis. Centralized federated learning introduces unique convergence challenges absent in standard training. The key issue is *client drift*: when clients perform multiple local updates on heterogeneous data, local models diverge from the global optimum, with drift accumulating across rounds [14]. Theoretical analysis establishes convergence under standard assumptions (strongly convex loss, Lipschitz gradients, bounded variance), but convergence degrades with data heterogeneity and excessive local computation [14]. For DRL applications, non-stationarity and sample correlation further complicate convergence, requiring careful tuning of local epoch numbers to avoid catastrophic divergence from extrapolation error [20].

Communication efficiency is critical in bandwidth-constrained IoT-Edge scenarios, as each round incurs bidirectional transfer of model parameters at megabyte scale across thousands of devices [15]. Techniques including gradient compression, periodic aggregation, and client sampling reduce communication overhead but typically trade off convergence speed or final model quality, requiring careful balance based on network conditions and application requirements [15].

Centralized Federated Single-Agent Approaches

Federating SARL methods for IoT-Edge-Cloud resource management presents unique challenges beyond standard federated supervised learning. DRL training involves sequential decision-making where each agent interacts with an environment, collects trajectories (sequences of state-action-reward transitions), and updates its policy based on these experiences. In a federated setting, edge nodes execute local policies to manage their respective resources, generating trajectories from their local environments (e.g., local workload patterns, network conditions, device characteristics) [68]. The core challenge is enabling these distributed nodes to collaboratively learn a unified global policy without sharing raw trajectory data [121].

The federated single-agent paradigm assumes all nodes ultimately deploy the same global policy π_θ but train on heterogeneous local environments. For instance, in a multi-

region edge content caching system, each edge region manages its cache using the same policy architecture, but local request patterns (state distributions) and cache hit rewards differ significantly across regions. Federated learning enables cross-region knowledge transfer while respecting data locality constraints [121].

Federated Value-Based Methods. Federating DQN-family methods applies federated aggregation to Q-networks. Each edge node k maintains a local Q-network Q_{θ_k} and experience replay buffer $\mathcal{D}_k$. During local training, nodes sample mini-batches from local buffers and perform standard DQN updates. After local training, nodes upload Q-network parameters to the central aggregator, which combines them into a global Q-network that is redistributed for the next training round.

Tong et al. [59] employed federated DQN with FedAvg-based aggregation and automated hyperparameter optimization to jointly minimize response time and energy consumption in multi-objective DAG task offloading, combining edge rank sorting with hyperparameter tuning for distributed collaborative learning while preserving privacy. Zhao et al. [122] developed federated DQN with FedAvg and Thompson sampling-based server selection to minimize system cost and maximize cache hit rate in edge caching, employing two-phase action selection to handle heterogeneous requests while reducing signaling overhead through smart agent selection.

However, federated value-based methods face the challenge of heterogeneous state-action distributions. When edge nodes observe different state distributions, the aggregated Q-function may average conflicting value estimates [123]. For example, an action might have high Q-value in a high-bandwidth region but low Q-value in a low-bandwidth region. Simple aggregation produces a mediocre Q-estimate potentially ineffective in both regions [123]. This challenge motivates personalized federated learning approaches or client clustering strategies based on environment similarity.

Federated Policy-Based Methods. Policy gradient methods (PPO) map more naturally to federated learning than value-based methods because policies can be directly averaged, and policy gradient estimators naturally aggregate across distributed rollouts. Each edge node k collects trajectories $\tau_k = \{(s_t, a_t, r_t)\}$ from its local environment us-

ing the current global policy π_θ, computes local policy gradients $\nabla_\theta J_k(\theta)$, and performs gradient ascent steps. The aggregator averages local policy parameters (or gradients) to update the global policy.

Wei et al. [55] employed federated PPO with personalized FedAvg aggregation to minimize delay and energy consumption in low-altitude vehicular fog computing through attention mechanism and contextual clustering. The multi-head attention mechanism enables agents to infer encoded global information while the personalized aggregation weights are based on cosine similarity between regional contexts and cluster centroids. Shen et al. [124] developed asynchronous federated PPO with staleness-aware FedProx aggregation to minimize latency and energy consumption in UAV-assisted DAG task offloading. The framework incorporates task priority models and dependency graphs to enable intelligent scheduling with real-time global updates through staleness-based mixing parameters.

A unique advantage of federated policy gradient methods is compatibility with importance sampling corrections [93]. Since nodes may perform multiple local updates before synchronization, the on-policy assumption (trajectories collected by the current policy) is violated when using stale global policies for local rollouts [117]. Standard off-policy corrections (e.g., importance sampling weights) can be applied during local advantage estimation to account for policy lag, maintaining convergence guarantees [93].

Federated Actor-Critic Methods. Off-policy actor-critic methods (DDPG, TD3, SAC) present both opportunities and challenges for federation. The opportunity lies in experience replay, which allows nodes to perform many local updates on their replay buffers before communicating, amortizing communication costs. The challenge is that both actor and critic networks can be federated, and critic networks are particularly sensitive to distribution shift as different nodes observe different state-action distributions.

Zhou et al. [125] employed federated DDPG with a weighted FedAvg to minimize delay and maximize cache hit rate in edge caching, using similarity-based weights for agent personalization. Zheng et al. [126] developed federated TD3 with FedAvg to maximize FL accuracy-to-energy ratio in EdgeIoT through data-proportional parameter aggregation and LSTM state prediction. Huang et al. [127] proposed transformer-

based federated SAC with attention-weighted FedAvg to minimize latency and energy in MEC, using multi-head attention for dynamic context-aware aggregation.

The key design consideration for federated actor-critic methods is determining which components to federate [15]. While actor networks can benefit from federation (as policies often generalize across similar environments), critic networks are more environment-specific [121]. Federated critics may suffer from averaging conflicting value estimates when nodes observe substantially different state-action distributions [121]. The choice between full federation (both actor and critic), partial federation (actor only), or hybrid approaches (federated actor with personalized critics) depends on environment similarity and communication constraints.

Centralized Federated Multi-Agent Approaches

Federating MARL introduces an additional layer of complexity. The system must not only handle federated learning across edge nodes but also coordinate multiple agents within each node's local environment [128]. This scenario arises naturally in hierarchical IoT-Edge-Cloud systems. For example, consider a smart city deployment where each city district operates an edge cluster managing traffic lights, streetlights, and surveillance cameras. Each district wants to learn coordinated policies for its infrastructure (multi-agent problem) while collaborating with other districts (federated learning) without sharing sensitive citizen data.

Existing federating MARL research predominantly focuses on federated CTDE approaches that leverage centralized training with decentralized execution structure to achieve stronger agent coordination in federated environments. CTDE methods require access to global state or joint observations during training (e.g., QMIX's mixing networks, MADDPG's centralized critics). In federated settings, sharing this global information across edge nodes violates privacy constraints, yet CTDE methods fundamentally rely on centralized information during training.

The solution is hierarchical information aggregation. Within each edge node k, local agents can access local global state (observations of all agents within that node), enabling local CTDE training. The central federated aggregator combines these locally-

trained CTDE structures. For instance, in Federated QMIX, each edge node k maintains local agent networks $\{Q_i^k\}$ and a local mixing network f_{mix}^k that combines them using local global state. During federation, both agent networks and mixing networks are aggregated across nodes: $Q_i^{global} = \sum_k \frac{n_k}{n} Q_i^k$ and $f_{mix}^{global} = \sum_k \frac{n_k}{n} f_{mix}^k$.

Yin et al. [69] employed federated QMIX with loss-weighted FedAvg to minimize migration latency and maximize task completion rate in mobile crowdsensing, integrating graph attention networks to extract DAG task dependencies while aggregating parameters based on loss function ratios. Lei et al. [129] developed federated MADDPG with loss-weighted FedAvg to maximize task completion rate in vehicular edge computing, proving the task offloading problem forms a non-cooperative potential game to ensure Nash equilibrium convergence. Zeng et al. [130] proposed a federated MAPPO-based approach with loss-weighted FedAvg to minimize average offloading delay in RIS-assisted V2X networks, aggregating actor and critic parameters based on objective function ratios to jointly optimize RIS reflection coefficients and beamforming vectors.

A critical design question is whether to federate the centralized training components (mixing networks, centralized critics) [131]. Current practice suggests federating both decentralized actors and centralized coordinators, as coordination patterns (e.g., how to balance load among agents) often generalize across environments even when state distributions differ [128]. However, this remains an active research area with limited theoretical guidance.

Comparative Analysis and Design Guidelines

Centralized federated architectures offer compelling advantages for IoT-Edge-Cloud resource management applications but require careful design to address inherent challenges.

Strengths. Centralized federation provides several key advantages. Synchronized global aggregation achieves faster convergence than decentralized alternatives. Privacy mechanism deployment is simplified as cryptographic protocols (secure aggregation, differential privacy) operate between clients and a single trusted server. The central aggregator enables sophisticated client management including adaptive selection, personalized

aggregation weights, and convergence-based early stopping. Theoretical convergence guarantees under standard assumptions (bounded heterogeneity, Lipschitz continuity) provide confidence for practical deployment.

Challenges and Limitations. Centralized architectures face fundamental scalability constraints. The central aggregator creates communication bottlenecks as all clients must exchange updates each round, consuming bandwidth that scales linearly with client count. Aggregator unavailability halts training system-wide, creating single points of failure. Simple averaging-based aggregation struggles under severe data heterogeneity, requiring specialized algorithms (FedProx) or client grouping strategies. Synchronous aggregation suffers from straggler problems where slow clients delay entire rounds, while asynchronous variants introduce staleness that degrades convergence quality.

Design Guidelines. Algorithm selection should match environment characteristics: standard FedAvg for homogeneous edge deployments, FedProx for heterogeneous data distributions with proximal coefficients tuned to distribution divergence, and adaptive methods (FedAdam, FedYogi) when convergence speed is critical. Hyperparameter tuning must balance communication frequency and convergence quality, with local epochs E set to avoid excessive drift in value-based methods and match episode lengths in policy gradient methods. For federated MARL, CTDE federates both actors and coordination structures with evaluation determining whether mixing networks remain local or federate. Privacy-performance trade-offs require choosing between differential privacy (strong guarantees, performance cost) and secure aggregation (minimal performance impact, computational overhead).

2.5.2 Decentralized Federated Architectures

Decentralized federated architectures implement peer-to-peer model aggregation where edge nodes exchange model updates directly with neighbors [119]. This architecture is particularly prevalent in IoT-Edge-Cloud scenarios where: (i) no mutually trusted central entity exists across all participating nodes (e.g., cross-operator edge federations,

collaborations between competing service providers) [47]; (ii) communication infrastructure supports reliable peer-to-peer links but edge-to-cloud uplinks are bandwidth-constrained or high-latency [119]; (iii) the number of participating nodes is sufficiently large that the scalability and fault tolerance advantages of decentralized topologies outweigh the slower convergence disadvantage [47].

The decentralized paradigm offers several practical advantages for resource management applications. First, communication bottlenecks are eliminated by distributing communication load across peer connections, with each node exchanging parameters with only a fixed number of neighbors regardless of total network size, enabling near-linear scaling to thousands of edge nodes [47]. Second, peer-to-peer topologies achieve graceful degradation by eliminating single points of failure, maintaining learning within connected components during node failures or network partitions and seamlessly reconverging once connectivity is restored [47]. Third, decentralized architectures strengthen privacy as no single entity observes all model updates, reducing data leakage risks and aligning with zero-trust security principles [119]. Fourth, decentralized topologies naturally align with edge infrastructure's inherent mesh connectivity, leveraging existing inter-edge communication channels (fiber links, metro area networks) rather than routing all traffic through distant cloud servers, reducing deployment complexity and operational costs [119].

Architecture Overview and Mechanisms

Peer-to-Peer Aggregation Architecture. The decentralized federated architecture consists of three primary components: (i) *distributed participating nodes*, each storing local training data and maintaining a local model replica; (ii) *communication topology*, defining connectivity relationships between nodes based on physical proximity, latency, or administrative boundaries; (iii) *peer exchange protocol*, specifying partner selection, exchange timing, and update combination strategies. In IoT-Edge-Cloud deployments, participating nodes are typically edge servers, gateways, or capable IoT devices [66]. Topologies can be static or dynamically adjusted based on network conditions [119].

The training process proceeds in locally-defined iterative rounds without global syn-

chronization. In each iteration, node i performs E local epochs on private dataset $\mathcal{D}_i$, then selects neighbors from its neighborhood set $\mathcal{N}_i$ and combines parameters through pairwise averaging $\theta_i^{t+1} \leftarrow (\theta_i^{t+1} + \theta_j^{t+1})/2$ to achieve local consensus. These operations typically execute asynchronously to tolerate network latency and heterogeneous processing speeds.

This architecture introduces three key design parameters: *topology degree* k trades convergence speed against communication overhead [47]; *local epoch count* E balances learning efficiency with parameter drift; *exchange frequency* impacts consensus speed versus bandwidth consumption [132]. These trade-offs require calibration based on network conditions and system scale.

Distributed Aggregation Mechanisms. Distributed aggregation mechanisms define how nodes combine parameters from neighbors to achieve global consensus, directly determining convergence speed, robustness to data heterogeneity, and adaptability to network dynamics.

Gossip averaging [133] is the foundational decentralized aggregation mechanism. In each communication round, node i selects a neighbor j from $\mathcal{N}_i$, and both nodes perform pairwise averaging: $\theta_i \leftarrow (\theta_i + \theta_j)/2$. This update rule ensures network-wide parameter average remains invariant, with all nodes converging exponentially to this average under connected graph assumptions. Gossip averaging requires no global coordination, supports asynchronous execution, and naturally tolerates node failures. However, it may converge slowly under severe data heterogeneity [47].

Consensus-based methods [47] achieve parameter aggregation through distributed protocols providing stronger security guarantees. Blockchain-inspired mechanisms ensure update integrity through distributed ledgers, achieving Byzantine fault tolerance [134]. Optimization frameworks such as ADMM provide alternative approaches through constrained optimization, converging faster than gossip for convex objectives [119]. These methods offer stronger theoretical guarantees but increase computational complexity through verification overhead.

Gradient tracking [135] addresses the limitation that parameter averaging may fail to minimize global objectives when local objectives differ significantly. Each node main-

tains both parameter and gradient estimates, with the protocol ensuring convergence to the true global gradient despite computing only local gradients. This enables gradient descent on the global objective while observing only local data, achieving linear convergence for strongly convex objectives [48].

Convergence and Communication Analysis. Unlike centralized federated learning where global aggregation ensures parameter consistency, decentralized architectures face consensus error from multi-hop information propagation [136]. When nodes perform local updates on heterogeneous data between peer exchanges, parameters across the network converge only asymptotically, with the spectral gap of the communication graph determining the exponential decay rate of consensus error [47]. Data heterogeneity and excessive local computation exacerbate this challenge. For DRL, the peer-to-peer nature introduces additional complications: nodes average parameters from neighbors whose policies have evolved independently on different environment interactions, creating misaligned value estimates and policy distributions [121].

Communication efficiency in decentralized settings exhibits distinct characteristics from centralized approaches. Topology design determines communication patterns, with node degree k controlling per-round volume and graph diameter determining convergence speed, while asynchronous exchange and network partition handling introduce unique trade-offs [133]. However, low-latency inter-edge links enable peer exchanges orders of magnitude faster than edge-to-cloud round trips, partially offsetting increased round counts for time-sensitive applications [47].

Decentralized Federated Single-Agent Approaches

Decentralized federated SARL extends the single global policy paradigm to peer-to-peer topologies, enabling edge nodes to collaboratively learn unified resource management policies without central coordination. Each edge node k maintains a local DRL agent (Q-network Q_{θ_k}, policy π_{θ_k}, or actor-critic pair), training on its private environment. Unlike centralized federation, nodes exchange parameters directly with neighbors through gossip protocols or structured topologies, gradually reaching consensus through pairwise averaging: $\theta_k \leftarrow (\theta_k + \theta_j)/2$.

Jeong et al. [137] employed consensus-based decentralized federated PPO with committee voting for 5G resource scaling, scoring local gradients via Euclidean distance among elected committee members while ensuring Byzantine resilience through adaptive consensus-based aggregation. Chai et al. [138] employed decentralized federated DQN with consensus-based attention-weighted aggregation to minimize delay and energy in Autonomous Aerial Vehicle (AAV)-assisted edge computing, computing neighbor influence weights via attention networks and aggregating distilled knowledge through KL divergence minimization. Zhou et al. [125] proposed a decentralized federated DDPG-based approach with consensus-based weighted selective aggregation to minimize delay in edge caching, preserving agent personalization while achieving global consensus.

The key design consideration for decentralized federated SARL is algorithm category selection and model component federation strategy. Off-policy methods (DQN, SAC) provide significant advantages as experience replay enables nodes to continue learning from local buffers while waiting for gossip exchanges, tolerating asynchronous communication and stale neighbor parameters [139]. On-policy methods (PPO) face more severe convergence issues as different node policies violate on-policy assumptions [139]. For actor-critic methods, actor networks can benefit from decentralized federation as policies generalize across similar environments, while federated critics are susceptible to averaging conflicting value estimates across heterogeneous state-action distributions [121]. For highly heterogeneous edge environments, gradient tracking provides an alternative through exchanging gradient estimates rather than parameters, optimizing the true global objective at the cost of increased computational overhead [121].

Decentralized Federated Multi-Agent Approaches

Decentralized federated MARL represents the most complex intersection in our taxonomy, requiring simultaneous coordination across three dimensions: multi-agent interaction within each edge node, federated learning across nodes under privacy constraints, and decentralized aggregation without central coordination. Convergence analysis must account for multi-agent non-stationarity, consensus dynamics of decentralized topologies, and heterogeneity across nodes [47, 105]. CTDE methods are particu-

larly challenging as they fundamentally rely on centralized components during training, while decentralized architectures lack such central entities [105].

To our knowledge, existing federated MARL for IoT-Edge-Cloud resource management primarily adopts centralized aggregation; fully decentralized federated MARL remains underexplored, making this an open research frontier. Several fundamental barriers impede progress. CTDE's incompatibility with decentralized architectures remains unresolved, as centralized training components cannot be naturally distributed across peer-to-peer topologies [47, 105]. Distributed credit assignment must account for remote agents operating under potentially stale policies using only local information [114]. Promising research directions include hierarchical decomposition strategies separating intra-node and inter-node coordination, gradient tracking extensions for multi-agent settings, communication-efficient partial information sharing protocols, and standardized benchmarks tailored to decentralized edge deployments [47].

Comparative Analysis and Design Guidelines

Decentralized federated architectures offer distinct advantages for planetary-scale and fault-tolerant IoT-Edge-Cloud deployments but introduce unique challenges in convergence analysis and system design.

Strengths. Decentralized federated architectures provide compelling advantages for large-scale and dynamic IoT-Edge-Cloud deployments. Elimination of communication bottlenecks enables superior scalability, with each node exchanging parameters with only a fixed number of neighbors regardless of network size, achieving near-linear scaling to thousands of edge nodes while centralized architectures encounter server bandwidth saturation. Peer-to-peer topologies eliminate single points of failure, exhibiting graceful degradation under node failures and maintaining consensus within connected components during network partitions. Decentralized architectures distribute privacy risk as no central entity observes all model updates, reducing high-value attack surfaces. Natural alignment with edge infrastructure's inherent mesh topologies leverages existing inter-edge communication channels, reducing deployment complexity and enabling continued operation in bandwidth-constrained or intermittently connected envi-

ronments.

Challenges and Limitations. Decentralized architectures introduce unique challenges that should be weighed against their advantages. Multi-hop consensus propagation requires more communication rounds to reach equivalent policy quality compared to centralized approaches. Topology design critically impacts convergence speed, requiring careful balance between connectivity and per-node communication overhead while accounting for physical constraints. Lack of global coordination prevents sophisticated orchestration strategies available in centralized settings, limiting decentralized architectures to uniform local algorithms. Limited theoretical understanding for DRL settings creates uncertainty in algorithm design and hyperparameter tuning, as convergence theory for decentralized reinforcement learning faces open questions due to policy nonstationarity and temporal correlation. Implementation complexity increases through distributed consensus requirements, sophisticated failure detection, and complex recovery mechanisms.

Design Guidelines. Topology selection should match deployment characteristics, with static topologies suited for stable infrastructure and dynamic topologies for environments with frequent node changes. Aggregation mechanism selection depends on environment heterogeneity: gossip averaging for reasonably aligned objectives, consensus-based methods for Byzantine fault tolerance requirements, and gradient tracking for highly heterogeneous environments despite increased computational overhead. Algorithm selection favors off-policy methods to tolerate asynchronous communication and stale parameters, with partial federation of actor-only components effective for actor-critic architectures.

2.5.3 Centralized vs Decentralized Federated DRL Architectures

Table 2.5 provides a systematic comparison of federated DRL methods. The choice between centralized and decentralized federated DRL architectures depends on deployment scale, trust models, and infrastructure characteristics. Centralized federated DRL is well-suited when: (i) a mutually trusted central entity (cloud service provider, edge

orchestrator) can be designated as the aggregator; (ii) deployment scale is moderate (typically fewer than several hundred nodes), where server bandwidth suffices to aggregate updates and broadcast global models; (iii) reliable high-bandwidth uplink/downlink channels exist between edge nodes and the central server; (iv) faster convergence is prioritized over fault tolerance, as centralized synchronized aggregation typically requires fewer communication rounds than decentralized consensus. Decentralized federated DRL is preferable when: (i) no mutually trusted central entity exists (cross-operator edge federations, competing service providers); (ii) deployment scale reaches thousands of nodes, where centralized server bandwidth becomes a bottleneck; (iii) communication infrastructure supports reliable peer-to-peer links but edge-to-cloud uplinks are bandwidth-constrained or high-latency (satellite networks, rural cellular); (iv) mission-critical applications demand continued operation despite infrastructure failures, favoring decentralized architectures' graceful degradation and tolerance to network partitions.

The architectures and training paradigms discussed thus far establish the foundational design space for DRL-based resource management in the IoT-Edge-Cloud continuum. However, deploying these methods in production environments reveals orthogonal challenges that cut across architectural choices, requiring additional techniques and mechanisms to enhance the core architectures. Section 2.6 examines these enhancements and their applications to IoT-Edge-Cloud resource management.

2.6 Advanced Techniques and Enhancements

Beyond control architecture and training paradigm choices, DRL research for IoT-Edge-Cloud resource management encompasses multiple orthogonal enhancement dimensions. This section systematically reviews these advanced techniques and their applications.

Table 2.5: Systematic Comparison of Federated DRL Methods for IoT-Edge-Cloud Continuum Resource Management

Architecture	Control Scope	Method Category	Representative Methods	Literature	Strengths	Limitations
Centralized	SARL	Value-Based	Cent-Fed-DQN	[59]; [122]	Faster convergence; Theoretical convergence guarantees; Sophisticated client selection; Simplified privacy deployment	Central communication bottleneck; Single point of failure; Straggler delays; Limited to moderate scale
		Policy-Based	Cent-Fed-PPO	[55]; [124]		
		Actor-Critic	Cent-Fed-DDPG	[125]		
			Cent-Fed-TD3	[126]		
			Cent-Fed-SAC	[127]		
	MARL	Independent	*Underexplored*			
		CTDE	Cent-Fed-QMIX	[69]		
			Cent-Fed-MADDPG	[129]		
			Cent-Fed-MAPPO	[130]		
Decentralized	SARL	Value-Based	Decent-Fed-DQN	[138]	No central bottleneck; Better fault tolerance; Leverages edge mesh connectivity; Distributed privacy risk; Near-linear scalability	Slower convergence; Consensus error from multi-hop; Topology design complexity; Limited practical guidance
		Policy-Based	Decent-Fed-PPO	[137]		
		Actor-Critic	Decent-Fed-DDPG	[125]		
	MARL	*Underexplored: Decentralized federated MARL remains an open research frontier.*				

2.6.1 Network Architectures

Standard DRL implementations employ fully-connected Multi-layer Perceptrons (MLPs) that treat timesteps independently, discarding temporal structure inherent to resource management. Edge workloads exhibit strong temporal correlations, and resource states evolve through autoregressive dynamics. Alternative network architectures explicitly model these dependencies, improving sample efficiency and policy quality.

Recurrent Neural Networks

Recurrent Neural Networks (RNNs) maintain hidden states capturing historical information, with Long Short-Term Memory (LSTM) networks addressing vanishing gradients through gating mechanisms and Gated Recurrent Units (GRUs) simplifying architecture with fewer parameters. Representative works include [126, 140, 141]. RNNs suit tasks with temporal dependencies and partial observability but can introduce Backpropagation Through Time (BPTT) complexity and susceptibility to overfitting on temporal patterns [142, 143].

Transformers and Attention Mechanisms

Transformers process sequences in parallel through multi-head self-attention, capturing long-range dependencies while avoiding vanishing gradients. Representative works include [53–55, 69, 127, 138]. Transformers suit multi-entity reasoning and offline learning but introduce quadratic attention complexity challenging edge deployment despite efficient variants [144].

Graph Neural Networks

Graph Neural Networks (GNNs) aggregate neighbor information over graph structures through message passing, with Graph Convolutional Networks (GCNs) and Graph Attention Networks (GATs) enabling policies to exploit network topology. Representative works include [24, 50, 145]. GNNs suit networked infrastructure with explicit topol-

ogy, enabling inductive generalization to unseen graph structures, but require graph-structured observations and face computational costs scaling with graph size [146].

2.6.2 Training Enhancements

Standard DRL training assumes agents learn policies from scratch for each environment. Edge deployments span diverse environments, and the ability to rapidly adapt learned knowledge to new settings is critical. Training enhancements enable DRL agents to leverage prior experience, reducing sample complexity and improving generalization.

Meta-Learning

Meta-learning, or learning to learn, trains models that quickly adapt to new tasks with minimal data by optimizing for rapid adaptation across task distributions. For DRL, this enables learning policies that quickly specialize to new edge environments through minimal fine-tuning. Representative works include [81, 147, 148]. Meta-learning suits scenarios requiring rapid adaptation to new deployments or periodic variations but requires diverse training task distributions and incurs higher meta-training computational costs [149, 150].

Transfer Learning

Transfer learning reuses knowledge from source domains through pre-training policies or value functions in resource-rich environments, then fine-tuning in target edge environments, with domain adaptation techniques handling distribution shifts through adversarial training or importance weighting. Representative works include [151, 152]. Transfer learning reduces target domain sample requirements when source and target share underlying structure but may cause negative transfer when domains differ significantly [153].

Multi-Task Learning

Multi-task learning trains single models simultaneously on multiple related tasks through shared representations, exploiting task commonalities via soft parameter sharing, hard parameter sharing, or attention mechanisms. Representative works include [56, 147]. Multi-task learning improves sample efficiency when tasks share structure and reduces total parameters, but negative transfer may occur when tasks conflict or require different features [154].

2.6.3 Model Compression

DRL policies employ deep networks with millions of parameters, creating prohibitive computational and memory requirements for resource-constrained edge devices. Model compression techniques reduce model size and inference cost while preserving policy quality, enabling deployment at the lowest tiers of the edge continuum.

Knowledge Distillation

Knowledge distillation transfers knowledge from large teacher models to compact students by minimizing divergence between outputs, exploiting that teacher soft targets contain richer information than hard targets for efficient student learning. Representative works include [46, 155]. Distillation enables cloud training with edge deployment and reduces federated learning communication overhead but introduces additional training stages and bounds student performance by teacher quality [51].

Quantization

Quantization reduces model size by lowering numerical precision from 32-bit floating point to 8-bit or lower integers, with post-training quantization applied after training and quantization-aware training simulating quantization effects during learning. Representative works include [156, 157]. Quantization provides memory reduction and inference speedup, particularly effective with hardware accelerators, but introduces numerical errors that may degrade performance for precision-sensitive tasks [158].

Pruning

Pruning removes redundant parameters through unstructured pruning of individual weights or structured pruning of entire neurons and layers, with magnitude-based selection and iterative refinement recovering performance. Representative works include [159, 160]. Pruning achieves high compression when models contain redundancy, with structured pruning providing practical speedup without specialized sparse libraries, but determining pruning rates requires experimentation and may fail for small models with limited redundancy [158].

2.6.4 Safety and Privacy Mechanisms

DRL policies deployed in production edge systems commonly need to satisfy hard constraints while protecting sensitive data. Safety mechanisms ensure constraint satisfaction during learning and deployment, preventing violations that may lead to service level agreement breaches, hardware damage, or safety hazards. Privacy mechanisms safeguard training data and model information from unauthorized access and inference attacks, essential for edge systems processing sensitive data across administrative boundaries.

Constrained Reinforcement Learning

Constrained RL formalizes constraint satisfaction as Constrained Markov Decision Processes (CMDPs) where agents optimize cumulative reward while satisfying cumulative constraints. Representative works include [25, 161]. Constrained RL provides principled frameworks for constraint optimization in safety-critical applications and service level agreements but assumes known constraint cost functions and may struggle with multiple competing constraints [162].

Safe Exploration

Safe exploration ensures agents do not violate constraints during learning by monitoring policies and intervening to prevent unsafe actions through pre-computed safety con-

straints or conservative action prediction. Representative works include [163, 164]. Safe exploration provides runtime protection for online learning in production systems but may interfere with policy improvement through frequent intervention or prove computationally infeasible in large state spaces [162].

Homomorphic Encryption

Homomorphic Encryption (HE) enables computation directly on encrypted data without decryption, allowing aggregators to process encrypted model updates while remaining unable to access plaintext values. For federated DRL, nodes encrypt local updates before transmission, and aggregators perform weighted averaging on ciphertexts, with only participating nodes possessing decryption keys. Representative works include [70, 165]. HE provides strong cryptographic privacy guarantees, protecting against malicious aggregators and eliminating trust requirements, but introduces substantial computational overhead (orders of magnitude slower than plaintext operations) that currently limits practical deployment to small models or computation-intensive phases [166].

2.7 Open Challenges and Future Directions

While DRL-based resource management in the IoT-Edge-Cloud continuum has achieved significant progress, fundamental challenges persist that prevent widespread production deployment. This section identifies six critical dimensions where substantial research gaps remain.

Scalability. Current DRL approaches face fundamental scalability barriers at planetary scale. SARL methods struggle with exponential state-action space growth, while MARL architectures rarely scale beyond hundreds of agents due to coordination overhead. Centralized federated DRL encounters server bandwidth bottlenecks, while decentralized variants suffer from slow consensus convergence across multiple network hops [58]. Promising directions include hierarchical decomposition across abstraction levels, graph neural networks for ultra-large-scale topologies, and adaptive topology design where nodes dynamically adjust neighborhoods based on learning dynamics.

Heterogeneity. The IoT-Edge-Cloud continuum exhibits extreme heterogeneity across device capabilities, data distributions, network conditions, and optimization objectives. Standard federated DRL struggles under severe data heterogeneity, often converging slowly or to suboptimal solutions. The fundamental tension lies between collaborative learning requiring shared representations and personalization requiring specialized models. Critical directions include personalized federated learning maintaining both global knowledge and client-specific adaptations, clustering-based approaches for targeted aggregation, and meta-learning frameworks enabling rapid adaptation to distribution shifts.

Dynamics and Non-stationarity. Edge environments exhibit complex temporal dynamics violating DRL's stationarity assumptions through multi-timescale workload fluctuations, topology evolution, and autocorrelated states. In federated settings, independently drifting client distributions create moving targets for convergence. The explore-exploit dilemma compounds this challenge as policies must adapt while avoiding catastrophic forgetting. Key directions include continual learning frameworks leveraging experience replay or elastic weight consolidation, online change detection mechanisms, and meta-learning approaches adapting to predictable non-stationarity patterns.

Resource Efficiency. Resource-constrained edge devices impose stringent limitations through limited memory, processing power, energy budgets, and network bandwidth. Model compression techniques provide partial solutions but face diminishing returns where aggressive compression degrades performance. The fundamental challenge balances model capacity required for complex decisions against hardware constraints. Future directions include neural architecture search optimized for edge constraints, split learning architectures balancing computation and communication, and hardware-software co-design for specialized DRL accelerators.

Privacy and Security. Privacy-preserving collaborative learning faces multifaceted challenges beyond standard federated scenarios. Centralized architectures create single points where inference attacks reconstruct private data, while decentralized learning introduces vulnerabilities through multi-hop propagation [33]. Adversarial threats including Byzantine attacks, model poisoning, and gradient inversion remain largely unaddressed in federated DRL. Critical directions include Byzantine-robust aggregation

handling policy gradient heterogeneity, secure multi-party computation for privacy-preserving training, and formal privacy leakage analysis.

Emerging Paradigms. Foundation models present transformative yet largely unexplored opportunities. Large Language Models demonstrate reasoning capabilities that could revolutionize decision-making through high-level policy generation, natural language interfaces, and few-shot adaptation via in-context learning. Generative models offer potential for workload prediction and synthetic trajectory generation. However, integrating billion-parameter models into resource-constrained environments poses significant challenges, and the black-box nature of LLM reasoning conflicts with verifiable safety requirements [167]. Critical directions include parameter-efficient fine-tuning, hybrid architectures combining lightweight DRL with occasional LLM consultation, and federated fine-tuning preserving privacy.

2.8 Summary

This chapter systematically organizes DRL-based resource management for IoT-Edge-Cloud continuum through a two-dimensional taxonomy separating control architecture from training paradigm. We comprehensively review related works across single-agent and multi-agent approaches under both standard and federated training, provide comparative analysis revealing fundamental trade-offs, survey orthogonal enhancement techniques, and identify open challenges with future research directions. The established frameworks and practical design guidelines advance toward intelligent, scalable, privacy-preserving resource management across the IoT-Edge-Cloud continuum.

Chapter 3

DRL-based Scheduling for Optimizing System Load and Response Time in Edge and Fog Computing Environments

Edge/fog computing has become the mainstream paradigm for IoT applications due to its low-latency capabilities. However, the exponential growth of IoT applications presents significant challenges including server overload that disrupts services and increases response time, dependent components that impose execution constraints, inherently dynamic and stochastic environments, and limited computational resources that prevent the use of computationally intensive optimal scheduling techniques. To address these challenges, we propose DRLIS, a Deep Reinforcement Learning-based scheduling algorithm that adaptively optimizes response time and balances server load for heterogeneous IoT applications. We implemented DRLIS as a practical scheduler and created an integrated edge-fog-cloud heterogeneous computing environment. Extensive experiments demonstrate that DRLIS significantly reduces execution cost by up to 55%, 37%, and 50% in load balancing, response time, and weighted cost, respectively, compared to metaheuristic algorithms and other reinforcement learning techniques.

This chapter is derived from:

- **Zhiyu Wang**, Mohammad Goudarzi, Mingming Gong, and Rajkumar Buyya, "Deep Reinforcement Learning-based Scheduling for Optimizing System Load and Response Time in Edge and Fog Computing Environments", *Future Generation Computer Systems (FGCS)*, Volume 152, Pages: 55-69, Elsevier, 2024.

3.1 Introduction

The past few years have witnessed the rapid rise of the Internet of Things (IoT) industry, enabling the connection of people to things and things to things, and facilitating the digitization of the physical world [168]. Meanwhile, with the explosive growth of IoT devices and various applications, the expectation for stability and low latency is higher than ever [169]. As the main enabler of IoT, cloud computing stores and processes data and information generated by IoT devices. Leveraging powerful computing capabilities and advanced storage technologies, cloud computing ensures the security and reliability of stored information. However, servers in the cloud computing paradigm are usually located at a long physical distance from IoT devices, and the high latency caused by long distances cannot efficiently satisfy real-time IoT applications. Prompted by these issues, edge and fog computing have emerged as popular computing paradigms in the IoT context. Although some researchers use the terms edge computing and fog computing interchangeably, we clearly define them in this chapter. We consider the case that uses only edge resources for real-time IoT applications as edge computing, and the case that uses edge and, whenever necessary, also utilizes cloud resources (along with edge resources in a seamless manner) as fog computing. Edge computing, as a decentralized computing architecture, brings processing, storage, and intelligent control to the vicinity of IoT devices [170]. This flexible architecture extends cloud computing services to the edge of the network. In contrast, the fog computing paradigm inherits the advantages of both cloud and edge computing [171], which not only provides powerful computational capabilities but also reduces the need to transfer data to the cloud for processing, analysis, and storage, thus reducing the inter-network distance. In the real world, edge and fog computing provide strong support for innovation and development in various fields. For example, in the field of smart healthcare, deploying edge computing nodes on wearable devices and medical devices can monitor patients' physiological parameters in real time and transmit the data to the cloud for analysis and diagnosis, realizing telemedicine and personalized medicine [172]; in the field of autonomous driving, deploying edge computing nodes on self-driving vehicles can perform real-time sensing and decision processing, enabling shorter response time and improving driving safety

[173].

However, the massive growth in the number of IoT applications and servers in fog computing environments also creates new challenges. Firstly, the execution time is expected to be minimized [18], which means that the applications should be processed by the best (i.e., the most powerful and physically closest) server. Besides, the load should be ideally balanced and distributed to run on multiple operating units. For example, by distributing requests across multiple servers in a seamless manner (as in serverless computing environments), load balancing can avoid overloading individual servers and ensure that each server handles a moderate load. This improves response times, overall system performance, and throughput, and also helps servers run more consistently. Therefore, improving the load balancing level of servers (i.e., lowering the variance of server resource utilization) while reducing the response time becomes an important but challenging problem for scheduling IoT applications on servers in edge/fog computing environments. Since this is an NP-hard problem, metaheuristic and rule-based solutions can be considered [174], [9]. However, these approaches often rely on omniscient knowledge of global information and require the solution proponent to have control over the changes. In the fog computing environment, there is often no regularity in server performance, utilization, and downtime. The number of IoT applications and the corresponding resource requirements are even more nearly random. Besides, in reality, Directed Acyclic Graphs (DAGs) are often used to model IoT applications [175], where nodes represent tasks and edges represent data communication between dependent tasks. The dependency among tasks introduces higher complexity in scheduling applications. Therefore, metaheuristic and rule-based solutions cannot efficiently cope with the IoT application scheduling problem in fog computing environments.

Deep Reinforcement Learning (DRL) is the product of combining deep learning with reinforcement learning, integrating the powerful understanding of deep learning on perceptual problems with the decision-making capabilities of reinforcement learning. In deep reinforcement learning, the agent continuously interacts with the environment, recording a large number of empirical trajectories (i.e., sequences of states, actions, and rewards), which are used in the training phase to learn optimal policies. In contrast to metaheuristic algorithms, agents in deep reinforcement learning are able to autonomously

sense and respond to changes in the environment, which allows deep reinforcement learning to solve complex problems in realistic scenarios. However, due to the limited computational resources of devices in fog computing environments [176], the computational requirements of complex Deep Neural Networks (DNNs) are often not supported [177]. Therefore, how to balance implementation simplicity, sample complexity, and solution performance becomes a key research problem in applying deep reinforcement learning to fog computing environments to cope with complex situations.

To address the above challenges, we propose a Deep Reinforcement Learning-based IoT application Scheduling algorithm (DRLIS), which employs Proximal Policy Optimization (PPO) [40] technique for solving the IoT applications scheduling problem in fog computing environments. DRLIS can effectively optimize the load balancing cost of the servers, the response time cost of the IoT applications, and their weighted cost. Besides, by using clipped surrogate objective to limit the magnitude of policy updates in each iteration and being able to perform multiple iterations of updates in the sampled data, the convergence speed of the algorithm is improved. Moreover, considering the limited computational resources and the optimization objective under study, we design efficient reward functions. The main contributions of this chapter are:

- We propose a weighted cost model regarding DAG-based IoT applications' scheduling in fog computing environments to improve the load balancing level of the servers while minimizing the response time of the application. In addition, we adapt this weighted cost model to make it applicable to DRL algorithms.

- We propose a DRL-based algorithm (DRLIS) to solve the defined weighted cost optimization problem in dynamic and stochastic fog computing environments. When the computing environment changes (e.g., requests from different IoT applications, server computing resources, the number of servers), it can adaptively update the scheduling policy with a fast convergence speed.

- Based on DRLIS, we implement a practical scheduler in the FogBus2 function-as-a-service framework[1] [44] for handling scheduling requests of IoT applications in heterogeneous fog and edge computing environments. We also extend the func-

[1]Please refer to [44, 178] for detailed description of the FogBus2 framework

tionality of the FogBus2 framework to make different DRL techniques applicable to it.

- We conduct practical experiments and use real IoT applications with heterogeneous tasks and resource demands to evaluate the performance of DRLIS in real system setup. By comparing with common metaheuristics (Non-dominated Sorting Genetic Algorithm 2 (NSGA2) [179], Non-dominated Sorting Genetic Algorithm 3 (NSGA3) [180]) and other reinforcement learning algorithms (Q-Learning [181]), we demonstrate the superiority of DRLIS in terms of convergence speed, optimization cost, and scheduling time.

The rest of the chapter is organized as follows. Section 3.2 discusses related work and Section 3.3 presents the system model and problem formulation. The Deep Reinforcement Learning model for IoT applications in edge and fog computing environments is presented in Section 3.4. DRLIS is discussed in Section 3.5. Section 3.6 evaluates the performance of DRLIS and compares it with other counterparts. Finally, Section 3.7 concludes the chapter.

3.2 Related Work

In this section, we review the literature on scheduling IoT applications in edge and fog computing environments. The related works are divided into metaheuristic and reinforcement learning categories.

3.2.1 Metaheuristic

In the dependent category, Liu et al. [182] adopted a Markov Decision Process (MDP) approach to achieving shorter average task execution latency in edge computing environments. They proposed an efficient one-dimensional search algorithm to find the optimal task scheduling policy. However, this work cannot adapt to changes in the computing environment and is difficult to extend to solve complex weighted cost optimization problems in heterogeneous fog computing environments. Wu et al. [183] modeled the

task scheduling problem in edge and fog computing environments as a DAG and used an estimation of distribution algorithm (EDA) and a partitioning operator to partition the graph in order to queue tasks and assign appropriate servers. However, they did not practically implement and test their work. Sun et al. [184] improved the NSGA2 algorithm and designed a resource scheduling scheme among fog nodes in the same fog cluster, taking into account the diversity of different devices. This work aims to reduce the service latency and improve the stability of task execution. Although capable of handling weighted cost optimization problems, this work only considers scheduling problems in the same computing environment. Hoseiny et al. [185] proposed a Genetic Algorithm (GA)-based technique for minimizing the total computation time and energy consumption of task scheduling in a heterogeneous fog cloud computing environment. By introducing features for tasks, the technique can find a more suitable computing environment for each task. However, it does not consider the dependencies of different tasks in the application, and due to the use of metaheuristic algorithms, scheduling rules need to be manually set, which cannot adapt to changing computing environments. Ali et al. [186] proposed an NSGA2-based technique for minimizing the total computation time and system cost of task scheduling in heterogeneous fog cloud computing environments. Their work formulates the task scheduling problem as an optimization problem in order to dynamically allocate appropriate resources for predefined tasks. Similarly, due to the limitations of metaheuristic algorithms, this work requires the assumption that the technique has some knowledge of the submitted tasks to develop the scheduling policy and thus cannot cope with dynamic and complex scenarios.

3.2.2 Reinforcement Learning

In the dependent category, Shahidani et al. [187] proposed a Q-learning-based algorithm to reduce task execution latency and balance the load in a fog cloud computing environment. However, this work does not consider the inter-task dependencies and the heterogeneity of fog and cloud computing environments. Baek et al. [188] adapted the Q-learning algorithm and proposed an approach that aims at improving load balancing in fog computing environments. This work considers the heterogeneity of nodes in

fog computing environments but still assumes that the tasks within the application are independent of each other. Jie et al. [189] proposed a Deep Q-Network (DQN)-based approach to minimize the total latency of task processing in edge computing environments. This work formulates task scheduling as a Markov Decision Process while considering the heterogeneity of IoT applications. However, this work only considers the scheduling problem in edge computing environments and investigates only one optimization objective. Xiong et al. [76] adapted the DQN algorithm and proposed a resource allocation strategy for IoT edge computing systems. This work aims at minimizing the average job completion time but does not take into account more complex functions with multiple optimization objectives. Wang et al. [190] focus on edge computing environments and propose a deep reinforcement learning-based resource allocation (DRLRA) scheme based on DQN. This work targets to reduce the average service time and balance the resource usage within the edge computing environment. However, the work does not consider the resources in fog computing environment, and the technique is not practically implemented and tested. Huang et al. [191] adopted a DQN-based approach to address the resource allocation problem in the edge computing environment. This work investigated minimizing the weighted cost, including the total energy consumption and the latency to complete the task. However, it does not consider the heterogeneity of servers in fog computing environments and assumes that the tasks are independent. Chen et al. [192] proposed an approach based on double DQN to balance task execution time and energy consumption in edge computing environments. Similarly, this work is only applicable to the edge environment and does not consider the dependencies between tasks. Zheng et al. [193] proposed a Soft Actor-Critic (SAC)-based algorithm to minimize the task completion time in an edge computing environment. This work focuses on the latency problem and the experiments are simulation-based. Zhao et al. [194] proposed a Twin Delayed DDPG (TD3)-based DRL algorithm. The goal of this work is to minimize the latency and energy consumption, but inter-task dependencies are not considered and the results are also simulation-based. Liao et al. [77] used Deep Deterministic Policy Gradient (DDPG) and Double Deep Q-Network (DQN) algorithms to model computation in an edge environment. This work aims to reduce energy consumption and latency but does not consider the fog environment and the heterogeneity

of devices. Sethi et al. [195] proposed a DQN-based algorithm to optimize energy consumption and load balancing of fog servers. Similarly, this work is simulation-based and does not consider the dependencies between tasks.

Table 3.1 presents the comparison of the related work with our proposed algorithm, in terms of application properties, architecture properties, algorithm properties, and evaluation. In the application properties section, the number of tasks included in the IoT application, and the dependencies between tasks are studied. In the architectural properties section, three aspects are studied including the IoT device layer, the edge/-fog layer, and the multi-cloud layer. For the IoT device layer, the application type and request type are identified. The real application section indicates that the work either deploys actual IoT applications, adopts simulated applications, or uses random data. The heterogeneous request type represents work considering that different IoT devices have different numbers of requests and different requirements. For the edge/fog layer, the computing environment and the heterogeneity of deployed servers are investigated. Besides, the multi-cloud layer studies whether the work considers the scenario of different cloud service providers with heterogeneity. In the algorithm properties section, we investigate the main technique on which each work is based and the corresponding optimization objectives. The evaluation section identifies whether the work is based on simulation or practical experiments. Recent works that we reviewed (e.g., [193], [194], [77], [195], [196], [197], [198]) have often used reinforcement learning approaches to deal with workload scheduling problems. This is because reinforcement learning can learn by interacting with the environment and continuously optimizing the policy through feedback signals (e.g., reward or penalty). This learning ability gives reinforcement learning an advantage when facing complex, dynamic environments [199], whereas metaheuristic techniques require manual adaptation and guidance.

3.3 System Model and Problem Formulation

In this section, we first introduce the topology of the IoT systems in the edge and fog computing environment. Then, we discuss the problem formulation. The key notations are listed in Table 3.2.

Table 3.1: A qualitative comparison of related works with ours

Works	Application Properties		Architectural Properties					Algorithm Properties					Evaluation
	Task Number	Dependency	IoT Device Layer		Edge/Fog Layer		Multi-Cloud Layer	Main Technique		Optimization Objectives			
			Real Applications	Request Type	Computing Environments	Heterogeneity				Time	Load Balancing	Weighted	
[182]	Single	Independent	○	Homogeneous	Edge	Homogeneous	×	Metaheuristic Algorithms	MDP	✓	×	×	Simulation
[184]	Multiple		◐	Homogeneous	Edge and Fog	Heterogeneous	×		NSGA2	✓	×	✓	Simulation
[185]	Single		○	Homogeneous	Edge and Fog	Heterogeneous	×		GA	✓	×	×	Simulation
[186]	Single		○	Homogeneous	Edge and Fog	Heterogeneous	×		NSGA2	✓	×	✓	Simulation
[183]	Multiple	Dependent	◐	Homogeneous	Edge and Fog	Heterogeneous	×		EDA	✓	×	✓	Simulation
[188]	Single	Independent	○	Homogeneous	Edge and Fog	Heterogeneous	×	Reinforcement Learning Techniques	Q-Learning	×	✓	×	Simulation
[187]	Single		○	Homogeneous	Edge and Fog	Homogeneous	×		Q-Learning	✓	✓	✓	Simulation
[189]	Single		◐	Homogeneous	Edge	Homogeneous	×		DQN	✓	×	×	Simulation
[76]	Multiple		◐	Homogeneous	Edge	Homogeneous	×		DQN	✓	×	×	Simulation
[191]	Multiple		◐	Heterogeneous	Edge	Homogeneous	×		DQN	✓	×	✓	Simulation
[190]	Single		◐	Homogeneous	Edge	Homogeneous	×		DQN	✓	✓	✓	Simulation
[192]	Single		◐	Heterogeneous	Edge	Homogeneous	×		Double DQN	✓	×	✓	Simulation
[193]	Single		◐	Homogeneous	Edge	Homogeneous	×		SAC	✓	×	×	Simulation
[194]	Single		◐	Homogeneous	Edge and Fog	Homogeneous	×		TD3	✓	×	✓	Simulation
[196]	Single		○	Homogeneous	Edge	Homogeneous	×		DQN	×	✓	×	Simulation
[77]	Single		◐	Homogeneous	Edge	Homogeneous	×		DDPG and DQN	✓	×	✓	Simulation
[197]	Single		○	Homogeneous	Edge	Homogeneous	×		DDPG	✓	×	✓	Simulation
[195]	Single		◐	Homogeneous	Edge and Fog	Homogeneous	×		DQN	×	✓	✓	Simulation
[198]	Multiple	Dependent	◐	Heterogeneous	Edge	Heterogeneous	×		GA and DQN	✓	×	×	Simulation
DRLIS	Multiple		●	Heterogeneous	Edge and Fog	Heterogeneous	✓		PPO	✓	✓	✓	Practical

●: Real IoT Application and Deployment, ◐: Simulated IoT Application, ○: Random

Table 3.2: List of key notations

Variable	Description	Variable	Description
S	The application set	$\psi^{ram}_{x_{S_{l_i}}}$	The variance of RAM utilization of the server set after the scheduling configuration $x_{S_{l_i}}$
S_l	One application (one task set)	$\Psi(\chi_l)$	The load balancing model after the scheduling configuration χ_l
S_{l_i}	One task	$\Psi(\chi)$	The load balancing model after the scheduling configuration χ
N	The server set	$\omega_{x_{S_{l_i}}}$	The total execution time (ms) for task S_{l_i} based on the scheduling configuration $x_{S_{l_i}}$
$x_{S_{l_i}}$	The scheduling configuration of task S_{l_i}	$\omega^{trt}_{x_{S_{l_i}}}$	The ready time (ms) for task S_{l_i} based on the scheduling configuration $x_{S_{l_i}}$
χ_l	The scheduling configuration of application S_l	$\omega^{trt}_{n_j,n_k}$	The time (ms) consumed for required data by task S_{l_i} to be sent from server n_j to server n_k
χ	The scheduling configuration of applications S	$P(S_{l_i})$	The parent tasks set of task $x_{S_{l_i}}$
$n^{cpu\_ut}_k$	The CPU utilization (%) of server n_k	$PS(S_{l_i})$	The server set to which the dependency tasks of task $x_{S_{l_i}}$ are assigned
n^{freq}_k	The CPU frequency (MHz) of server n_k	$\omega^{trans}_{n_j,n_k}$	The transmission time (ms) between server n_j and server n_k
$n^{ram\_ut}_k$	The RAM utilization (%) of server n_k	$\omega^{prop}_{n_j,n_k}$	The propagation time (ms) between server n_j and server n_k
$n^{ram\_size}_k$	The RAM size (GB) of server n_k	p_{n_j,n_k}	The packet size (MB) from server n_j to server n_k for task S_{l_i}
N^{cpu_uti}	The CPU utilization (%) of each server in server set N, denoted as a set	b_{n_j,n_k}	The data rate (bit/s) between server n_j and server n_k
N^{ram_uti}	The RAM utilization (%) of each server in server set N, denoted as a set	$CP(S_{l_i})$	Equals to 1 if S_{l_i} is on the critical path of application S_l, otherwise 0
$S^{ram}_{l_i}$	The minimum RAM required for executing task S_{l_i}	$\omega^{proc}_{x_{S_{l_i}}}$	The processing time (ms) for task S_{l_i} based on the scheduling configuration $x_{S_{l_i}}$
$\psi_{x_{S_{l_i}}}$	The load balancing model after the scheduling configuration $x_{S_{l_i}}$	$\Omega(\chi_l)$	The total execution time (ms) for application S_l based on the scheduling configuration χ_l
$\psi^{cpu}_{x_{S_{l_i}}}$	The variance of CPU utilization of the server set after the scheduling configuration $x_{S_{l_i}}$	$\Omega(\chi)$	The total execution time (ms) for the application set S based on the scheduling configuration χ

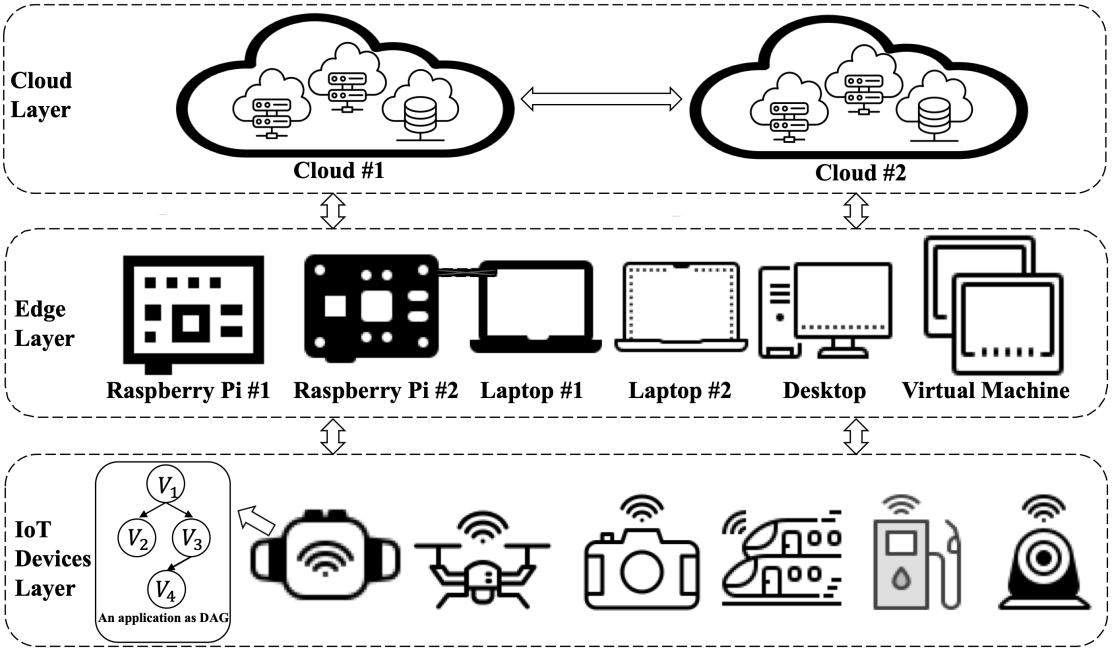


Figure 3.1: A view of the IoT system in fog computing

3.3.1 System Model

Fig. 3.1 represents a layered view of the IoT Systems in the fog computing environment. Consider $S = \{S_l | 1 \leq l \leq |S|\}$ as a collection of $|S|$ applications, where each application contains one or more tasks, denoted as $S_l = \{S_{l_i} | 1 \leq i \leq |S_l|\}$. The DAG $G = (V, E)$ is used to model an IoT application, as depicted in Fig. 3.2. A vertex $v_i = S_{l_i}$ denotes a certain task of the application, and an edge $e_{i,j}$ denotes the data flow between tasks v_i and v_j, so some tasks must be executed after predecessor tasks are completed. $CP(S_l)$ represents the critical path (i.e., the path with the highest cost) of the DAG, marked in red in the figure.

A set containing $|N|$ servers is used to process application set S, denoted as $N = \{n_k | 1 \leq k \leq |N|\}$. To reflect the heterogeneity of the servers, for each server n_k, $n_k^{cpu\_ut}$ represents its CPU utilization (%), n_k^{freq} represents its CPU frequency (MHz), $n_k^{ram\_ut}$ represents its RAM utilization (%), and $n_k^{ram\_size}$ represents its RAM size (GB). Moreover, $PS(S_{l_i})$ represents the server set to which the parent tasks of task S_{l_i} are assigned, and

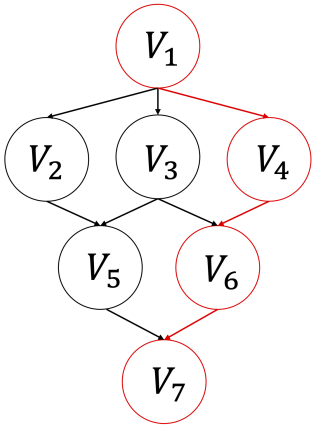


Figure 3.2: Sample IoT application with the critical path in red color

$\omega^{trans}_{n_j,n_k}$, $\omega^{prop}_{n_j,n_k}$, p_{n_j,n_k}, and b_{n_j,n_k} denote the transmission time (ms), the propagation time (ms), the packet size (MB), and the data rate (bit/s) between server n_j and server n_k, respectively.

3.3.2 Problem Formulation

Since an application contains one/multiple tasks, it may be executed on different servers. With a set of servers N, the scheduling configuration $x_{S_{l_i}}$ of a task S_{l_i} is defined as:

$$x_{S_{l_i}} = \{n_k\}, \tag{3.1}$$

where k shows the server's index. Accordingly, the scheduling configuration χ_l of an application S_l is equal to the set of the scheduling configuration of the tasks it contains, defined as:

$$\chi_l = \{x_{S_{l_i}} | S_{l_i} \in S_l, 1 \leq i \leq |S_l|\}. \tag{3.2}$$

The scheduling configuration χ of the application set S is equal to the set of scheduling configuration per application:

$$\chi = \{\chi_l | 1 \leq l \leq |S|\}. \tag{3.3}$$

In addition, we consider that for a given application, the execution model of tasks can be hybrid (i.e., sequential and/or parallel). That is, children tasks have some dependencies on the parent tasks that need to be executed after their completion, and we use $P(S_{l_i})$ to represent the parent task set of task S_{l_i} [200]. While tasks that do not depend on each other can be executed in parallel, and we use $CP(S_{l_i})$ to indicate that if a task S_{l_i} is located on a critical path of application S_l.

Load Balancing Model

The load balancing model is used to measure the resource balancing level of the server set N during the processing of the application set S. Regarding the server resource, both CPU and RAM are considered. For task S_{l_i}, the load balancing model $\psi_{x_{S_{l_i}}}$ is defined as:

$$\psi_{x_{S_{l_i}}} = a_1 \psi^{cpu}_{x_{S_{l_i}}} + a_2 \psi^{ram}_{x_{S_{l_i}}}, \tag{3.4}$$

where $\psi^{cpu}_{x_{S_{l_i}}}$ and $\psi^{ram}_{x_{S_{l_i}}}$ represent the CPU and RAM models, and a_1 and a_2 are the control parameters by which the weighted load balancing model can be tuned. They satisfy:

$$a_1 + a_2 = 1,\ 0 \leq a_1, a_2 \leq 1. \tag{3.5}$$

CPU model $\psi^{cpu}_{x_{S_{l_i}}}$ and RAM model $\psi^{ram}_{x_{S_{l_i}}}$ are defined as the variance of CPU and RAM utilization of the server set N after the scheduling configuration $x_{S_{l_i}}$:

$$\psi^{cpu}_{x_{S_{l_i}}} = \mathrm{Var}[N^{cpu\_uti}], \tag{3.6}$$

$$\psi^{ram}_{x_{S_{l_i}}} = \mathrm{Var}[N^{ram\_uti}], \tag{3.7}$$

where

$$x_{S_{l_i}} = \{n_k\}. \tag{3.1}$$

Correspondingly, for application S_l, the load balancing model $\Psi(\chi_l)$ is defined as the

sum of the load balancing models for each task processed by server set N:

$$\Psi(\chi_l) = \sum_{i=1}^{|S_l|} \psi_{x_{S_{l_i}}}. \tag{3.8}$$

Our main goal is to find the best-possible scheduling configuration for the application set S such that the variance of the overall CPU and RAM utilization of the server set N during the processing of the application set S can be minimized. Therefore, for the application set S, the load balancing model $\Psi(\chi)$ is defined as:

$$\Psi(\chi) = \sum_{l=1}^{|S|} \Psi(\chi_l) = \sum_{l=1}^{|S|} \sum_{i=1}^{|S_l|} \psi_{x_{S_{l_i}}}. \tag{3.9}$$

Response Time Model

We consider the response time model $\omega_{x_{S_{l_i}}}$ for the task S_{l_i} consisting of two components, the task ready time model $\omega^{trt}_{x_{S_{l_i}}}$ and the processing model $\omega^{proc}_{x_{S_{l_i}}}$:

$$\omega_{x_{S_{l_i}}} = \omega^{trt}_{x_{S_{l_i}}} + \omega^{proc}_{x_{S_{l_i}}}. \tag{3.10}$$

The task ready time model $\omega^{trt}_{x_{S_{l_i}}}$ represents the maximum time for the data required by the task S_{l_i} to arrive at the server to which it is assigned, defined as:

$$\omega^{trt}_{x_{S_{l_i}}} = max \;\; \omega^{trt}_{n_j,n_k}, \quad \forall n_j \in PS(S_{l_i}), \tag{3.11}$$

where $\omega^{trt}_{n_j,n_k}$ denotes the time consumed for required data by task S_{l_i} sent from server n_j to server n_k, and n_k is the server where the task S_{l_i} will be executed based on scheduling configuration $x_{S_{l_i}}$, and n_j represents the server where the parent task of task S_{l_i} is executed. Therefore, $\omega^{trt}_{n_j,n_k}$ depends on the transmission time $\omega^{trans}_{n_j,n_k}$ and the propagation time $\omega^{prop}_{n_j,n_k}$ for task S_{l_i} between server n_j and server n_k:

$$\omega^{trt}_{n_j,n_k} = \begin{cases} \omega^{trans}_{n_j,n_k} + \omega^{prop}_{n_j,n_k} & n_j \neq n_k, \\ 0 & n_j = n_k. \end{cases} \tag{3.12}$$

And the transmission time $\omega^{trans}_{n_j,n_k}$ can be calculated as:

$$\omega^{trans}_{n_j,n_k} = \frac{p_{n_j,n_k}}{b_{n_j,n_k}}, \tag{3.13}$$

where p_{n_j,n_k} represents the packet size from server n_j to server n_k for task S_{l_i}, and b_{n_j,n_k} represents the current bandwidth between server n_j and server n_k when the data for task S_{l_i} is transmitted.

The processing model $\omega^{proc}_{x_{S_{l_i}}}$ is defined as the time it takes for assigned server n_k to process the task S_{l_i} based on scheduling configuration $x_{S_{l_i}}$, and can be calculated as:

$$\omega^{proc}_{x_{S_{l_i}}} = \frac{S^{size}_{l_i}}{n^{freq}_k}, \tag{3.14}$$

where $S^{size}_{l_i}$ represents the required CPU cycles for task S_{l_i} and n^{freq}_k represents the CPU frequency of server n_k (for multi-core CPUs, the average frequency is considered).

Accordingly, the response time model $\Omega(\chi_l)$ for application S_l is defined as:

$$\Omega(\chi_l) = \sum_{i=1}^{|S_l|}(\omega_{x_{S_{l_i}}} \times CP(S_{l_i})), \tag{3.15}$$

where $CP(S_{l_i})$ equals to 1 if task S_{l_i} is on the critical path of application S_l, otherwise 0.

The main goal for the response time model $\Omega(\chi)$ is to find the best-possible scheduling configuration for the application set S such that the total time for the server set N processing them can be minimized. Therefore, for the application set S, the response time model $\Omega(\chi)$ is defined as:

$$\Omega(\chi) = \sum_{l=1}^{|S|}\Omega(\chi_l) = \sum_{l=1}^{|S|}\sum_{i=1}^{|S_l|}(\omega_{x_{S_{l_i}}} \times CP(S_{l_i})). \tag{3.16}$$

Weighted Cost Model

The weighted cost model is defined as the weighted sum of the normalized load balancing and normalized response time models. For task S_{l_i}:

$$\phi_{x_{S_{l_i}}} = w_1 \frac{\psi_{x_{S_{l_i}}} - \psi^{min}}{\psi^{max} - \psi^{min}} + w_2 \frac{\omega_{x_{S_{l_i}}} - \omega^{min}}{\omega^{max} - \omega^{min}}, \tag{3.17}$$

where $\psi_{x_{S_{l_i}}}$ and $\omega_{x_{S_{l_i}}}$ are the load balancing model and response time model of task S_{l_i}, and ψ^{min}, ψ^{max}, ω^{min}, and ω^{max} represent the minimum and the maximum value of the load balancing model and response time model, respectively. Moreover, w_1 and w_2 are the control parameters by which the weighted cost model can be tuned. The reason we use the normalized models instead of the original models is that the values of the two models may be in different ranges. For example, the load balancing model may have a value from 0 to 1, while the response time model may have a value from 0 to 100. We need to normalize them so that the model values are in the same range.

Accordingly, the weighted cost model for application S_l is defined as:

$$\Phi(\chi_l) = w_1 \times Norm(\Psi(\chi_l)) + w_2 \times Norm(\Omega(\chi_l)), \tag{3.18}$$

where $\Psi(\chi_l)$ and $\Omega(\chi_l)$ are obtained from Eq. 3.8 and Eq. 3.15, and *Norm* represents the normalization. The weighted cost model for the application set S is defined as:

$$\Phi(\chi) = w_1 \times Norm(\Psi(\chi)) + w_2 \times Norm(\Omega(\chi)), \tag{3.19}$$

where $\Psi(\chi)$ and $\Omega(\chi)$ are obtained from Eq. 3.9 and Eq. 3.16.

Therefore, the weighted cost optimization problem of IoT applications can be formu-

lated as:

$$min \quad \Phi(\chi) \tag{3.20}$$

$$\text{s.t.} \quad C1: \ Size(x_{S_{l_i}}) = 1, \ \forall x_{S_{l_i}} \in \chi_l \tag{3.21}$$

$$C2: \ 0 \leq n_k^{ram\_ut}, n_k^{cpu\_ut} \leq 1, \ \forall n_k \in N \tag{3.22}$$

$$C3: \ n_k^{freq}, n_k^{ram\_size} \geq 0, \ \forall n_k \in N \tag{3.23}$$

$$C4: \ S_{l_i}^{ram} < n_k^{ram\_size}, \ \forall S_{l_i} \in S_l, \ \forall n_k \in N \tag{3.24}$$

$$C5: \ \Phi(x_{S_{l_j}}) \leq \Phi(x_{S_{l_j}} + x_{S_{l_i}}), \forall S_{l_j} \in P(S_{l_i}) \tag{3.25}$$

$$C6: \ w_1 + w_2 = 1, \ 0 \leq w_1, w_2 \leq 1 \tag{3.26}$$

where $C1$ states that any task can only be assigned to one server for processing. $C2$ states that for any server, the CPU utilization and RAM utilization are between 0 and 1. Besides, $C3$ states that the CPU frequency and the RAM size of any server are larger than 0. Moreover, $C4$ denotes that any server should have sufficient RAM resources to process any task. Also, $C5$ denotes that any task can only be processed after its parent tasks have been processed, and thus the cumulative cost is always larger than or equal to the parent task. In addition, $C6$ denotes that the control parameters of the weighted cost model can only take value from 0 to 1, and the sum of them should be equal to 1.

The problem being formulated is presented to be a non-convex optimization problem, because there may be an infinite number of local optima in the set of feasible domains, and usually, the complexity of the algorithm to find the global optimum is exponential (NP-hard) [201]. To cope with such non-convex optimization problems, most work decomposes them into several convex sub-problems and then solves these sub-problems iteratively until the algorithm converges [202]. This type of approach reduces the complexity of the original problem at the expense of accuracy [203]. In addition, such approaches are highly dependent on the current environment and cannot be applied in dynamic environments with complex and continuously changeable parameters and computational resources [203]. To deal with this problem, we propose DRLIS to efficiently handle uncertainties in dynamic environments by learning from interaction with the environment.

3.4 Deep Reinforcement Learning Model

In reinforcement learning, the autonomous agent first interacts with the surrounding environment through action. Under the action and the environment, the agent generates a new state, while the environment gives an immediate reward. In this cycle, the agent interacts with the environment continuously and thus generates sufficient data. The reinforcement learning algorithm uses the generated data to modify its own action policy, then interacts with the environment to generate new data, and uses the new data to further improve its behavior. Formally, we use Markov Decision Process (MDP) to model the reinforcement learning problem. Specifically, the learning problem can be described by the tuple $< \mathbb{S}, \mathbb{A}, \mathbb{P}, \mathbb{R}, \gamma >$, where $\mathbb{S}$ denotes a finite set of states; $\mathbb{A}$ denotes a finite set of actions; $\mathbb{P}$ denotes the state transition probability; $\mathbb{R}$ denotes the reward function; $\gamma \in [0, 1]$ is the discount factor, used to compute the cumulative rewards.

We assume that the time $\mathbb{T}$ of the learning process is divided into multiple time steps t and the agent will interact with the environment at each time step and have multiple states S_t. At a particular time step t, the agent possesses the environment state $S_t = s$, where $s \in \mathbb{S}$. The agent chooses an action $A_t = a$ according to the policy $\pi(a|s)$, where $a \in \mathbb{A}$, and $\pi(a|s) = Pr[A_t = a|S_t = s]$ is the policy function, which denotes the probability of choosing the action a in state s. After choosing action a, the agent receives a reward $r = \mathbb{R}[S_t = s, A_t = a]$ from the environment based on the reward function $\mathbb{R}$, and it moves to the next state $S_{t+1} = s'$ based on the state transition function $P^a_{ss'} = \mathbb{P}[S_{t+1} = s'|S_t = s, A_t = a]$. The goal of the reinforcement learning agent is to learn a policy π that maximizes the expectation of cumulative discounted reward $\mathbb{E}_\pi[\sum_{t \in T} \gamma_t r_t]$.

Based on the weighted cost optimization problem of IoT applications in edge and fog computing environments, the state space $\mathbb{S}$, action space $\mathbb{A}$, and reward function $\mathbb{R}$ for the MDP are defined as follows:

- **State space** $\mathbb{S}$: Since the optimization problem is related to tasks and servers, the state of the problem consists of the feature space of the task currently being processed and the state space of the current server set N. Based on the discussion in Section 3.3, at the time step t, the feature space of the task S_{l_i} includes the task ID,

the tasks' predecessors and successors, the application ID to which the task belongs, the number of tasks in the current application, the estimate of the occupied CPU resources for the execution of the task, the task's RAM requirements, the estimate of the task's response time, etc. Formally, the feature space $\mathbb{F}$ for task S_{l_i} at the time step t is defined as follows:

$$\mathbb{F}_t(S_{l_i}) = \{f_t^y(S_{l_i}) | S_{l_i} \in S_l, 0 \leq y \leq |\mathbb{F}|\}, \tag{3.27}$$

where y represents the index of the feature in the task feature space $\mathbb{F}$, and $|\mathbb{F}|$ represents the number of features. Moreover, at the time step t, the state space of the current server set N includes the number of servers, each server's CPU utilization, CPU frequency, RAM utilization, and RAM size, and the propagation time and bandwidth between different servers, etc. Formally, the state space $\mathbb{G}$ for the server set N at the time step t is defined as:

$$\begin{aligned} \mathbb{G}_t(N) = \{|N|, g_t^z(n_k), h_t^q(n_j, n_k) | n_j, n_k \in N, \\ 0 \leq z \leq |g|, 0 \leq q \leq |h|\}, \end{aligned} \tag{3.28}$$

where g represents the state type that is related to only one server (i.e., CPU utilization), z represents its index, and $|g|$ represents the length of this type of state; besides, h denotes the state type that is related to two servers (i.e., propagation time), and similarly, q represents its index and $|h|$ represents the length of this type of state. Therefore, the state space $\mathbb{S}$ is defined as:

$$\mathbb{S} = \{S_t = (\mathbb{F}_t(S_{l_i}), \mathbb{G}_t(N)) | S_{l_i} \in S_l, t \in \mathbb{T}\}. \tag{3.29}$$

- **Action space** $\mathbb{A}$: The goal is to find the best-possible scheduling configuration for the application set S to minimize the objective function Eq. 3.20. Therefore, at the time step t, the action can be defined as the assignment of the server to the task S_{l_i}:

$$A_t = x_{S_{l_i}} = n_k. \tag{3.30}$$

Accordingly, the action space $\mathbb{A}$ can be defined as the server set N:

$$\mathbb{A} = N. \tag{3.31}$$

- **Reward function** $\mathbb{R}$: Since this is a weighted cost optimization problem, we need to define the reward function for each sub-problem. First, as the *penalty*, a very large negative value is introduced if the task cannot be processed on the assigned server for any reason. Also, for the load balancing problem, based on the discussion in section 3.3.2, the reward function r_t^{lb} is defined as:

$$r_t^{lb} = \begin{cases} \psi_{x_{S_{l_{i-1}}}} - \psi_{x_{S_{l_i}}} & succeed \\ penalty & fail, \end{cases} \tag{3.32}$$

where $\psi_{x_{S_{l_i}}}$ is obtained from Eq. 3.4. The value output by reward function r_t^{lb} is the difference between the load balancing models of the server set after scheduling the current task and the previous one. If the value of the load balancing model of the server set is reduced after scheduling the current task, the output reward is positive, otherwise it is negative. Beside, for the response time problem, based on the discussion in section 3.3.2, the reward function r_t^{rt} is defined as:

$$r_t^{rt} = \begin{cases} \omega_{x_{S_{l_i}}}^{mean} - \omega_{x_{S_{l_i}}} & succeed \\ penalty & fail, \end{cases} \tag{3.33}$$

where $\omega_{x_{S_{l_i}}}$ is obtained from Eq. 3.10, and $\omega_{x_{S_{l_i}}}^{mean}$ represents the average response time for task S_{l_i}. The value output by reward function r_t^{rt} is the difference between the average response time (the current response time is also considered) and the current response time for task S_{l_i}. If the current response time is lower than the average one, the output reward is positive, otherwise it is negative. The reward function r_t for the weighted cost optimization problem is defined as:

$$r_t = \begin{cases} w_1 \times Norm(r_t^{lb}) + w_2 \times Norm(r_t^{rt}) & succeed \\ penalty & fail, \end{cases} \tag{3.34}$$

where w_1 and w_2 are the control parameters, and $Norm$ represents the normalization process.

Currently, many advanced deep reinforcement learning algorithms (e.g., PPO, TD3, SAC) have been proposed by different researchers. They show excellent performance in different fields. PPO improves convergence and sampling efficiency by adopting importance sampling and proportional clipping [40]. TD3 (Twin Delayed DDPG) introduces a dual Q network and delayed update strategy to effectively solve the overestimation problem in the continuous action space [83]. SAC (Soft Actor-Critic) combines policy optimization and learning of Q-value functions, providing more robust and exploratory policy learning through maximum entropy theory [84]. These algorithms have achieved remarkable results in different tasks and environments. In our research problem, the agent's action and state space is discrete, which hinders the application of TD3, because it is designed for continuous control [204]. In addition, the original SAC only considers the problem of continuous space [84], although there are some works discussing how to apply SAC to discrete space, they usually need to adopt some special tricks and extensions, such as using soft-max or sample-prune techniques to accommodate discrete actions [205]. Besides, Wang et al. [206] shows that SAC requires more computation time and convergence time than PPO. Whereas our study focuses on edge and fog computing environments, where handling latency sensitivity and variation are important considerations for choosing the appropriate DRL algorithm. We choose PPO as the basis of DRLIS, because PPO is designed to be more easily adaptable to discrete action spaces [207] and we aim for the algorithm to converge quickly and perform well in diverse environments.

3.5 DRL-based Optimization Algorithm

Based on the above-mentioned MDP model, we propose DRLIS to achieve weighted cost optimization of IoT applications in edge and fog computing environments. In this section, we introduce the mathematical principle of the PPO algorithm and discuss the proposed DRLIS.

3.5.1 Preliminaries

The PPO algorithm belongs to the Policy Gradient (PG) algorithm which considers the impact of actions on rewards and adjusts the probability of actions [208]. We use the same notations as in section 3.3 to describe the algorithm. We consider the time horizon $\mathbb{T}$ is divided into multiple time steps t, and the agent has a policy π_θ for determining its actions and interactions with the environment. The objective can be expressed as adjusting the parameter θ to maximize the expected cumulative discounted rewards $\mathbb{E}_{\pi_\theta}[\sum_{t\in T}\gamma_t r_t]$ [40], expressed by the formula:

$$J(\theta) = \mathbb{E}_{\pi_\theta}[\sum_{t\in T}\gamma_t r_t]. \tag{3.35}$$

Since this is a maximization problem, the gradient ascent algorithm can be used to find the maximum value:

$$\theta' = \theta + \alpha\nabla_\theta J(\theta). \tag{3.36}$$

The key is to obtain the gradient of the reward function $J(\theta)$ with respect to θ, which is called the policy gradient. The algorithm for solving reinforcement problems by optimizing the policy gradient is called the policy gradient algorithm. The policy gradient can be presented as,

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta log\pi_\theta(a_t|s_t)A_\theta(a_t|s_t)], \tag{3.37}$$

where $A_\theta(a_t|s_t)$ is the advantage function at time step t, used to evaluate the action a_t at the state s_t. Here, the policy gradient indicates the expectation of $\nabla_\theta log\pi_\theta(a_t|s_t)A_\theta(a_t|s_t)$, which can be estimated using the empirical average obtained by sampling. However, the PG algorithm is very sensitive to the update step size, and choosing a suitable step size is challenging [209]. Moreover, practice shows that the difference between old and new policies in training is usually large [40].

To address this problem, Trust Region Policy Optimization (TRPO) [41] is proposed. This algorithm introduces importance sampling to evaluate the difference between the old and new policies and restricts the new policy if the importance sampling ratio grows large. Importance sampling refers to replacing the original sampling distribution with

a new one to make sampling easier or more efficient. Specifically, TRPO maintains two policies, the first policy $\pi_{\theta_{old}}$ is the current policy to be refined, and the second policy π_θ is used to collect the samples. The optimization problem is defined as follows:

$$\underset{\theta}{\text{maximize}} \quad \mathbb{E}_t[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}A_t] \tag{3.38}$$

$$\text{subject to} \quad \mathbb{E}_t[KL[\pi_{\theta_{old}}(\cdot|s_t), \pi_\theta(\cdot|s_t)]] \leq \delta, \tag{3.39}$$

where KL represents Kullback-Leibler Divergence, used to quantify the difference between two probability distributions [210], and δ represents the restriction of the update between old policy $\pi_{\theta_{old}}$ and new policy π_θ. After linear approximation of the objective and quadratic approximation of the constraints, the problem can be efficiently approximated using the conjugate gradient algorithm. However, the computation of conjugate gradient makes the implementation of TRPO more complex and inflexible in practice [211], [212].

To make this algorithm well applied in practice, the KL-PPO algorithm [40] is proposed. Rather than using the constraint function $\mathbb{E}_t[KL[\pi_{\theta_{old}}(\cdot|s_t), \pi_\theta(\cdot|s_t)]] \leq \delta$, the KL divergence is added as a penalty in the objective function:

$$L^{KLPEN}(\theta) = \mathbb{E}_t[r_t(\theta)A_t - \beta KL[\pi_{\theta_{old}}(\cdot|s_t), \pi_\theta(\cdot|s_t)]], \tag{3.40}$$

where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ is the ratio of the new policy and the old policy, obtained in Eq. 3.38, and the parameter β can be dynamically adjusted during the iterative process according to the KL divergence. If the current KL divergence is larger than the predefined maximum value, indicating that the penalty is not strong enough and the parameter β needs to be increased. Conversely, if the current KL divergence is smaller than the predefined minimum value, the parameter β needs to be reduced.

Moreover, another idea to restrict the difference between old policy $\pi_{\theta_{old}}$ and new policy π_θ is to use clipped surrogate function $clip$. The PPO algorithm using the clip function (CLIP-PPO) removes the KL penalty and the need for adaptive updates to simplify the algorithm. Practice shows CLIP-PPO usually performs better than KL-PPO [40]. Formally, the objective function of CLIP-PPO is defined as follows:

$$L^{CLIP}(\theta) = \mathbb{E}_t[min(r_t(\theta)A_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t]. \tag{3.41}$$

And $clip(r_t(\theta), 1-\epsilon, 1+\epsilon)$ restrict the ratio $r_t(\theta)$ into $(1-\epsilon, 1+\epsilon)$, defined as:

$$clip(r_t(\theta), 1-\epsilon, 1+\epsilon) = \begin{cases} 1-\epsilon & r_t(\theta) < 1-\epsilon \\ r_t(\theta) & 1-\epsilon \leq r_t(\theta) \leq 1+\epsilon \\ 1+\epsilon & r_t(\theta) > 1+\epsilon. \end{cases} \tag{3.42}$$

By removing the constraint function as discussed in TRPO, both PPO algorithms significantly reduce the computational complexity, while ensuring that the updated policy deviates not too large from the previous one.

3.5.2 DRLIS: DRL-based IoT Application Scheduling

Since CLIP-PPO usually outperforms KL-PPO in practice, we choose it as the basis for the optimization algorithm. DRLIS is based on the actor-critic framework, which is a reinforcement learning method combining Policy Gradient and Temporal Differential (TD) learning. As the name implies, this framework consists of two parts, the actor and the critic, and in implementation, they are usually presented as Deep Neural Networks (DNNs). The actor network is used to learn a policy function $\pi_\theta(a|s)$ to maximize the expected cumulative discounted reward $\mathbb{E}_\pi[\sum_{t\in T} \gamma_t r_t]$, while the critic network is used to evaluate the current policy and to guide the next stage of the actor's action. In the learning process, at the time step t, the reinforcement learning agent inputs the current state s_t into the actor network, and the actor network outputs the action a_t to be performed by the agent in the MDP. The agent performs the action a_t, receives the reward r_t from the environment, and moves to the next state s_{t+1}. The critic network receives the states s_t and s_{t+1} as input and estimates their value functions $V_{\pi_\theta}(s_t)$ and $V_{\pi_\theta}(s_{t+1})$. The agent then computes the TD error δ_t for the time step t:

$$\delta_t = r_t + \gamma V_{\pi_\theta}(s_{t+1}) - V_{\pi_\theta}(s_t), \tag{3.43}$$

where γ denotes the discount factor, as discussed in Section 3.3, and the actor network and critic network update their parameters using the TD error δ_t. DRLIS continues this process after multiple steps, as an estimate $\hat{A}_t$ of the advantage function A_t, which can

be written as:

$$\hat{A}_t = -V_{\pi_\theta}(s_t) + r_t + \gamma r_{t+1} + \cdots + \gamma^{T-t+1} r_{T-1} + \gamma^{T-t} V_{\pi_\theta}(s_T). \tag{3.44}$$

DRLIS maintains three networks, one critical network, and two actor networks (i.e., the old actor and the new actor), representing the old policy function $\pi_{\theta_{old}}$ and the new policy function π_θ, as discussed in Section 3.5.1. Algorithm 3.1 describes DRLIS for the weighted cost optimization problem in edge and fog computing environments.

Algorithm 3.1: DRLIS for weighted cost optimization

Input : new actor network Π_θ with parameter θ; old actor network $\Pi_{\theta_{old}}$ with parameter θ_{old}, where $\theta_{old} = \theta$; critic network V_μ with parameter μ; max time step T; update epoch K; policy objective function coefficient a_c; value function loss function coefficient a_v; entropy bonus coefficient a_e; clipping ratio ϵ

while *True* **do**
  $servers \leftarrow GetServers()$
  $task \leftarrow GetTask()$
  if $servers \neq servers_{old}$ **then**
    $agent \leftarrow InitializeAgent(servers)$
    $servers_{old} \leftarrow servers$
  end if
  $s_1 \leftarrow GeneralizeState(servers, task)$
  $\mathcal{D} \leftarrow InitializeBuffer()$
  for $t \leftarrow 1$ *to* T **do**
    $a_t \leftarrow \Pi_\theta(s_t)$
    $Schedule(task, a_t)$
    $r_t \leftarrow GetReward()$
    $servers \leftarrow GetServers()$
    if $servers \neq servers_{old}$ **then**
      break
    end if
    $task \leftarrow GetTask()$
    $s_{t+1} \leftarrow GeneralizeState(servers, task)$
    $u_t = (s_t, a_t, r_t)$
    $\mathcal{D}.Append(u_t)$
  end for
  $\hat{A}_t \leftarrow -V_\mu(s_t) + r_t + \gamma r_{t+1} + \cdots + \gamma^{T-t+1} r_{T-1} + \gamma^{T-t} V_\mu(s_T)$
  for $k \leftarrow 1$ *to* K **do**
    $L^{CLIP}(\theta) = \sum_1^t min(\frac{\Pi_\theta(a_t|s_t)}{\Pi_{\theta_{old}}(a_t|s_t)} \hat{A}_t, clip(\frac{\Pi_\theta(a_t|s_t)}{\Pi_{\theta_{old}}(a_t|s_t)}, 1-\epsilon, 1+\epsilon)\hat{A}_t$
    $L^{VF}(\mu) = \sum_1^t (V_\mu(s_t) - \hat{A}_t)^2$
    $L^{ET}(\theta) = \sum_1^t Entropy(\Pi_\theta(a_t|s_t))$
    $L(\theta, \mu) = -a_c L^{CLIP}(\theta) + a_v L^{VF}(\mu) - a_e L^{ET}(\theta)$
    update θ and μ with $L(\theta, \mu)$ by Adam optimizer
  end for
  $\theta_{old} \leftarrow \theta$
end while

We consider a scheduler that is implemented based on DRLIS. When this scheduler

receives a scheduling request from an IoT application, it obtains information about the set of servers currently available and initializes a DRL agent based on the information. This agent contains three deep neural networks, a new actor network Π_θ with parameter θ, an old actor network $\Pi_{\theta_{old}}$ with parameter θ_{old}, where $\theta_{old} = \theta$, and a critic network V_μ with parameter μ. After that, the scheduler obtains the information about the currently submitted task and generates the current state s_t based on the information regarding the task and servers. Inputting the state s_t to the new actor network Π_θ will output an action a_t, representing the target server to which the current task is to be assigned. The scheduler then assigns the task to the target server and receives the corresponding reward r_t, which is calculated based on Eq. 3.32, 3.33, 3.34. The reward r_t is essential for indicating the positive or negative impact of the agent's current scheduling policy on the optimization objectives (e.g., IoT application response time and servers load balancing level). Also, a tuple u_t with three values (s_t, a_t, r_t) will be stored in buffer $\mathcal{D}$. The scheduler repeats the process T times until sufficient information is collected to update the neural networks. When updating the neural networks, the estimate of the advantage function is first computed based on Eq. 3.44. Then the neural networks are optimized for K times. Both actor network and critic network use Adam optimizer, and the loss function is computed as:

$$L(\theta, \mu) = -a_c L^{CLIP}(\theta) + a_v L^{VF}(\mu) - a_e L^{ET}(\theta), \tag{3.45}$$

where $L^{CLIP}(\theta)$ is the policy objective function from Eq. 3.41, and $L^{VF}(\mu)$ is loss function for the state value function:

$$L^{VF}(\mu) = \sum_{t=1}^{T} (V_\mu(s_t) - \hat{A}_t)^2. \tag{3.46}$$

And $L^{ET}(\theta)$ is the entropy bonus for the current policy:

$$L^{ET}(\theta) = \sum_{t=1}^{T} Entropy(\Pi_\theta(a_t|s_t)). \tag{3.47}$$

In addition, a_c, a_v, and a_e are the coefficients. After updating the neural networks, the parameter θ of the new actor network Π_θ will be copied to the old actor network $\Pi_{\theta_{old}}$.

Assuming that there are N tasks, from Algorithm 3.1, the agent will update the policy K times after scheduling T tasks, so the complexity of the algorithm as $O(N + \frac{N}{T}K)$. In practical applications, both T and K as hyperparameters can be customized to suit different computational environments. Thus the computational complexity of the algorithm actually depends on the number of tasks N and can be written as $O(N)$. For the edge/-fog environment with limited computational resources, we consider this computational complexity to be acceptable.

3.5.3 Practical Implementation in the FogBus2 Framework

We extend the scheduling module of the FogBus2 framework[2] [44] to design and develop the DRLIS in practice for processing placement requests from different IoT applications in edge and fog computing environments.

FogBus2 is a lightweight container-based distributed/ serverless framework (realized using Docker microservices software) for integrating edge and fog/cloud computing environments. A scheduling module is implemented to decide the deployment of heterogeneous IoT applications, enabling the management of distributed resources in the hybrid computing environment. There are five main components within FogBus2 framework, namely *Master*, *Actor*, *RemoteLogger*, *TaskExecutor*, and *User*. Fig. 3.3 shows the relationship between different components in the FogBus2 framework, and the updated sub-components used to implement the reinforcement learning function.

- *Remote Logger*: It is designed for collecting and storing logs from other components, whether periodic or event-driven.
- *Master*: It contains the scheduling module of FogBus2, responsible for the registration and scheduling of IoT applications. It can also discover resources and self-scale based on the input load. We implement a reinforcement learning scheduling module in the *Scheduler & Scaler* sub-component. Besides, we extend the functionality of the *Profiler* and the *Message Handler* components to allow *Master* components to receive and handle information from other components for reinforcement learning scheduling.

[2]https://github.com/Cloudslab/FogBus2

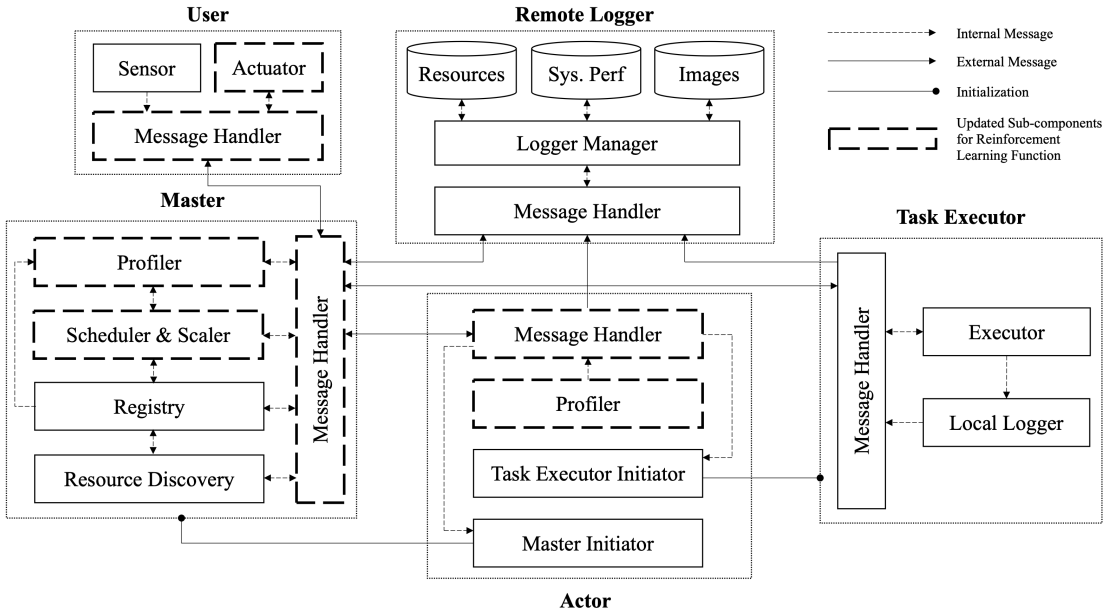


Figure 3.3: Updated Sub-Components for Reinforcement Learning in FogBus2 Framework

- *Actor*: It informs the *Remote Logger* and *Master* components of the computing resources of the corresponding node to coordinate the resource scheduling of the framework. Furthermore, it is responsible for launching the appropriate *Task Executor* components to process the submitted IoT application. We extend the functionality of the *Profiler* and the *Message Handler* components to allow system characteristics regarding servers to be passed to the reinforcement learning scheduling module in *Master* components.

- *Task Executor*: It is responsible for executing the corresponding tasks of the submitted application. The results are passed to the *Master* component.

- *User*: It runs on IoT devices and is responsible for processing raw data from sensors and users. It sends the processed data to the *Master* component and submits the execution request. We extend the functionality of the *Actuator* and the *Message Handler* components to allow information related to IoT applications to be passed to the reinforcement learning scheduling module in *Master* components.

Fig. 3.4 shows our implementation of the reinforcement learning scheduling module

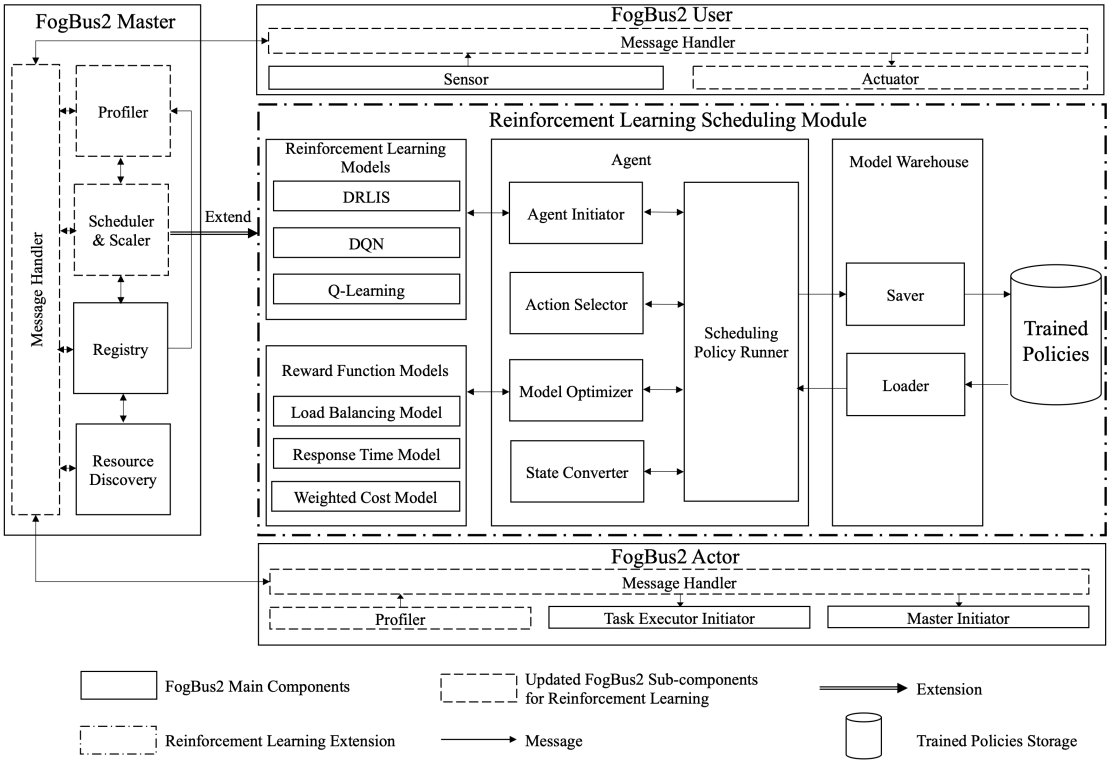


Figure 3.4: Reinforcement Learning Scheduling Module in FogBus2 Framework

in the FogBus2 framework. The module can be divided into four sub-modules: 1) Reinforcement Learning Models, 2) Rewards Models, 3) Reinforcement Learning Agent, and 4) Model Warehouse.

- *Reinforcement Learning Models*: This sub-module contains the reinforcement learning models. According to Algorithm 3.1, we implement a DRLIS-based model. In addition, to evaluate the performance of DRLIS, we also implement DQN and Q-Learning-based models.

- *Rewards Models*: This sub-module contains the models associated with the reward functions. According to Section 3.3.2 and Section 3.4, we implemented Load Balancing Model, Response Time Model, and Weighted Cost Model. This sub-module is responsible for calculating the reward values based on the information (e.g., CPU and RAM utilization) and transferring them to the *Agent* sub-module.

- *Reinforcement Learning Agent*: This sub-module implements the functions of the

reinforcement learning agent. The *Agent Initiator* calls the *Reinforcement Learning Models* sub-module and initializes the corresponding models. The *Action Selector* is responsible for outputting the target server index for the currently scheduled task. The *Model Optimizer* optimizes the running reinforcement learning scheduling policy based on the reward values returned from the *Reward Function Models* sub-module. The *State Converter* is responsible for converting the parameters of the server and IoT application into state vectors that can be recognized by the reinforcement learning scheduling model. The *Scheduling Policy Runner* is the running program of the reinforcement learning scheduling *Agent* and is responsible for receiving submitted tasks, saving or loading the trained policies, and requesting and accessing parameters from other FogBus2 components (e.g., *FogBus2 Actor*, *FogBus2 User*) for the computation of reward functions.

- *Model Warehouse*: This sub-module can save the hyperparameters of the trained scheduling policy to the database and loads the hyperparameters to initialize a well-trained scheduling *Agent*.

Algorithm 3.2 summarizes the scheduling mechanism based on DRLIS. The framework first initializes a scheduler, based on Algorithm 3.1. In addition, two buffers $\mathcal{D}_A$ and $\mathcal{D}_U$ for storing information from the *Actor* component and the *User* component are also initialized. After the *User* component submits the IoT application to be processed, the *Master* component first checks whether the *Actor* components that have been registered to the framework have the corresponding resources to process the application. If true, the IoT application which contains one or multiple tasks will be scheduled; otherwise, the *Master* component will inform the *User* component that the current application cannot be processed. For each task of an IoT application, the scheduler will place it to the target *Actor* component for execution based on Algorithm 3.1. After that, the *Actor* component sends the relevant information (i.e., CPU utilization, RAM utilization, etc.) to the *Master* component, which is stored in the buffer $\mathcal{D}_A$. The *User* component also sends relevant information (i.e., response time, the result of task execution, etc.) to the *Master* component, which is stored in the buffer $\mathcal{D}_U$. When the *Master* collects sufficient information, it will update the scheduler, where the data in $\mathcal{D}_A$ and $\mathcal{D}_E$ are used to

Algorithm 3.2: Reinforcement learning scheduler in FogBus2 framework based on the proposed weighted cost optimization algorithm

Input : master component M; registered actor component set A; user component U; tasks to be processed T

$Scheduler \leftarrow InitializeScheduer(DRLIS)$
$\mathcal{D}_A \leftarrow InitializeActorBuffer()$
$\mathcal{D}_U \leftarrow InitializeUserBuffer()$
while *True* **do**
  $U.SubmitTasks(T)$
  $AvailableActors \leftarrow M.CheckResources(T)$
  if *AvailableActors is empty* **then**
    $M.Message(U, Fail)$
    break
  end if
  foreach $t_i \in T$ **do**
    $Scheduler.TaskPlacement(t_i, A)$
    $A.Message(M, I_A)$
    $\mathcal{D}_A.Append(I_A)$
    $U.Message(M, I_U)$
    $\mathcal{D}_U.Append(I_U)$
    if *UpdateScheduler is True* **then**
      $Rewards \leftarrow ComputeRewards(\mathcal{D}_A, \mathcal{D}_U)$
      $Scheduler.Update()$
    end if
  end foreach
end while

compute the reward for each step, as discussed in Algorithm 3.1 and Eq. 3.32, 3.33, 3.34.

3.6 Performance Evaluation

In this section, we first describe the experimental setup and sample applications used in the evaluation. Then, we investigate the hyperparameters of DRLIS. Finally, we discuss the performance of DRLIS by comparing it with its counterparts.

3.6.1 Experiment Setup

We first give a short introduction about the experimental environment and describe the IoT applications used in the experiment. Next, the baseline algorithms used to compare with DRLIS are presented.

Experiment Environment

As discussed in Section 3.5.3, we implemented a scheduler based on DRLIS in the FogBus2 framework, and we use this scheduler for evaluation. We consider a heterogeneous experimental environment consisting of IoT devices, resource-limited fog servers, and resource-rich cloud servers. To simulate the heterogeneous multi-cloud computing environment, we used two instances of Nectar Cloud infrastructure (Intel Xeon 2 cores @2.0GHz, 9GB RAM, and Intel Xeon 16 cores @2.0GHz, 64GB RAM) and one instance of AWS Cloud (AMD EPYC 2 cores @2.2GHz, 4GM RAM). In the fog computing environment, to reflect the heterogeneity of the servers, we used a Raspberry Pi 3B (Broadcom BCM2837 4 cores @1.2GHz, 1GB RAM), a MacBook Pro (Apple M1 Pro 8 cores, 16GB RAM), and a Linux virtual machine (Intel Core i5 2 cores @3.1GHz, 4GB RAM). In addition, the IoT devices are configured with 2 cores @3.2GHz and 4GB RAM. Furthermore, we profiled the average bandwidth (i.e., data rate) and latency between servers as follows: the latency between the IoT device and the cloud server is around 15ms, and the bandwidth is around 6MB/s, while the latency between the IoT device and the fog server is around 3ms, and the bandwidth is around 25MB/s. Also, both w_1 and w_2 are

set to 0.5 in Eq. 3.19, meaning that the importance of load balancing and response time are equal.

Sample IoT Applications

We used four IoT applications for evaluating the performance of the scheduler based on DRLIS. All applications implement both real-time and non-real-time features. Real-time means that the application can receive live streams and non-real-time means that the application can receive pre-recorded video files. Specifically, applications follow a sensor-actuator architecture, with each application operating as a single data stream. Sensors (e.g., cameras) capture environmental information and process it into data patterns (e.g., image frames) that will be forwarded to surrogate servers for processing, while actuators receive the processed data and represent the final outcome to the user. In addition, all applications provide a parameter called *application label*, which can be used to set the frame size in the video. These applications are described as follows:

- *Face Detection* [178]: Detects and captures human faces. The human faces in the video are marked by squares. This application is implemented based on OpenCV[3].

- *Color Tracking* [178]: Tracks colors from video. The user can dynamically configure the target colors through the GUI provided by the application. This application is implemented based on OpenCV[3].

- *Face And Eye Detection* [178]: In addition to detecting and capturing human faces, the application also detects and captures human eyes. This application is implemented based on OpenCV[3].

- *Video OCR* [44]: Recognizes and extracts text information from the video and transmits it back to the user. The application will automatically filter out keyframes. This application is implemented based on Googles Tesseract-OCR Engine[4].

[3]https://github.com/opencv/opencv
[4]https://github.com/tesseract-ocr/tesseract

Baseline Algorithms

To evaluate the performance of DRLIS, three other schedulers based on metaheuristic algorithms and reinforcement learning techniques are implemented, as follows:

- *DQN*: It is one of the most adapted techniques in deep reinforcement learning, which constructs an end-to-end architecture from perception to decision. This algorithm has been used by many works in the current literature such as [189], [76], [190], and [191]. To compare with our proposed algorithm, we implement a DQN-based scheduler and integrate it into the FogBus2 framework. This scheduler can minimize the weighted load balancing and response time cost.

- *Q-Learning*: This technique belongs to value-based reinforcement learning techniques that combine the Monte Carlo method and the TD method. Its ultimate goal is to learn a table (*Q-Table*). Works including [188], [213] adopt this technique. To integrate it into the FogBus2 framework, we implemented a scheduling policy. Furthermore, as a comparison, the scheduler can be used in the weighted cost problem to minimize the weighted load balancing and response time cost.

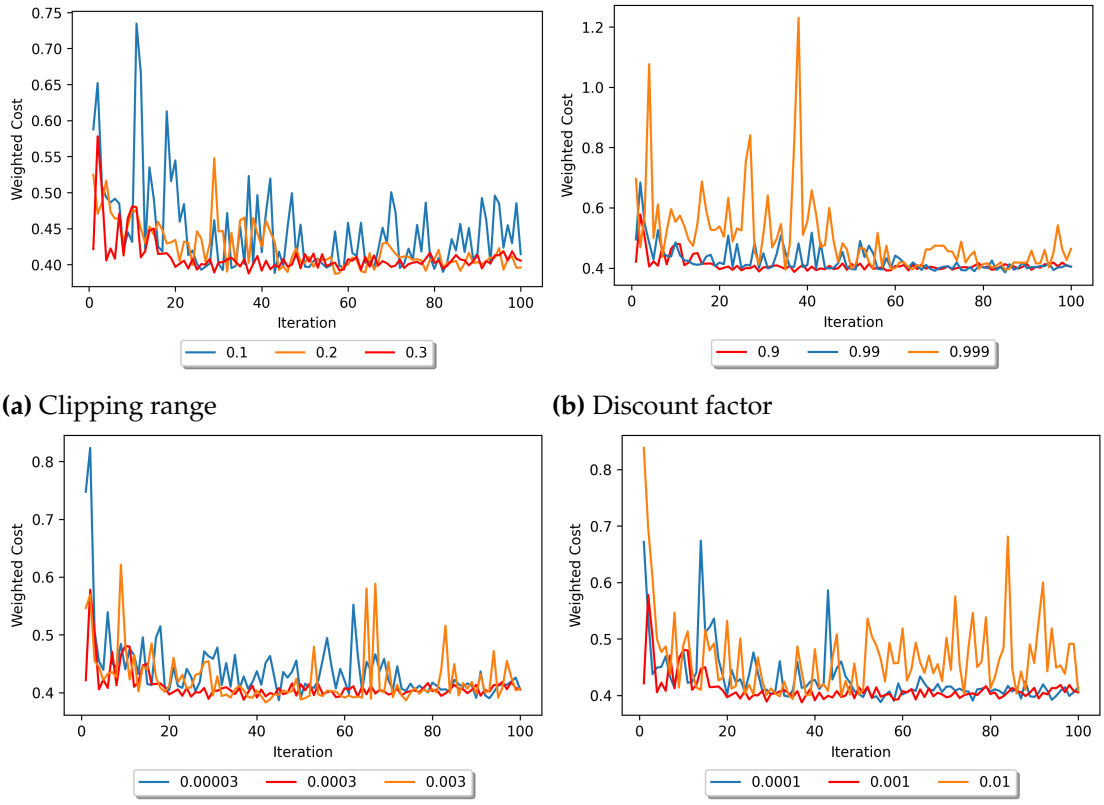
- *NSGA2*: It is a weighted cost genetic algorithm. It adopts the strategy of fast non-dominated sorting and crowding distance to reduce the complexity of the non-dominated sorting genetic algorithm. The algorithm has high efficiency and fast convergence rate [214]. This algorithm is implemented using Pymoo [215].

- *NSGA3*: The framework of NSGA3 is basically the same as NSGA2, using fast non-dominated sorting to classify population individuals into different non-dominated fronts, and the difference mainly lies in the change of selection mechanism. Compared with NSGA2 using crowding distance to select individuals of the same non-dominated level, NSGA3 introduces well-distributed reference points to maintain population diversity under high-dimensional goals [216]. This algorithm is implemented using Pymoo [215].

3.6.2 Hyperparameter Tuning

The scheduler based on DRLIS is implemented via PyTorch. Considering the limited computational resources of some devices in the fog computing environment, both actor network and critic network consist of an input layer, a hidden layer, and an output layer. Henderson et al. [217] investigate the effect of hyperparameter settings on the performance of reinforcement learning models. They survey the literature on different reinforcement learning techniques, list the hyperparameter settings used in the literature, and compare the actual performance of the models under different hyperparameter settings. They compare the performance of the PPO algorithm under different network architectures and the result shows that the model performs best under the network architecture where the hidden layer contains 64 hidden units and the hyperbolic tangent (TanH) function is used as the activation function. Therefore, we used the same network architecture for our experiments. In addition, we performed a grid search to tune the four main hyperparameters (i.e., clipping range, discount factor, learning rate for actor network, and learning rate for critic network), and the results are shown in Fig. 3.5. The load balancing model control parameters $a1$ and $a2$ are both set to 0.5 to show the equal importance of CPU and RAM, however, these values can be tuned by users based on the objectives.

All the experiments regarding hyperparameter tuning are conducted in order to solve the weighted cost problem, as discussed in Section 3.3.2. We describe the process of hyperparameter tuning of our reinforcement learning model. For tuning the clipping range ϵ, we followed Schulman et al. [40], who proposed PPO and described that the model performs best with settings of clipping range ϵ among 0.1, 0.2, and 0.3. Fig. 3.5a shows that our model performs best when the clipping range ϵ is set to 0.3. For the discount factor γ, we reviewed related work on DRL in order to understand the common range for γ. According to [40, 218], the best setting for γ sits somewhere among $\{0.9\text{-}0.999\}$. Accordingly, to keep the search area for tuning γ in a viable range, we used the nominated values in these works and found that our model converges faster when γ is set to 0.9. Fig. 3.5b shows the tuning process of γ. Based on the similar approach for tuning ϵ and γ, for tuning the actor network learning rate lr_a, we referred to [40, 217, 219] for designing our tuning range. Accordingly, we used 0.003, 0.0003, and



(a) Clipping range **(b)** Discount factor

(c) Actor network learning rate **(d)** Critic network learning rate

Figure 3.5: Hyperparameter tuning results

0.00003 to tune lr_a. Fig. 3.5c shows that our model performs best when the lr_a is set to 0.0003. Considering the same approach for tuning, we followed [220–222] and set our tuning range among {0.01, 0.001, 0.0001} and found that our model works best when lr_c is 0.001. Fig. 3.5d shows the performance of our model under different settings for lr_c. Overall, the deep neural network and training hyperparameters setting is presented in Table 3.3. Besides, we also tune the hyperparameters for baseline techniques to fairly study their performance. The corresponding results are shown in Table 3.5.

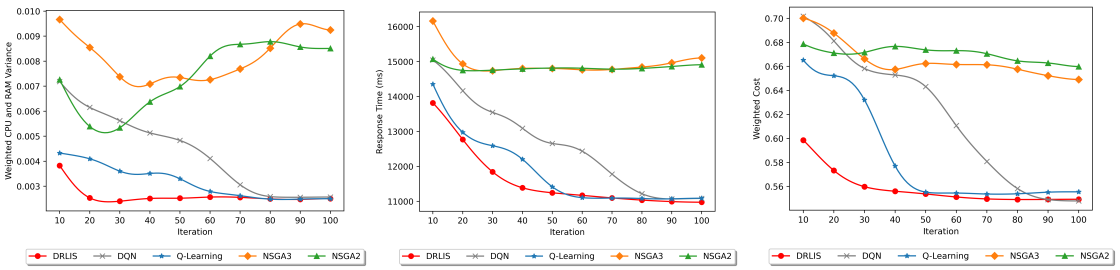
Table 3.3: The hyperparameters setting for DRLIS

DRLIS Hyperparameter	**Value**
Neural Network Layers	3
Hidden Layer Units	64
Optimization Method	Adam
Activation Function	TanH
Clipping Range ϵ	0.3
Discount Factor γ	0.9
Actor Learning Rate lr_a	0.0003
Critic Learning Rate lr_c	0.001
Policy Objective Function Coefficient a_c	1
Value Function Loss Function Coefficient a_v	0.5
Entropy Bonus Coefficient a_e	0.01
Load Balancing Model CPU Control Parameter a_1	0.5
Load Balancing Model RAM Control Parameter a_2	0.5

Table 3.5: The hyperparameters setting for baseline techniques

DQN Hyperparameter	**Value**
Neural Network Layers	3
Hidden Layer Units	64
Optimization Method	Adam
Activation Function	ReLU
Discount Factor	0.99
Learning Rate	0.0001
Exploration Rate	1
Exploration Decay	0.9
Minimum Exploration	0.05
Q-Learning Hyperparameter	**Value**
Discount Factor	0.9
Learning Rate	0.1
NSGA2 and NSGA3 Hyperparameter	**Value**
Population Size	200
Generation Numbers	100

is set to 100. The training results of algorithms with the three optimization objectives are shown in Fig. 3.6.



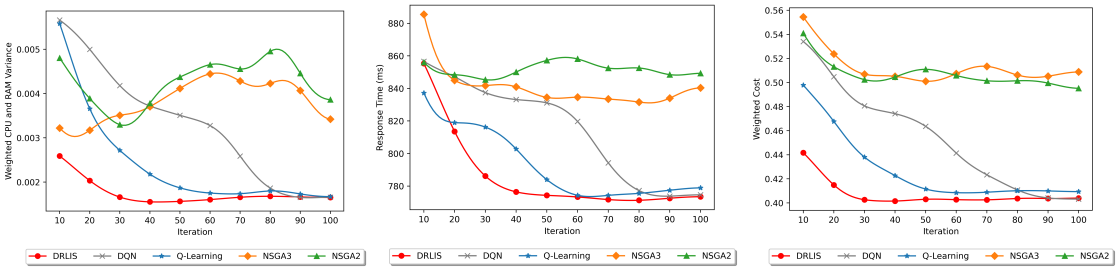
(a) Load balancing **(b)** Response time **(c)** Weighted cost

Figure 3.6: Cost vs policy update analysis - train phase

As shown in Fig. 3.6a, when optimizing the load-balancing problem of the servers, the average computational resource variance of the servers is lower for the Q-Learning-based, DQN-based, and DRLIS-based schedulers than for the NSGA2-based and NSGA3-based schedulers. Moreover, only the reinforcement learning-based scheduler can achieve a stable convergence state. However, the Q-Learning-based scheduler requires more

than 60 iterations before reaching a converged state, and the DQN-based scheduler requires more than 80 iterations, while the DRLIS-based scheduler only requires about 20 updates to converge to a similar stable state. The NSGA2-based and NSGA3-based schedulers are unable to reach the convergence state. As shown in Fig. 3.6b, when optimizing the response time problem of the application, unlike the former problem, all schedulers can converge. However, the average response time of Q-Learning-based, DQN-based, and DRLIS-based schedulers is still lower than that of NSGA2-based and NSGA3-based schedulers. In addition, the DRLIS-based scheduler still outperforms the Q-Learning-based and the DQN-based schedulers in terms of convergence speed. Finally, as Fig. 3.6c shows, when optimizing the weighted cost problem, similar to the load balancing problem, the average cost is lower for the Q-Learning-based, DQN-based, and DRLIS-based schedulers than for the NSGA2 and NSGA3-based schedulers, and only the first three can reach a stable convergence state. Moreover, although the Q-Learning-based, DQN-based, and DRLIS-based schedulers have similar final convergence levels, the DRLIS-based scheduler converges much faster than the Q-Learning-based and DQN-based schedulers. This proves that the DRLIS-based scheduler outperforms the other techniques in terms of average cost, convergence, and convergence speed during the training phase.

In the evaluation phase, we set the resolution to 240, which will make the demand for computational resources and response time of the IoT application different from the training phase. The evaluation phase results of the different algorithms regarding the three optimization objectives are shown in Fig. 3.7. It can be observed that when the optimization objective is server load balancing, IoT application response time, and weighted cost, respectively, the schedulers based on different algorithms have similar performances as the training phase. Specifically, only the cost of the Q-Learning-based, DQN-based, and DRLIS-based schedulers converges, and the cost of the NSGA2-based and NSGA3-based schedulers fluctuates up and down in a higher range. Moreover, the average and final costs of the Q-Learning-based, DQN-based, and DRLIS-based schedulers are significantly lower than those of the NSGA2-based and NSGA3-based schedulers during the evaluation phase. In addition, in the weighted cost scenario, the DRLIS-based scheduler can converge the cost to a stable level after about 30 policy updates,



(a) Load balancing **(b)** Response time **(c)** Weighted cost

Figure 3.7: Cost vs policy update analysis - evaluation phase

while the Q-Learning-based scheduler usually takes about 60 updates to converge to a slightly higher level, and the DQN-based scheduler needs more than 80 updates to converge to the same level. Overall, compared with the Q-Learning-based scheduler, which can converge stably and with the fastest convergence speed in the baseline algorithms, the average performance of the DRLIS-based scheduler improves by 55%, 37%, and 50%, in terms of servers load balancing, IoT application response time, and weighted cost, respectively.

Scheduling Overhead Analysis

In this section, we investigate the scheduling overhead of different techniques-based schedulers when handling IoT applications. The environment settings are the same as Section 3.6.1, and the resolution of the IoT applications is set to 480. For each scheduler, we repeat the experiment for 100 rounds, feeding four IoT applications to the scheduler in each round. Besides, we define the average scheduling overhead as $T_{ave} = \frac{T_{total}}{100}$, where T_{total} represents the total overhead spent by the scheduler to handle the applications in 100 rounds.

Figure 3.8 depicts the average scheduling overhead T_{ave} with a 95% Confidence Interval (CNFI) of schedulers based on different technologies when handling IoT applications. It is obvious that the scheduling overheads of reinforcement learning techniques (i.e., DRLIS, DQN, Q-Learning) are usually lower than metaheuristics techniques (i.e., NSGA2, NSGA3). In addition, the 95% CNFI of the scheduling overhead of reinforcement learning techniques is also much shorter than metaheuristic techniques. Specif-

ically, the scheduling overhead of DRLIS is more than 50% lower than NSGA2 and NSGA3, and more than 33% lower than DQN, but it is about 2ms more than Q-Learning. However, considering that the convergence speed of DRLIS is much faster than that of Q-Learning, as discussed in Section 3.6.3, the increased overhead cost of DRLIS over Q-Learning can be negligible. Therefore, in the heterogeneous edge and fog computing environment, our proposed DRLIS-based algorithm can handle the weighted cost optimization problem of IoT applications more efficiently than other techniques.

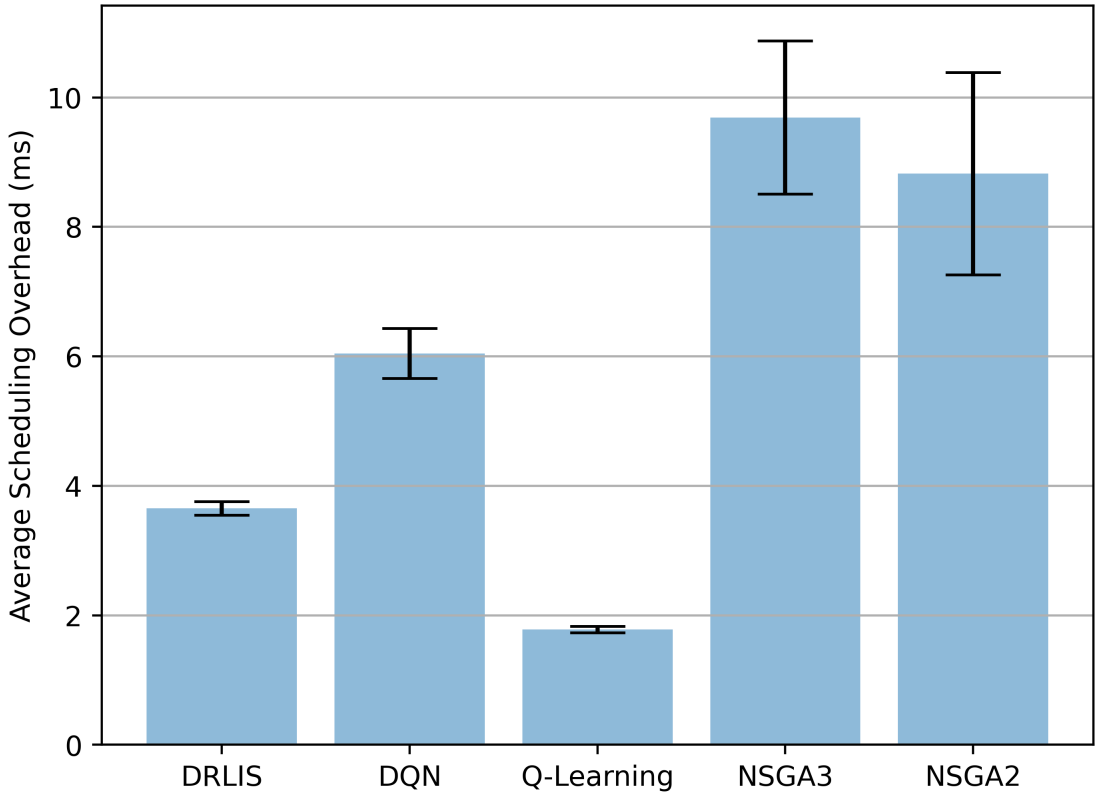


Figure 3.8: Average scheduling overhead with a 95% CNFI

3.7 Summary

In this chapter, we proposed DRLIS, a DRL-based algorithm to solve the weighted cost optimization problem for IoT applications scheduling in heterogeneous edge and fog

computing environments. First, we proposed corresponding cost models for optimizing load balancing and response time in heterogeneous edge and fog computing environments and formulate a weighted cost model based on both of them. In addition, we implemented a practical scheduler in the FogBus2 function-as-a-service framework for scheduling IoT applications. Compared to existing work, DRLIS has significant advantages in convergence speed, optimization cost, and scheduling overhead. Through extensive experiments and comparisons with other works in the literature, DRLIS achieves performance improvements of up to 49%, 60%, and 55% in terms of load balancing, response time, and weighted cost, respectively.

Chapter 4

TF-DDRL: A Transformer-enhanced Distributed DRL Technique for Scheduling IoT Applications in Edge and Cloud Computing Environments

With the continuous increase of IoT applications, their effective scheduling in edge and cloud computing has become a critical challenge. The inherent dynamism and stochastic characteristics of edge and cloud computing, along with IoT applications, necessitate solutions that are highly adaptive. Currently, several centralized Deep Reinforcement Learning (DRL) techniques are adapted to address the scheduling problem. However, they require a large amount of experience and training time to reach a suitable solution. Moreover, many IoT applications contain multiple interdependent tasks, imposing additional constraints on the scheduling problem. To overcome these challenges, we propose a Transformer-enhanced Distributed DRL scheduling technique, called TF-DDRL, to adaptively schedule heterogeneous IoT applications. This technique follows the Actor-Critic architecture, scales efficiently to multiple distributed servers, and employs an off-policy correction method to stabilize the training process. In addition, Prioritized Experience Replay (PER) and Transformer techniques are introduced to reduce exploration costs and capture long-term dependencies for faster convergence. Extensive results of practical experiments show that TF-DDRL, compared to its counterparts, significantly reduces response time, energy consumption, monetary cost, and weighted cost by up to 60%, 51%, 56%, and 58%, respectively.

This chapter is derived from:

- **Zhiyu Wang**, Mohammad Goudarzi, and Rajkumar Buyya, "TF-DDRL: A Transformer-Enhanced Distributed DRL Technique for Scheduling IoT Applications in Edge and Cloud Computing Environments", *IEEE Transactions on Services Computing (TSC)*, Volume 18, Number 2, Pages: 1039-1053, IEEE, 2025.

4.1 Introduction

The Internet of Things (IoT) paradigm is rapidly emerging as a transformative force and revolutionizing information technology and connectivity. The proliferation of IoT devices and applications has been exponential, reshaping the way humans interact and perceive their surroundings. Cloud computing, as a major driver of the IoT ecosystem, plays a critical role in the storage and process of the large volume of data generated by IoT devices [9]. However, due to the potentially long physical distance between servers in cloud computing and IoT devices, high latency arises, thereby impeding the effective implementation of real-time IoT applications [223]. In response to these challenges, edge computing, as a decentralized computing paradigm, has emerged to provide the ability to process, store, and intelligently control IoT applications [9]. It has quickly become a popular computing paradigm in the IoT environment, offering substantial solutions in various domains. For instance, in the healthcare sector, edge computing can enable real-time monitoring and diagnosis, facilitating faster and more accurate medical decisions [224]; In smart cities, edge computing can be applied to real-time traffic management, improving traffic efficiency, and reducing congestion [225].

However, the considerable increase of IoT applications and servers within edge and cloud computing environments has brought new challenges, necessitating innovative solutions. First, there is an urgent need to minimize the expected response time of IoT applications to ensure efficient and timely performance [26, 226]. Furthermore, in edge and cloud computing environments, the imperative need to minimize server energy consumption and monetary cost is equally crucial for sustainable and cost-effective operations [199]. Thus, scheduling IoT applications on distributed servers to reduce the response time of IoT applications while simultaneously minimizing server energy consumption and monetary cost has become an important and challenging problem.

Given the inherent complexity of this challenge, which can be characterized as an NP-hard problem [227], various solutions have been explored, including heuristic and rule-based approaches [9]. However, these methods face limitations when dealing with the dynamic and unpredictable characteristics of servers in edge and cloud computing. The performance, utilization, and downtime of servers often lack regularity, and the

number of IoT applications and their corresponding resource requirements may exhibit randomness. Additionally, IoT applications typically employ Directed Acyclic Graphs (DAGs) for modeling, where nodes represent tasks and edges signify data communication between related tasks [11]. The dependencies between tasks introduce further complexity to the application scheduling process, rendering heuristic and rule-based solutions ineffective in addressing the scheduling challenges presented by IoT computing environments.

Due to the continuous changes in edge and cloud computing environments, decision-making for scheduling IoT applications must be capable of adaptive updates. Deep Reinforcement Learning (DRL), which combines Reinforcement Learning (RL) with Deep Neural Networks (DNN), offers a promising solution. DRL agents can dynamically learn optimal policies and long-term rewards in a stochastic environment without the need for a prior understanding of the system. However, DRL agents must invest substantial time during the exploration phase by collecting extensive and diverse experience trajectories, which are later used to learn optimal policies [228]. Hence, the effectiveness of the DRL technique can be prevented by the high exploration costs and slow convergence speeds, negatively impacting the scheduling of IoT applications in highly heterogeneous and stochastic edge and cloud computing environments.

Existing works have increasingly employed DRL techniques for scheduling IoT applications in edge and cloud environments. However, these works face several significant challenges and limitations in practical deployments. Firstly, most existing studies utilize centralized DRL techniques (e.g., Deep Q-Network (DQN), Deep Deterministic Policy Gradient (DDPG)), which require extensive exploration within complex state spaces to identify optimal scheduling strategies [19]. This can lead to high exploration costs and slow convergence rates, particularly in dynamic and heterogeneous edge and cloud environments where exploration costs are further exacerbated. While some works have explored distributed DRL techniques, they predominantly use Asynchronous Advantage Actor Critic (A3C). Although these works leverage multiple parallel agents to collect experience trajectories, the training process primarily relies on asynchronous updates based on the local experiences of individual agents [88]. This limited global knowledge sharing between agents can result in less efficient convergence and suboptimal

overall performance [93]. Secondly, existing works struggle to capture long-term dependencies between tasks, especially in the context of scheduling IoT applications that involve complex task dependency graphs (i.e., DAGs). This limitation often leads to suboptimal resource utilization and increased task completion times in real-world IoT systems. Thirdly, many existing cost models focus on optimizing only one or two objectives, such as minimizing response time or energy consumption. This narrow focus restricts their effectiveness in addressing the multifaceted challenges of dynamic and heterogeneous computing environments. These challenges highlight the need for a comprehensive solution that can effectively address multiple optimization objectives while efficiently handling task dependencies and leveraging distributed learning in heterogeneous edge and cloud environments.

To address these challenges, we propose a distributed DRL technique, named TF-DDRL, based on the Importance Weighted Actor-Learner Architecture (IMPALA) [93], specifically designed for scheduling IoT applications in edge and cloud computing environments. Unlike A3C-based approaches that rely on parameter sharing, IMPALA facilitates the direct sharing of raw experiences between distributed agents, allowing them to collaboratively learn policies more effectively. This enables TF-DDRL to adaptively optimize scheduling decisions across multiple servers, efficiently handling the dynamic and heterogeneous nature of edge and cloud environments. To achieve comprehensive optimization, we propose a weighted cost model that balances multiple objectives, including response time, energy consumption, and monetary costs. This model addresses the limitations of existing models that typically focus on single objectives, enabling more holistic optimization that better reflects real-world requirements. To further tackle the challenge of capturing complex dependencies between IoT tasks and system states in heterogeneous edge and cloud environments, we integrate the Transformer technique [229]. The self-attention mechanism in the Transformer captures both local and global relationships among diverse state components [42], which is essential for making optimal scheduling decisions in highly dynamic and stochastic environments. Additionally, we incorporate Prioritized Experience Replay (PER) [230] to reduce exploration costs by prioritizing more informative experiences, thereby expediting the learning process. The combined use of Transformer and PER not only accelerates the convergence speed of

TF-DDRL but also enhances its capacity to derive more effective scheduling strategies, ultimately optimizing response time, energy consumption, and operational costs.

To the best of our knowledge, this is the first work that integrates distributed DRL with Transformer and PER techniques for IoT application scheduling in edge and cloud environments. The main contributions of this chapter are as follows.

- We propose a weighted cost model for scheduling DAG-based IoT applications in edge and cloud computing. The objective is to optimize the response time of the application, the energy consumption of the system, and the monetary cost associated with execution. Also, we customize this weighted cost model to comply with DRL algorithms.

- We propose a distributed DRL technique, called TF-DDRL, to solve the weighted cost optimization problem. It can adaptively learn the optimal scheduling policy in response to changes in the computing environment, including diverse IoT application requests and fluctuations of computing resources.

- We design the network structure of TF-DDRL, integrating advanced techniques including PER and the Transformer. This design can significantly improve the convergence speed of the TF-DDRL, ensuring more efficient and effective model performance.

- To evaluate the performance of TF-DDRL, we carry out extensive practical experiments and employ real IoT applications. Through comparisons with distributed DRL techniques including ApeX-Deep Q-Network (ApeX-DQN) [92] and A3C [88], as well as centralized DRL techniques including Dueling Double DQN-RNN (D3QN-RNN) [73, 74] and Soft Actor-Critic (SAC) [84], we highlight the superior performance of TF-DDRL in terms of convergence speed, optimization cost, scalability, and scheduling overhead.

The remainder of the chapter is organized as follows. The related literature is provided in Section 4.2, and Section 4.3 details the system model and formulates the scheduling problem. The main concepts of the DRL model are presented in Section 4.3.3. The

TF-DDRL is discussed in Section 4.4. Section 4.5 evaluates the performance of TF-DDRL and its counterparts. Finally, the conclusion is provided in Section 4.6.

4.2 Related Work

The related works that research IoT application scheduling problems in edge and cloud computing environments are studied. Related work is categorized into two groups: centralized reinforcement learning and distributed reinforcement learning.

4.2.1 Centralized Reinforcement Learning

In the category of centralized reinforcement learning, Bansal et al. [231] proposed a Dueling-DQN-based technique to place IoT applications in edge and cloud environments. It aims at optimizing the user-side latency and system energy. Hoang et al. [232] proposed an online resource management framework based on the Actor-Critic framework, which considers the long-term constraints of queue stability and computational delay of the queuing system to minimize the average power consumption of the entire system. Huang et al. [191] focused on the resource allocation problem in edge computing environments. They developed a DQN-based approach to minimize a weighted cost, comprising total energy consumption and task completion delay. Zhao et al. [233] proposed a mobile-aware dependent task offloading scheme based on DDPG, with the aim of minimizing the average response time and the average energy consumption of the system. Hsieh et al. [234] investigated the task allocation problem in collaborative Mobile Edge Computing (MEC) networks, developing and comparing the performance of Double-DQN, PG, and Actor-Critic in optimizing delay and task overflow rate. The results demonstrated that the Actor-Critic approach performed the best in dynamic MEC network environments. Fan et al. [140] studied the problem of user task offloading in MEC network environments and proposed a technique based on Dueling-DQN and Double-DQN to optimize response time and dropped task ratio. Zheng et al. [193] defined an optimization problem involving computational offloading and resource allocation in collaborative vehicle networks. A Soft Actor-Critic (SAC)-

based technique is proposed to reduce the overall delay of the system. Wang et al. [190] proposed a computing resource allocation solution based on DQN specifically for edge computing environments. The objective of their research is to optimize the average time overhead and achieve a more balanced utilization of resources within edge environments. Xiong et al. [76] aimed at reducing the average job completion time within edge computing environments by employing a DQN-based resource allocation policy. Jie et al. [189] focused on the time overhead optimization problem in edge computing environments and employed a DQN-based method to reduce the task execution time. They formulate the optimization problem as a Markov Decision Process (MDP). [235] proposed a dependency-aware task offloading method with DQN to optimize task offloading in cloud-edge environments. Their approach models mobile applications as DAGs and leverages DQN to adaptively handle dynamic resource changes and parallel task scheduling without presetting task priorities.

4.2.2 Distributed Reinforcement Learning

Wen et al. [236] introduced an adaptive scheduler based on environmental changes, aiming to reduce the tail latency of edge-cloud jobs. The work employs Ape-X DQN to expedite the training process. Wang et al. [237] proposed an Asynchronous Advantage Actor-Critic (A3C)-based approach to address the cloud-edge computing network optimization problem, aiming at satisfying the latency requirements of applications while reducing the cost of cloud servers. Garaali et al. [238] investigated the optimization problem of computational offloading and resource allocation in an MEC environment and proposed a solution based on the A3C method. In order to reduce the system latency, each agent aims to learn the optimal offloading policy independently of the other agents in an interactive manner. Ju et al. [239] considered the task offloading problem in vehicular edge computing networks, where the joint optimization is formulated as MDP. This work proposes an A3C-based approach to solve the MDP problem, with the goal of minimizing the system energy consumption while satisfying computational delay constraints. Sellami et al. [240] investigated IoT application scheduling and offloading problems in edge computing environments. This work introduces a scheduling policy

based on A3C to enhance energy efficiency. Chen et al. [241] studied the task offloading problem in cloud-edge collaborative mobile computing environments, proposing an A3C-based algorithm to address the joint optimization problem involving task execution delay and energy consumption. Utilizing a distributed learning approach, the algorithm acquires knowledge about the probability distribution of an approximate reward and optimizes network parameters using the computing resource in the cloud, with the objective of achieving faster and more efficient decision-making.

4.2.3 A Qualitative Comparison

Table 4.1 provides a qualitative analysis of current research work and ours in various dimensions, including application properties, architectural properties, algorithm properties, and evaluation. Application properties explore whether the IoT application has multiple tasks or not and their interdependencies. Architectural properties are divided into three layers. The IoT device layer identifies the practical characteristics of the application and the request type of the IoT devices. The section named real applications provides information on whether the work utilizes real-world IoT applications, simulations, or randomly generated data, and distinct IoT devices with varying quantities of requests and diverse requirements are classified as heterogeneous request types. The edge/cloud layer delves into the computing environment considered by the work and the heterogeneity of deployment servers. Moreover, the multi-cloud layer assesses whether the work takes into account cloud computing resources from different providers. In the algorithm properties section, the focus is on the primary techniques employed by the work and the optimization objectives. Finally, the evaluation section determines whether the work is evaluated through simulation or practical applications.

In the literature, many works (e.g., [232], [191], [234], [193], [190], [76], [189], [237], [238], [239], and [240]) assume that tasks are mutually independent and do not consider the common occurrence of task dependencies in the real world. However, in practical scenarios, such dependencies significantly impact scheduling performance. For example, in smart city traffic management, vehicle detection relies on real-time video data collection, while traffic flow prediction depends on the results of vehicle detection. Ne-

glecting these dependencies can lead to processing delays and reduced system efficiency. Similarly, in healthcare, initial patient data processing must be completed before cloud-based analysis, or else diagnoses could be delayed. In industrial IoT, equipment fault detection is closely linked to subsequent maintenance tasks, where improper scheduling could result in production disruptions. Also, works including [231], [232], [191], [233], [234], [140], [193], [190], [76], [189], and [235], adopt centralized DRL techniques, which may incur high exploration costs and exhibit low convergence speed [242]. This poses challenges when deployed in highly distributed computing environments, especially as the number of features, environmental complexity, and application constraints increase. Furthermore, most of the work employing distributed DRL techniques is based on A3C, including [237], [238], [239], [240], and [241]. Despite deploying distributed agents to collect experience trajectories, distributed agents in A3C train their local policies based on their limited experiences, subsequently forwarding these parameters to learners for aggregation and training, diminishing the usage efficiency of experience trajectories. To address these issues, we propose a distributed DRL technique, called TF-DDRL, which learns policies based on direct sharing of original experience, rather than parameters. Besides, TF-DDRL employs PER to enhance sampling efficiency and incorporates the Transformer to capture long-term dependencies between features, further improving convergence speed and optimizing performance. Moreover, our work considers inter-task dependencies when addressing IoT application scheduling problems in edge and multi-cloud heterogeneous environments. Also, we establish a practical experimental environment employing both real-time and non-real-time IoT applications to evaluate the performance of TF-DDRL.

4.3 System Model and Problem Formulation

This section first describes the topology of IoT systems in this work. Next, we tackle the scheduling of IoT applications by formulating it as an optimization problem, aiming at reducing application response time, system energy consumption, and monetary cost of running applications. Table 4.2 depicts the key notations used in this chapter.

Table 4.1: A comparison of our work with existing related works

Work	Application Properties		Architectural Properties					Algorithm Properties						Evaluation
	Task Number	Dependency	IoT Device Layer		Edge/Cloud Layer		Multi-Cloud Layer	Main Technique		Optimization Objectives				
			Real Applications	Request Type	Computing Environment	Heterogeneity				Time	Energy	Finance	Multi Objective	
[231]	Multiple	Dependent	◐	Heterogeneous	Edge and Cloud	Heterogeneous	×	Centralized	Dueling-DQN	✓	✓	×	✓	Simulation
[232]	Single	Independent	○	Homogeneous	Edge	Heterogeneous	×		Actor-Critic	×	✓	×	×	Simulation
[191]	Multiple	Independent	◐	Heterogeneous	Edge	Homogeneous	×		DQN	✓	✓	×	✓	Simulation
[233]	Multiple	Dependent	◐	Heterogeneous	Edge	Heterogeneous	×		DDPG	✓	✓	×	✓	Simulation
[234]	Single	Independent	◐	Heterogeneous	Edge and Cloud	Heterogeneous	×		Double-DQN, PG, and Actor-Critic	✓	×	×	×	Simulation
[140]	Multiple	Dependent	○	Homogeneous	Edge	Homogeneous	×		Dueling-DQN and Double-DQN	✓	×	×	✓	Simulation
[193]	Single	Independent	◐	Homogeneous	Edge	Homogeneous	×		SAC	✓	×	×	×	Simulation
[190]	Single	Independent	◐	Homogeneous	Edge	Homogeneous	×		DQN	✓	×	×	✓	Simulation
[76]	Multiple	Independent	◐	Homogeneous	Edge	Homogeneous	×		DQN	✓	×	×	×	Simulation
[189]	Single	Independent	◐	Homogeneous	Edge	Homogeneous	×		DQN	✓	×	×	×	Simulation
[235]	Multiple	Dependent	◐	Heterogeneous	Edge and Cloud	Heterogeneous	×		DQN	✓	×	×	×	Simulation
[236]	Multiple	Dependent	◐	Heterogeneous	Edge and Cloud	Heterogeneous	×	Distributed	Ape-X DQN	✓	×	×	×	Simulation
[237]	Single	Independent	○	Homogeneous	Edge and Cloud	Homogeneous	×		A3C	✓	×	✓	✓	Simulation
[238]	Multiple	Independent	○	Homogeneous	Edge	Homogeneous	×		A3C	✓	×	×	×	Simulation
[239]	Single	Independent	◐	Heterogeneous	Edge	Heterogeneous	×		A3C	×	✓	×	×	Simulation
[240]	Single	Independent	◐	Heterogeneous	Edge	Heterogeneous	×		A3C	×	✓	×	×	Simulation
[241]	Multiple	Dependent	○	Homogeneous	Edge and Cloud	Heterogeneous	×		A3C	✓	✓	×	✓	Simulation
Our work	Multiple	Dependent	●	Heterogeneous	Edge and Cloud	Heterogeneous	✓		IMPALA	✓	✓	✓	✓	Practical

●: Real IoT Application and Deployment, ◐: Simulated IoT Application, ○: Random

Table 4.2: List of key notations

Variable	Description	Variable	Description
$\mathcal{A}$	One application set	$PR(\mathcal{A}_i^j)$	The set of predecessor tasks of task $\mathcal{A}_i^j$
$\mathcal{A}_i$	One application	$SU(\mathcal{A}_i^j)$	The set of successor tasks of task $\mathcal{A}_i^j$
$\mathcal{A}_i^j$	One single task	T	The response time model
N	The server set	T^{dat}	The Data Arrival Time (DAT) model
$\mathcal{CS}$	The cloud server set	T^{ex}	The execution time model
$\mathcal{ES}$	The edge server set	T^{tr}	The transmission time model
N_k	One single server	E	The energy consumption model
$Freq(\mathcal{N}_k)$	The available CPU frequency (MHz) of server $\mathcal{N}_k$	E^{ex}	The execution energy model
$Ram(\mathcal{N}_k)$	The available RAM size (GB) of server $\mathcal{N}_k$	E^{tr}	The transmission energy model
$\mathcal{C}_i$	The scheduling configuration for application $\mathcal{A}_i$	$W^{ex}(\mathcal{N}_k)$	The power of the server $\mathcal{N}_k$ when executing task
$\mathcal{C}_i^j$	The scheduling configuration for task $\mathcal{A}_i^j$	$W^{tr}(\mathcal{N}_k)$	The power of the server $\mathcal{N}_k$ when transmitting data
$\mathcal{P}_{\mathcal{N}_l,\mathcal{N}_k}$	The propagation time (ms) between server $\mathcal{N}_l$ and server $\mathcal{N}_k$	F	The monetary cost model
$DS_{\mathcal{N}_l,\mathcal{N}_k}(\mathcal{A}_i^j)$	The data size for task $\mathcal{A}_i^j$ sent from server $\mathcal{N}_l$ to server $\mathcal{N}_k$	$CLP(\mathcal{N}_k)$	The pricing of the cloud server $\mathcal{N}_k$
$\mathcal{B}_{\mathcal{N}_l,\mathcal{N}_k}$	The data rate (bits/second) between server $\mathcal{N}_l$ and server $\mathcal{N}_k$	$EP(\mathcal{N}_k)$	The electricity price for running edge server $\mathcal{N}_k$
$L(\mathcal{A}_i^j)$	The required CPU cycles for task $\mathcal{A}_i^j$	J	The weighted cost model

4.3.1 System Model

Fig. 4.1 provides a layered perspective of the IoT system in an edge and cloud environment. Consider $\mathcal{A} = \{\mathcal{A}_i | 1 \leq i \leq |\mathcal{A}|\}$ as a collection of $|\mathcal{A}|$ applications, with each application comprising one or more tasks denoted as $\mathcal{A}_i = \{\mathcal{A}_i^j | 1 \leq j \leq |\mathcal{A}_i|\}$. To model an IoT application, we use a DAG, as illustrated in Fig. 4.1, where each vertex $\mathcal{V}_j = \mathcal{A}_i^j$ corresponds to a specific task within the application $\mathcal{A}_i$. The edges, represented as $\mathcal{E}_{j,k}$, signify the data flow between tasks $\mathcal{V}_j$ and $\mathcal{V}_k$, indicating that successor tasks must follow the completion of their predecessors. Also, the critical path of the DAG, denoted as $CP(\mathcal{A}_i)$ and marked in red in the figure, shows the path with the highest cost.

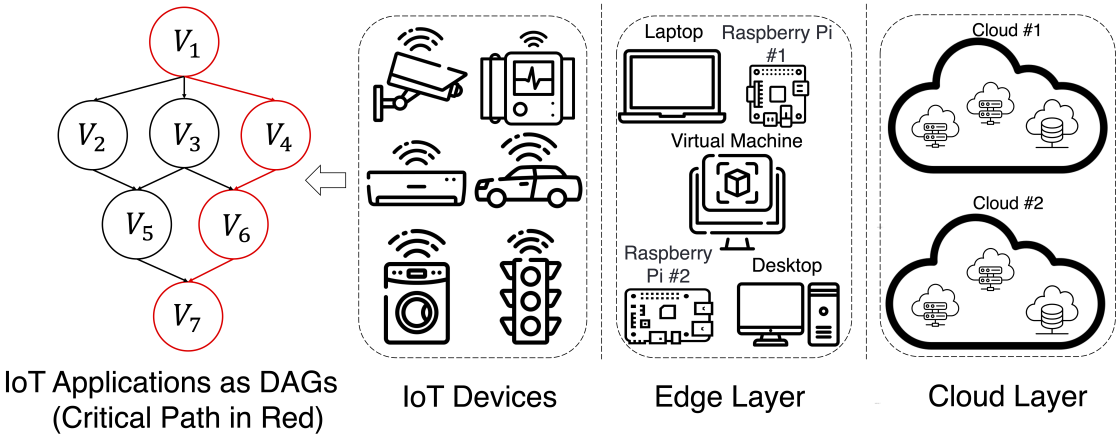


Figure 4.1: An overview of Edge and Cloud computing

We consider a server set comprising $|\mathcal{N}|$ servers to handle the application set $\mathcal{A}$, denoted as $\mathcal{N} = \mathcal{CS} \cup \mathcal{ES} = \{\mathcal{N}_k | 1 \leq k \leq |\mathcal{N}|\}$. $\mathcal{CS}$ denotes the cloud server set and $\mathcal{ES}$ denotes the edge server set. To consider server heterogeneity, each server $\mathcal{N}_k$ is characterized by different available CPU frequency (MHz) $Freq(\mathcal{N}_k)$ and available RAM size (GB) $Ram(\mathcal{N}_k)$. Also, we consider $\mathcal{P}_{\mathcal{N}_l,\mathcal{N}_k}$ as the propagation time (ms) and $\mathcal{B}_{\mathcal{N}_l,\mathcal{N}_k}$ as the data rate (b/s) between server $\mathcal{N}_l$ and $\mathcal{N}_k$.

4.3.2 Problem Formulation

Since an application consists of one or more tasks, it can run on various servers. Considering server set as $\mathcal{N}$, the scheduling configuration $\mathcal{C}_i^j$ for task $\mathcal{A}_i^j$ is defined as:

$$\mathcal{C}_i^j = \mathcal{N}_k, \quad k \in \{1, \ldots, |\mathcal{N}|\}, \tag{4.1}$$

where N_k denotes a particular server, and $|N|$ is the total number of servers. This variable serves as the atomic decision unit in the scheduling process, determining the computational resource allocated for each task.

The scheduling configuration $\mathcal{C}_i$ for the application $\mathcal{A}_i$ is a collection of the scheduling configurations for the tasks within $\mathcal{A}_i$, and is defined as:

$$\mathcal{C}_i = \{\mathcal{C}_i^j | 1 \leq j \leq |\mathcal{A}_i|\}, \tag{4.2}$$

where $|A_i|$ represents the total number of tasks in application A_i. By grouping task-level scheduling configurations, C_i provides a comprehensive view of how the entire application A_i is distributed across available servers.

Also, the task execution model of one application can exhibit hybrid characteristics, incorporating both sequential and/or parallel processes. Accordingly, each task cannot be executed unless all predecessor tasks complete their execution, while tasks that are not dependent on each other can be executed in parallel. We use $PR(\mathcal{A}_i^j)$ to denote the set of predecessor tasks of task $\mathcal{A}_i^j$ and use $CP(\mathcal{A}_i^j)$ to indicate whether task $\mathcal{A}_i^j$ is located on the critical path of the application $\mathcal{A}_i$.

Response Time Model

Assuming that the scheduling configuration for task $\mathcal{A}_i^j$ is $\mathcal{C}_i^j$, the response time model $T(\mathcal{C}_i^j)$ consists of two parts, the Data Arrival Time (DAT) model $T^{dat}(\mathcal{C}_i^j)$ and the execution time model $T^{ex}(\mathcal{C}_i^j)$:

$$T(\mathcal{C}_i^j) = T^{dat}(\mathcal{C}_i^j) + T^{ex}(\mathcal{C}_i^j). \tag{4.3}$$

The DAT model $T^{dat}(\mathcal{C}_i^j)$ signifies the maximum time for the data, required by task $\mathcal{A}_i^j$, to reach the designated server:

$$T^{dat}(\mathcal{C}_i^j) = max \ \ T^{dat}_{\mathcal{C}_i^k,\mathcal{C}_i^j}, \quad \forall \mathcal{A}_i^k \in PR(\mathcal{A}_i^j), \tag{4.4}$$

where $T^{dat}_{\mathcal{C}_i^k,\mathcal{C}_i^j}$ shows the time consumed for the required data to be transmitted from scheduled server $\mathcal{C}_i^k$ to server $\mathcal{C}_i^j$. Here, $\mathcal{C}_i^j$ signifies the server scheduled for the execution of task $\mathcal{A}_i^j$, while $\mathcal{C}_i^k$ corresponds to the server where the predecessor task of task $\mathcal{A}_i^j$ is executed. Thus, $T^{dat}_{\mathcal{C}_i^k,\mathcal{C}_i^j}$ depends on both the transmission time $T^{tr}_{\mathcal{C}_i^k,\mathcal{C}_i^j}$ and the propagation time $\mathcal{P}_{\mathcal{C}_i^k,\mathcal{C}_i^j}$ for task $\mathcal{A}_i^j$ between server $\mathcal{C}_i^k$ and server $\mathcal{C}_i^j$:

$$T^{dat}_{\mathcal{C}_i^k,\mathcal{C}_i^j} = \begin{cases} T^{tr}_{\mathcal{C}_i^k,\mathcal{C}_i^j} + \mathcal{P}_{\mathcal{C}_i^k,\mathcal{C}_i^j} & \mathcal{C}_i^k \neq \mathcal{C}_i^j, \\ 0 & \mathcal{C}_i^k = \mathcal{C}_i^j. \end{cases} \tag{4.5}$$

where the transmission time $T^{tr}_{\mathcal{C}_i^k,\mathcal{C}_i^j}$ is calculated as follows:

$$T^{tr}_{\mathcal{C}_i^k,\mathcal{C}_i^j} = \frac{DS_{\mathcal{C}_i^k,\mathcal{C}_i^j}(\mathcal{A}_i^j)}{\mathcal{B}_{\mathcal{C}_i^k,\mathcal{C}_i^j}}, \tag{4.6}$$

$DS_{\mathcal{C}_i^k,\mathcal{C}_i^j}(\mathcal{A}_i^j)$ denotes the data size for task $\mathcal{A}_i^j$ sent from server $\mathcal{C}_i^k$ to server $\mathcal{C}_i^j$, and $\mathcal{B}_{\mathcal{C}_i^k,\mathcal{C}_i^j}$ denotes the current bandwidth between server $\mathcal{C}_i^k$ and server $\mathcal{C}_i^j$.

The execution time model $T^{ex}(\mathcal{C}_i^j)$ is defined as the time required to execute task $\mathcal{A}_i^j$ based on scheduling configuration $\mathcal{C}_i^j$. It can be calculated as follows:

$$T^{ex}(\mathcal{C}_i^j) = \frac{L(\mathcal{A}_i^j)}{Freq(\mathcal{C}_i^j)}, \tag{4.7}$$

where $L(\mathcal{A}_i^j)$ denotes the necessary CPU cycles for task $\mathcal{A}_i^j$ to be executed and $Freq(\mathcal{C}_i^j)$ shows the CPU frequency of scheduled server $\mathcal{C}_i^j$ (if the CPU has multiple cores, the model considers the average frequency). Accordingly, the formulation of the response

time model $T(\mathcal{C}_i)$ for application $\mathcal{A}_i$ is expressed as follows:

$$T(\mathcal{C}_i) = \sum_{j=1}^{|\mathcal{A}_i|} (T(\mathcal{C}_i^j) \times CP(\mathcal{A}_i^j)), \tag{4.8}$$

where $CP(\mathcal{A}_i^j)$ is the critical path indicator. If task $\mathcal{A}_i^j$ belongs to the critical path of application $\mathcal{A}_i$, $CP(\mathcal{A}_i^j)$ is set to 1; otherwise, it assumes a value of 0.

System Energy Consumption Model

In this work, we consider the energy consumption of the edge layer and the cloud layer. Given that the scheduling configuration for task $\mathcal{A}_i^j$ is $\mathcal{C}_i^j$, the energy consumption $E(\mathcal{C}_i^j)$ is determined by the energy consumed during the actual task processing (i.e., the execution energy model) $E^{ex}(\mathcal{C}_i^j)$, plus the energy consumed by the servers when transmitting the required data to other servers (i.e., transmission energy model) $E^{tr}(\mathcal{C}_i^j)$:

$$E(\mathcal{C}_i^j) = E^{ex}(\mathcal{C}_i^j) + (E^{tr}(\mathcal{C}_i^j) \times ED(\mathcal{A}_i^j)), \tag{4.9}$$

where $ED(\mathcal{A}_i^j)$ is 0 if $\mathcal{A}_i^j$ is the ending task (i.e., has no successor task) in application $\mathcal{A}_i$ and 1 otherwise.

The execution energy model $E^{ex}(\mathcal{C}_i^j)$ is the energy consumed by the server to execute the task, defined as:

$$E^{ex}(\mathcal{C}_i^j) = T^{ex}(\mathcal{C}_i^j) \times W^{ex}(\mathcal{C}_i^j), \tag{4.10}$$

where $T^{ex}(\mathcal{C}_i^j)$ is obtained from Eq. 4.7 and $W^{ex}(\mathcal{C}_i^j)$ represents the power of the server when executing the task.

Considering the dependency between tasks, one task $\mathcal{A}_i^j$ can have one or more predecessor tasks. The transmission energy model $E^{tr}(\mathcal{C}_i^j)$ is defined as the sum of the energy consumed to transmit the data to the servers where the successor tasks are assigned, as follows:

$$\begin{aligned} E^{tr}(\mathcal{C}_i^j) = & \sum_{\mathcal{A}_i^l \in SU(\mathcal{A}_i^j)} \frac{DS_{\mathcal{C}_i^j, \mathcal{C}_i^l}(\mathcal{A}_i^j)}{\mathcal{B}_{\mathcal{C}_i^j, \mathcal{C}_i^l}} \times W^{tr}(\mathcal{C}_i^j) \\ & \times OS(\mathcal{C}_i^j, \mathcal{C}_i^l), \end{aligned} \tag{4.11}$$

where $SU(\mathcal{A}_i^j)$ denotes the set of successor tasks of task $\mathcal{A}_i^j$, $\mathcal{C}_i^l$ is the scheduling configuration of task $\mathcal{A}_i^l$, $W^{tr}(\mathcal{C}_i^j)$ show the transmission power of the server when transmitting data, and $OS(\mathcal{C}_i^j, \mathcal{C}_i^l)$ is 0 if $\mathcal{C}_i^j$ and $\mathcal{C}_i^j$ are the same server and 1 otherwise. Similar to [243], [244], $W^{tr}(\mathcal{C}_i^j)$ is set as a constant value, but this parameter can also be dynamically adjusted.

Accordingly, the energy consumption model $E(\mathcal{C}_i)$ for application $\mathcal{C}_i$ is formulated as follows:

$$E(\mathcal{C}_i) = \sum_{j=1}^{|\mathcal{A}_i|} E(\mathcal{C}_i^j), \tag{4.12}$$

Monetary Cost Model

The server set $\mathcal{N}$ comprises both the cloud server set $\mathcal{CS}$ and the edge server set $\mathcal{ES}$. Without loss of generality, we assume that the edge servers are on-premises servers and are owned by users, so their execution cost only depends on their electricity usage. Otherwise, the cloud-like pricing model can be used for edge servers. Given that the scheduling configuration for the task $\mathcal{A}_i^j$ is $\mathcal{C}_i^j$, the monetary cost $F(\mathcal{C}_i^j)$ depends both on the cloud server and the edge server price models. Formally, the monetary cost model $F(\mathcal{C}_i^j)$ is:

$$F(\mathcal{C}_i^j) = \begin{cases} \sum\limits_{\mathcal{C}_i^j \in \mathcal{CS}} T(\mathcal{C}_i^j) \times CLP(\mathcal{C}_i^j) & \mathcal{C}_i^j \in \mathcal{CS}, \\ \\ \sum\limits_{\mathcal{C}_i^j \in \mathcal{ES}} E(\mathcal{C}_i^j) \times EP(\mathcal{C}_i^j) & \mathcal{C}_i^j \in \mathcal{ES}, \end{cases} \tag{4.13}$$

where $CLP(\mathcal{C}_i^j)$ shows the cloud server pricing, and $EP(\mathcal{C}_i^j)$ denotes the electricity price for running edge server $\mathcal{C}_i^j$. Consequently, the monetary cost model $F(\mathcal{C}_i)$ for the application $\mathcal{C}_i$ is formulated as follows:

$$F(\mathcal{C}_i) = \sum_{j=1}^{|\mathcal{A}_i|} F(\mathcal{C}_i^j). \tag{4.14}$$

Weighted Cost Model

The weighted cost model $J(\mathcal{C}_i^j)$ is defined as the weighted sum of the normalized response time models, the energy consumption model, and the monetary cost model. Given the scheduling configuration for task $\mathcal{A}_i^j$ is $\mathcal{C}_i^j$:

$$J(\mathcal{C}_i^j) = w_1 \frac{T(\mathcal{C}_i^j) - T^{min}}{T^{max} - T^{min}} + w_2 \frac{E(\mathcal{C}_i^j) - E^{min}}{E^{max} - E^{min}} + w_3 \frac{F(\mathcal{C}_i^j) - F^{min}}{F^{max} - F^{min}}, \tag{4.15}$$

where T^{min}, T^{max}, E^{min}, E^{max}, F^{min}, and F^{max} represent the minimum and the maximum value that can be achieved by the response time model, the energy consumption model, and the monetary cost model, respectively. Also, w_1, w_2, and w_3 are the control parameters used to fine-tune the weighted cost model. The reason for employing normalized models, rather than the original models, is that the values of the models may fall within different ranges.

Accordingly, the weighted cost model for application $\mathcal{A}_i$ is defined as:

$$J(\mathcal{C}_i) = w_1 \times Norm(T(\mathcal{C}_i)) + w_2 \times Norm(E(\mathcal{C}_i)) + w_3 \times Norm(F(\mathcal{C}_i)), \tag{4.16}$$

where $T(\mathcal{C}_i)$, $E(\mathcal{C}_i)$, and $F(\mathcal{C}_i)$ are obtained from Eq. 4.8 , Eq. 4.12 and Eq. 4.14, and $Norm$ represents the normalization.

Therefore, the optimization problem of scheduling IoT applications can be formu-

lated as:

$$min \quad J(\mathcal{C}_i) \tag{4.17}$$

$$\text{s.t.} \quad C1: \ Size(\mathcal{C}_i^j) = 1, \ \forall \mathcal{C}_i^j \in \mathcal{C}_i \tag{4.18}$$

$$C2: \ DS_{\mathcal{C}_i^k,\mathcal{C}_i^j}(\mathcal{A}_i^j), \mathcal{B}_{\mathcal{C}_i^k,\mathcal{C}_i^j} > 0, \ \forall \mathcal{C}_i^k, \mathcal{C}_i^j \in \mathcal{N}, \ \forall \mathcal{A}_i^j \in \mathcal{A}_i \tag{4.19}$$

$$C3: \ Freq(\mathcal{N}_k), Ram(\mathcal{N}_k) > 0, \ \forall \mathcal{N}_k \in \mathcal{N} \tag{4.20}$$

$$C4: \ \sum_{\mathcal{A}_i \in \mathcal{A}} \sum_{\mathcal{A}_i^j \in \mathcal{A}_i} Ram(\mathcal{A}_i^j) \times SO(\mathcal{A}_i^j, \mathcal{N}_k) < Ram(\mathcal{N}_k), \ \forall \mathcal{N}_k \in \mathcal{N} \tag{4.21}$$

$$C5: \ T(\mathcal{A}_i^j) \leq T(\mathcal{A}_i^j + \mathcal{A}_i^k), \forall \mathcal{A}_i^j \in PR(\mathcal{A}_i^k) \tag{4.22}$$

$$C6: \ w_1 + w_2 + w_3 = 1, \ 0 \leq w_1, w_2, w_3 \leq 1 \tag{4.23}$$

where $C1$ enforces the rule that each task can be assigned to only one server. $C2$ specifies the transmission constraints for data size and bandwidth. Additionally, $C3$ defines constraints related to the available CPU frequency and available RAM size of the server by setting a lower bound. Furthermore, $C4$ ensures that every server has adequate RAM resources to process all tasks scheduled on it, preventing resource overutilization. $SO(\mathcal{A}_i^j, \mathcal{N}_k)$ equals 1 if task $\mathcal{A}_i^j$ is scheduled on server $\mathcal{N}_k$, otherwise 0. $C5$ specifies that each task is eligible for processing only after the completion of its predecessor tasks, ensuring that the accumulative cost is no less than that of the predecessor task. Lastly, $C6$ places restrictions on the control parameters within Eq. 4.16, confining them to values between 0 and 1.

The problem under consideration is characterized as a non-convex optimization problem, primarily due to the potential existence of an infinite number of local optima within the feasible domain. Typically, algorithms aimed at finding the global optimum in such problems exhibit exponential complexity and are classified as NP-hard [201]. To solve such a non-convex optimization problem, most approaches decompose these problems into several convex sub-problems [245], subsequently solving these sub-problems iteratively until convergence is achieved [202]. However, this strategy often sacrifices accu-

racy for reduced complexity [203]. Also, these approaches are heavily dependent on the current environment and are not suitable for dynamic environments with highly heterogeneous computational resources [30]. To tackle this issue, we propose TF-DDRL to adaptively manage uncertainties in dynamic and stochastic environments. It can dynamically learn scheduling policies through continuous interaction with the environment.

4.3.3 Deep Reinforcement Learning Model

To apply the DRL approach, the optimization problem should be formulated as a MDP. More specifically, the problem can be defined by the tuple $< \mathbb{S}, \mathbb{A}, \mathbb{P}, \mathbb{R}, \gamma >$, where $\mathbb{S}$ signifies a finite set of states, $\mathbb{A}$ represents a finite set of actions, $\mathbb{P}$ represents the state transition probability, $\mathbb{R}$ stands for the reward function, and $\gamma \in [0, 1]$ serves as the discount factor employed in calculating cumulative rewards.

We consider the learning process to be divided into multiple time steps t within a total time span $\mathbb{T}$. At each time step, the agent interacts with the environment, resulting in multiple states S_t. At time step t, the agent observes the environment state $S_t = s$, where $s \in \mathbb{S}$. Guided by the policy $\pi(a|s)$, where $a \in \mathbb{A}$, the agent chooses an action $A_t = a$. The policy function $\pi(a|s) = Pr[A_t = a|S_t = s]$ explicitly defines the probability of selecting action a given state s. Following the execution of action a, the agent receives a reward $r = \mathbb{R}[S_t = s, A_t = a]$ from the environment, determined by the reward function $\mathbb{R}$. The agent then undergoes a state transition to $S_{t+1} = s'$ based on the state transition function $P^a_{ss'} = \mathbb{P}[S_{t+1} = s'|S_t = s, A_t = a]$. The ultimate objective of the agent is to acquire a policy π maximizing the expected cumulative discounted reward, denoted as $\mathbb{E}\pi[\sum t \in T\gamma_t r_t]$.

Considering the scheduling problem of IoT applications in edge and cloud computing environments, the MDP's state space $\mathbb{S}$, action space $\mathbb{A}$, and reward function $\mathbb{R}$ are defined as follows:

- **State space** $\mathbb{S}$: In this work, the formulated problem pertains to tasks and servers, with the state $\mathbb{S}$ containing $\mathbb{F}$ for the task feature and $\mathbb{G}$ for the server set state. At time step t, the feature space $\mathbb{F}$ of task $\mathcal{A}^j_i$ captures essential details related to the

task, defined as:

$$\mathbb{F}_t(\mathcal{A}_i^j) = \{f_t^y(\mathcal{A}_i^j) | \mathcal{A}_i^j \in \mathcal{A}_i, 0 \leq y \leq |\mathbb{F}|\}, \tag{4.24}$$

where y denotes the feature index and $|\mathbb{F}|$ represents the total number of features. Specifically, the feature space $\mathbb{F}$ includes task ID, application ID, required CPU cycles $L(\mathcal{A}_i^j)$, required RAM size $Ram(\mathcal{A}_i^j)$, task dependencies (predecessors $PR(\mathcal{A}_i^j)$ and successors $SU(\mathcal{A}_i^j)$), previously configured tasks and their scheduled servers, execution status of dependent tasks, etc. This comprehensive feature space enables the DRL agent to make informed scheduling decisions.

Also, in time step t, the state space $\mathbb{G}$ of the server set $\mathcal{N}$ contains the number of servers, the CPU frequency, RAM size, label (e.g., cloud or edge), expense per time unit (for cloud servers), electricity price (for edge servers), propagation time, bandwidth between different servers, etc, which is formally defined as:

$$\begin{aligned} &\mathbb{G}_t(\mathcal{N}) = \{|\mathcal{N}|, g_t^z(\mathcal{N}_k), h_t^q(\mathcal{N}_j, \mathcal{N}_k)| \\ &\mathcal{N}_j, \mathcal{N}_k \in \mathcal{N}, 0 \leq z \leq |g|, 0 \leq q \leq |h|\}, \end{aligned} \tag{4.25}$$

where g is the sub-state set containing states associated with an individual server (e.g., CPU utilization), and z corresponds to its index. Additionally, h signifies the sub-state set containing states associated with two servers (e.g., propagation time), and q denotes the index. Consequently, $\mathbb{S}$ is defined as:

$$\mathbb{S} = \{S_t = (\mathbb{F}_t(\mathcal{A}_i^j), \mathbb{G}_t(\mathcal{N})) | \mathcal{A}_i^j \in \mathcal{A}_i, t \in \mathbb{T}\}. \tag{4.26}$$

- **Action space** $\mathbb{A}$: In this work, scheduling involves the action of assigning the current task $\mathcal{A}_i^j$ to an individual server $\mathcal{N}_k$. Consequently, the definition of the action at time step t is as follows:

$$A_t = \mathcal{C}_i^j = \mathcal{N}_k. \tag{4.27}$$

Therefore, the action space $\mathbb{A}$ equals to the server set N:

$$\mathbb{A} = \mathcal{N}. \tag{4.28}$$

- **Reward function** $\mathbb{R}$: As outlined in Section 4.3.2, the primary objective is to minimize the weighted cost model presented in Eq. 4.17. Thus, in time step t, the reward r_t can be defined as the negative value of Eq. 4.15 if the task can be successfully executed. However, if the task $\mathcal{A}_i^j$ fails to be executed on the scheduled server $\mathcal{C}_i^j$, a substantial negative value is introduced as a penalty. Formally, r_t is defined as:

$$r_t = \begin{cases} -J(\mathcal{C}_i^j) & succeed \\ penalty & fail, \end{cases} \tag{4.29}$$

4.4 TF-DDRL: Distributed DRL Framework

The high-level architecture of the TF-DDRL framework is depicted in Fig. 4.2. The architecture comprises multiple Actors responsible for collecting data to create experience trajectories and a Learner that leverages the experience trajectories to learn a policy π. The architecture comprises multiple distributed Actors, which are responsible for interacting with the environment, collecting data, and generating experience trajectories by executing tasks based on their local policies. These experience trajectories are then sent to a Learner, whose role is to aggregate these trajectories, update the global policy π, and broadcast the updated policy back to the Actors to ensure consistent and improved decision-making across the system. Both the Actors and the Learner can be flexibly deployed on edge or cloud servers, depending on the systems requirements. For example, deployment on cloud servers may be preferred when higher computational capacity is needed, whereas edge servers may be more suitable when low latency or data locality is prioritized. The primary objective is to identify a policy π that maximizes the expected sum of future discounted rewards:

$$V^{\pi}(s) = \mathbb{E}_{\pi}[\sum_{t \in T} \gamma_t r_t], \tag{4.30}$$

where π represents the policy, $\gamma \in [0, 1]$ is the discount factor, $r_t = r(s_t, a_t)$ denotes the reward at time t, s_t is the state at time t, s is the initial state s_0, and $a_t = \pi(a_t|s_t)$ is the action generated by following a specific policy π. Fig. 4.3 presents an overview of the

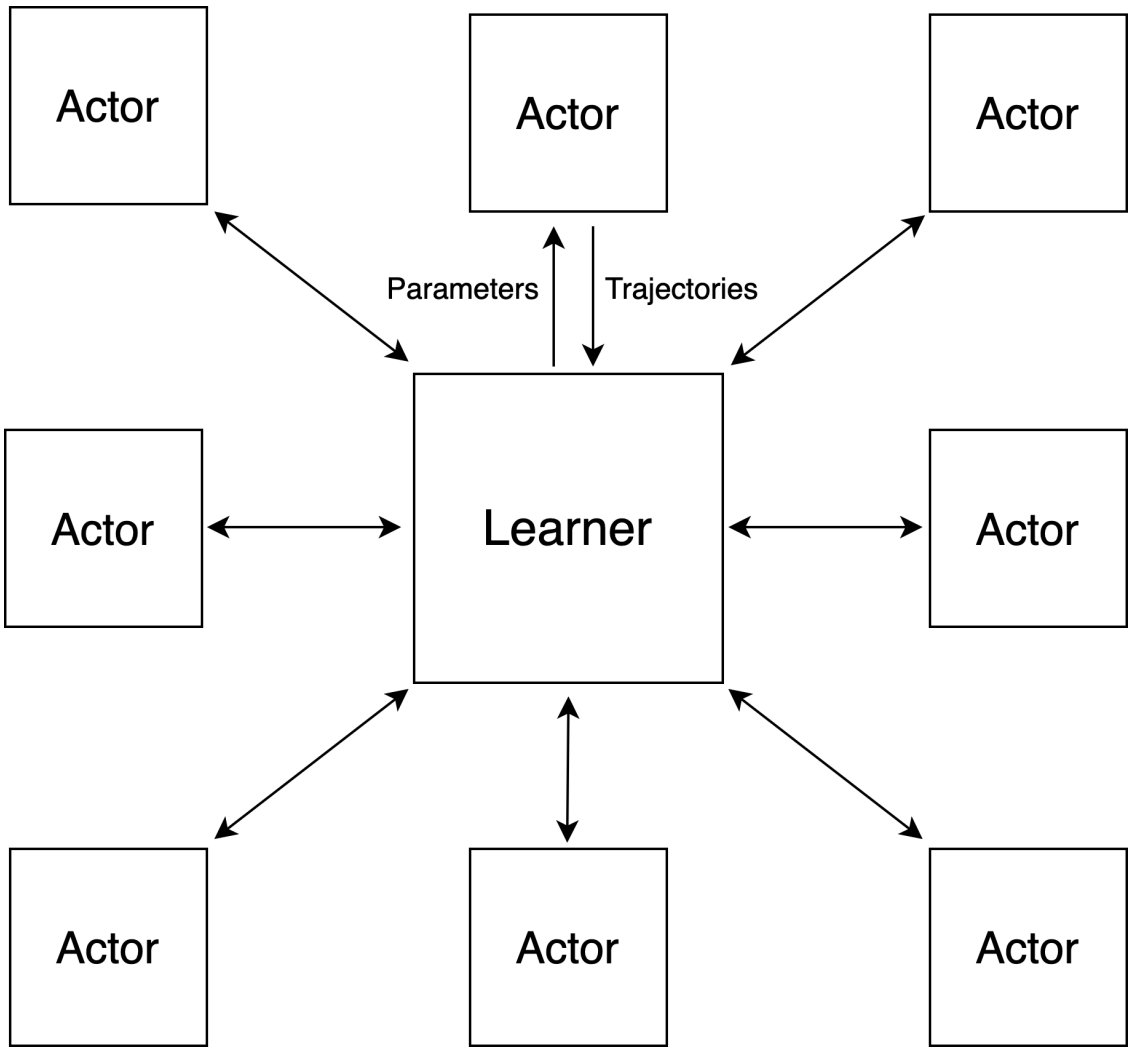


Figure 4.2: High-level architecture of Actors and Learner

TF-DDRL framework. In what follows, each component and communication process is discussed in detail.

4.4.1 Actor: Experience Trajectories Generation

Algorithm 4.1 describes how the Actor in the TF-DDRL framework generates experience trajectories. In order to improve the efficiency of sampling and the speed of convergence of TF-DDRL, PER is introduced to store the trajectory experiences of the Actor. At the beginning, the Actor updates its local policy μ to the most recent Learner policy π and

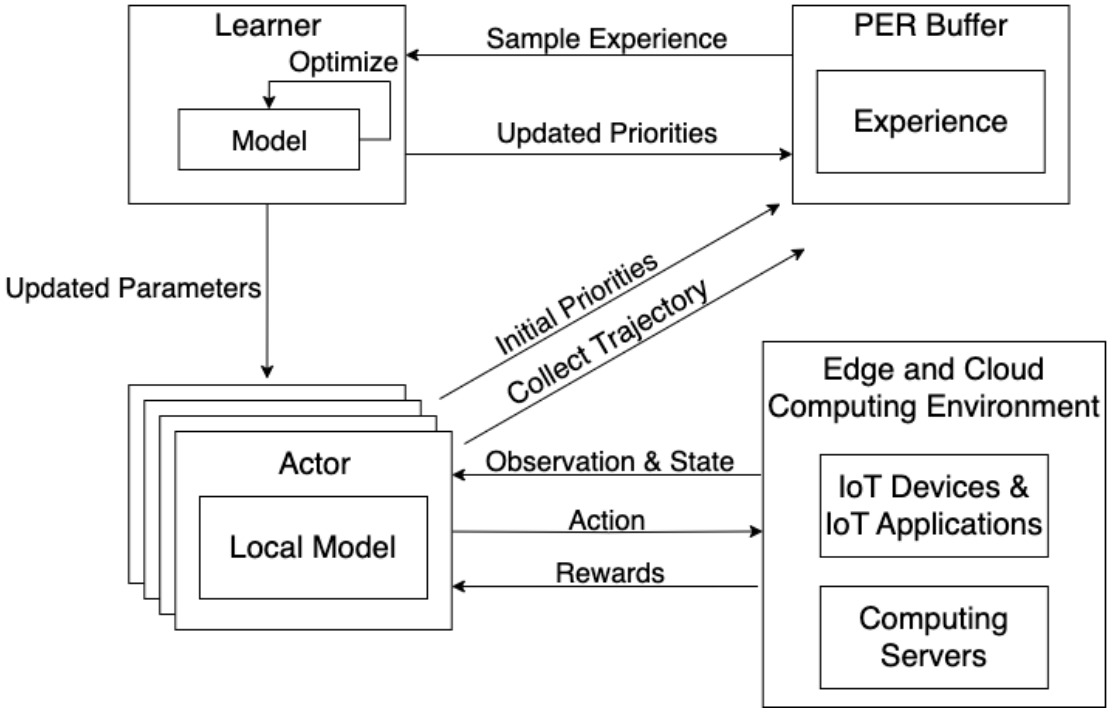


Figure 4.3: An overview of TF-DDRL framework

initializes a PER buffer $\mathcal{P}$ to store the collected transitions. Before one trajectory, the Actor generates the initial state based on the information of the current task and server set. After that, based on the output a_t of the policy μ, the Actor schedules the current task to the corresponding server. Then, the reward r_t of the current action a_t is calculated based on Eq. 4.29, and the next state s_{t+1} is also generated based on the information of the next task and the server set. Afterward, the Actor calculates the importance measure m_t and stores the current transition $(s_t, a_t, r_t, \mu(a_t|s_t), s_{t+1})$ into $\mathcal{P}$ based on m_t. After n steps, the Actor sends the trajectory $\{s_1, a_1, r_1, \mu(a_1|s_1), ..., s_n, a_n, r_n, \mu(a_n|s_n), s_{n+1}\}$ to the Learner. The Learner then iteratively updates its policy π over a batch of trajectories gathered from different Actors. This framework decouples data collection and learning, allowing more Actors to be added and distributed across multiple machines for efficient utilization of computing resources in edge and cloud IoT systems.

Algorithm 4.1: Actor: experience generation

Input : the Actors's local policy μ; the Learner's policy π; the Learner's address *Learner*; max time step n;

```
while True do
    μ ← UpdateActorPolicy(π, Learner)
    P ← InitializePERBuffer()
    servers ← GetServers()
    task ← GetTask()
    s_1 ← GenerateState(servers, task)
    for t ← x to x + n − 1 do
        a_t ← μ(s_t)
        Schedule(task, a_t)
        r_t ← GetReward()
        servers ← GetServers()
        task ← GetTask()
        s_{t+1} ← GenerateState(servers, task)
        e_t = (s_t, a_t, r_t, μ(a_t|s_t), s_{t+1})
        m_t = r_t + γ_t V(s_{t+1}) − V(s_t)
        Store transition e_t into P based on m_t
    end for
    if Length(P) == n then
        SubmitTrajectory(P, Learner)
    end if
end while
```

4.4.2 Learner: Schedule Policy Update

However, it's worth noting that after a few updates, the Actor's strategy μ may fall behind the Learner's strategy π. To address the gap between the Actor's policy μ and the Learner's policy π, an off-policy correction method named V-trace [93] is introduced to rectify this discrepancy.

V-trace Correction Method

The Learner in TF-DDRL maintains a state value function V based on the samples from the Actors. The purpose of the V-trace correction method is to provide an estimate of the current state value function V, called V-trace target $\hat{V}$. After n steps of interaction with the environment, the Actor collects a trajectory $(s_t, a_t, r_t, \mu(a_t|s_t), s_{t+1})_{t=x}^{t=x+n}$ following its policy μ. The n-steps V-trace target $\hat{V}(s_x)$ for state s_x is:

$$\hat{V}(s_x) = V(s_x) + \sum_{t=x}^{x+n-1} \gamma^{t-x} (\prod_{i=x}^{t-1} c_i) \delta_t V, \tag{4.31}$$

where $\delta_t V$ is the truncated Temporal Difference (TD) for V, and $\prod_{i=x}^{t-1} c_i$ measures the impact of $\delta_t V$ observed at time t on the update of the value function V at the previous time x. Specifically, $\delta_t V$ is defined as:

$$\delta_t V = \rho_t(r_t + \gamma_t V(s_{t+1})) - V(s_t)), \tag{4.32}$$

and c_i and ρ_t are truncated importance sampling weights,:

$$c_i = min(\bar{c}, \frac{\pi(a_i|s_i)}{\mu(a_i|s_i)}), \tag{4.33}$$

$$\rho_t = min(\bar{\rho}, \frac{\pi(a_t|s_t)}{\mu(a_t|s_t)}), \tag{4.34}$$

where $\bar{c}$ and $\bar{\rho}$ are the truncation constants with $\bar{c} \leq \bar{\rho}$. $\bar{c}$ affects the speed of convergence, while $\bar{\rho}$ affects the solution to which the value function V converges. Considering $\bar{\rho}$, the corresponding target policy $\pi_{\bar{\rho}}(a|x)$ is defines as:

$$\pi_{\bar{\rho}}(a|x) = \frac{min(\bar{\rho}\mu(a|x), \pi(a|x))}{\sum_{b \in A} min(\bar{\rho}\mu(b|x), \pi(b|x))} \tag{4.35}$$

Actor-Critic-based Algorithm

The implementation of TF-DDRL follows the Actor-Critic architecture. TF-DDRL optimizes two DNNs, the actor (policy) network and the critic (value) network. The actor network focuses on acquiring a policy π to maximize the expected cumulative discounted reward $\mathbb{E}_\pi[\sum_{t \in T} \gamma_t r_t]$. Meanwhile, the critic network evaluates the current policy π by computing the TD error, which measures the difference between the current reward and the estimate of the value function V.

Algorithm 4.2 describes how the Learner in the TF-DDRL framework updates policies. The Learner first obtains the collected trajectories from all Actors. In order to improve the efficiency of sampling and the speed of convergence of the algorithm, trajectory experiences are sampled based on importance measure m_x. When updating the

networks, the loss function of TF-DDRL is defined as follows:

$$loss_{total} = a_v * loss_{value} + a_p * loss_{policy} + a_e * loss_{entropy}, \tag{4.36}$$

where $loss_{value}$ is the loss function for value function, $loss_{policy}$ is the loss function for policy, $loss_{entropy}$ is the loss function for entropy bonus, and a_v, a_p, and a_e are the corresponding weights. Considering π_ϕ is the current policy parameterized by ϕ, V_θ is the value function parameterized by θ, and μ is the Actor's local policy, the value loss function $loss_{value}$ is defined as the L_2 loss between the current value V_θ and the V-trace target value $\hat{V}$:

$$loss_{value} = (\hat{V}(s_x) - V_\theta(s_x))^2, \tag{4.37}$$

where $\hat{V}(s_x)$ is from Eq. 4.31. Considering the objective function Eq. 4.30, the policy gradient can be presented as:

$$\nabla V^\pi(s) = \mathbb{E}_\pi[\nabla log \pi(a_x|s_x) Q_\pi(s_x, a_x)], \tag{4.38}$$

where $Q_\pi(s_x, a_x)$ is the state-value of policy π at (s_x, a_x). In TF-DDRL, the truncated importance sampling weight ρ_x between the policy $\pi_{\bar{\rho}}$ and the Actor's local policy μ is employed to suppress the divergence. Also, we use $r_s + \gamma v_{s+1}$, named as the v-trace advantage, to estimate $Q_{\pi_{\bar{\rho}}}(a_x|s_x)$. Besides, state-dependent baseline $V_\theta(s_x)$ is subtracted from the v-trace advantage to reduce bias. Therefore, the policy loss function $loss_{policy}$ is defined as:

$$loss_{policy} = -\rho_x log \pi_\phi(a_x|s_x)(r_x + \gamma v_{x+1} - V_\theta(s_x)), \tag{4.39}$$

where ρ_x is from Eq. 4.34. We also exploit the entropy $H(\pi_\phi)$ as a bonus to encourage exploration, with the loss function $loss_{entropy}$ defined as:

$$loss_{entropy} = -H(\pi_\phi) = \sum_a \pi_\phi(a_x|s_x) log \pi_\phi(a_x|s_x) \tag{4.40}$$

Therefore, the value function parameter θ is updated in the direction of:

$$\Delta\theta = a_v * (\hat{V}(s_x) - V_\theta(s_x)) \nabla_\theta V_\theta(s_x), \tag{4.41}$$

and the policy parameter ϕ is updated through policy gradient:

$$\begin{aligned}\Delta\phi = a_p\rho_x\nabla_\phi log\pi_\phi(a_x|s_x)(r_x+\gamma v_{s+1}-V_\theta(s_x)) \\ -a_e\nabla_\phi\sum_a \pi_\phi(a_x|s_x)log\pi_\phi(a_x|s_x).\end{aligned} \tag{4.42}$$

Algorithm 4.2: Learner: policy update

Input : current policy π_ϕ; value function V_θ; update epoch X; buffer size N; value function loss coefficient a_v; policy objective function loss coefficient a_c; entropy bonus loss coefficient a_e; the Actors set $Actors$

while *True* **do**
  $\mathcal{D} \leftarrow InitializeBuffer()$
  for *actor in Actors* **do**
    $\mathcal{D}.append(ReceiveTrajectory(actor))$
  end for
  for *trajectory in* $\mathcal{D}$ **do**
    for $x \leftarrow 1$ *to* X **do**
      Sample experience $e_x = (s_x, a_x, r_x, \mu(a_x|s_x), s_{x+1}) \sim \mathcal{M}(x) = m_x^\alpha / \sum_i m_i^\alpha$
      $w_x = (N\mathcal{M}(x))^{-\beta} / max_i w_i$
      $\hat{V}(s_x) \leftarrow V_\theta(s_x) + \sum_{t=x}^{x+n-1}\gamma^{t-x}(\prod_{i=x}^{t-1} c_i)\delta_t V_\theta$
      $m_x \leftarrow |\delta_t|$
      $loss_{value} \leftarrow (\hat{V}(s_x) - V_\theta(s_x))^2$
      $loss_{policy} \leftarrow -\rho_x log\pi_\phi(a_x|s_x)(r_x+\gamma v_{x+1}-V_\theta(s_x))$
      $loss_{entropy} \leftarrow \sum_a \pi_\phi(a_x|s_x)log\pi_\phi(a_x|s_x)$
      $loss_{total} \leftarrow a_v * loss_{value} + a_p * loss_{policy} + a_e * loss_{entropy}$
      $\Delta\theta \leftarrow \Delta\theta + w_x a_v(\hat{V}(s_x) - V_\theta(s_x))\nabla_\theta V_\theta(s_x)$
      $\Delta\phi \leftarrow \Delta\phi + w_x(a_p\rho_x\nabla_\phi log\pi_\phi(a_x|s_x)(r_x+\gamma v_{s+1}-V_\theta(s_x)) - a_e\nabla_\phi\sum_a \pi_\phi(a_x|s_x)log\pi_\phi(a_x|s_x))$
    end for
    update θ and ϕ by Adam optimizer
  end for
  $BroadcastPolicy(Actors, \pi_\phi)$
end while

4.4.3 Prioritized Experience Replay

The Learner in TF-DDRL relies on the experience trajectories collected from Actors to update the parameters. However, in dynamic edge and cloud environments, the experience changes over time, resulting in significant gaps between samples. Each sample can contribute to different improvements to the model. To enhance sampling efficiency, expedite convergence, and enable the model to quickly adapt to changes by focusing

on the most pertinent experiences during the training phase, PER [230] is introduced in both the data collection phase and the model update phase.

As presented in Algorithm 4.1, during the data collection phase, the Actor assigns importance measure m_t to each experience sample when storing it in the buffer. Since the TD error reflects the difference between the model's estimated value of the current state and the next state, and when this difference is significant, it indicates that the experience sample provides valuable information for updating the current policy. Therefore, TF-DDRL uses the TD error as a metric to measure the importance of samples, defined as:

$$m_t = r_t + \gamma_t V(s_{t+1}) - V(s_t) + \epsilon, \tag{4.43}$$

where ϵ is a tiny positive number from 0 to 1, in case the experience is not sampled when the TD error is 0. As presented in Algorithm 4.2, when the Learner samples experience e_x from the trajectory, the sampling probability $\mathcal{M}(x)$ is calculated as follows:

$$e_x \sim \mathcal{M}(x) = \frac{m_x^\alpha}{\sum_i m_i^\alpha} \tag{4.44}$$

where α determines the degree of priority, and $\alpha = 0$ corresponds to the uniform case (i.e., each experience has the same probability of being sampled).

However, when experiences are given priority, they have different probabilities of being sampled, which will introduce bias in the update of the value network, thus changing the direction of the convergence of the value network. In order to correct this error, an importance-sampling weight is added to each empirical sample, calculated as follows:

$$w_x = (\frac{1}{N\mathcal{M}(x)})^\beta * (max_i w_i)^{-1}, \tag{4.45}$$

where N is the number of experience samples in the PER buffer, β is a hyperparameter within 0 and 1 that will gradually increase and finally settle at 1, and $(max_i w_i)^{-1}$ is to normalize the weight to improve stability. The purpose of using the importance-sampling weight is to strike a balance between prioritizing samples to learn important experiences and reducing the potential bias. As β continues to rise to 1, the bias gradually decreases, and the learning process gradually reduces the impact of prioritization,

ensuring a more stable and unbiased learning process. This helps prevent the model from becoming too sensitive to specific experiences, encouraging a more robust and accurate learning process.

4.4.4 Gated Transformer-XL

Due to the heterogeneity and dynamics of edge and cloud environments, TF-DDRL uses the Gated Transformer-XL [229] to allow the model to better capture long-term dependencies and global relationships between states. The network architecture of TF-DDRL is shown in Fig. 4.4.

In the Transformer layer of TF-DDRL, the Multi-Head Attention block applies the attention mechanism to different linear mappings (heads) of the input and concatenates them together, allowing the model to focus on different parts of the input sequence simultaneously, which helps to capture the relationships between different features in the input. The Position-wise Multi-Layer Perceptron (MLP) block is used to perform independent nonlinear transformations of features at each position, enhancing the model's ability to capture complex patterns and relationships at different positions in the input sequence, providing a more expressive representation for downstream processing. The Gating Layer is used to weight the features of each position when passing through different blocks. The model can control the flow of input information by learning the appropriate weights, making it more suitable for specific tasks and data distribution.

While the inclusion of Transformers in TF-DDRL can enhance the models ability to capture long-term dependencies in scheduling tasks, it also introduces additional computational overhead due to its multi-head attention mechanism. However, by leveraging the distributed Actor-Learner architecture, TF-DDRL distributes the computational burden across multiple servers, thus mitigating the impact on individual nodes. This design choice allows the framework to maintain scalability and efficiency in dynamic IoT environments, despite the added complexity. Additionally, the use of PER further optimizes sampling efficiency, reducing the exploration costs associated with training.

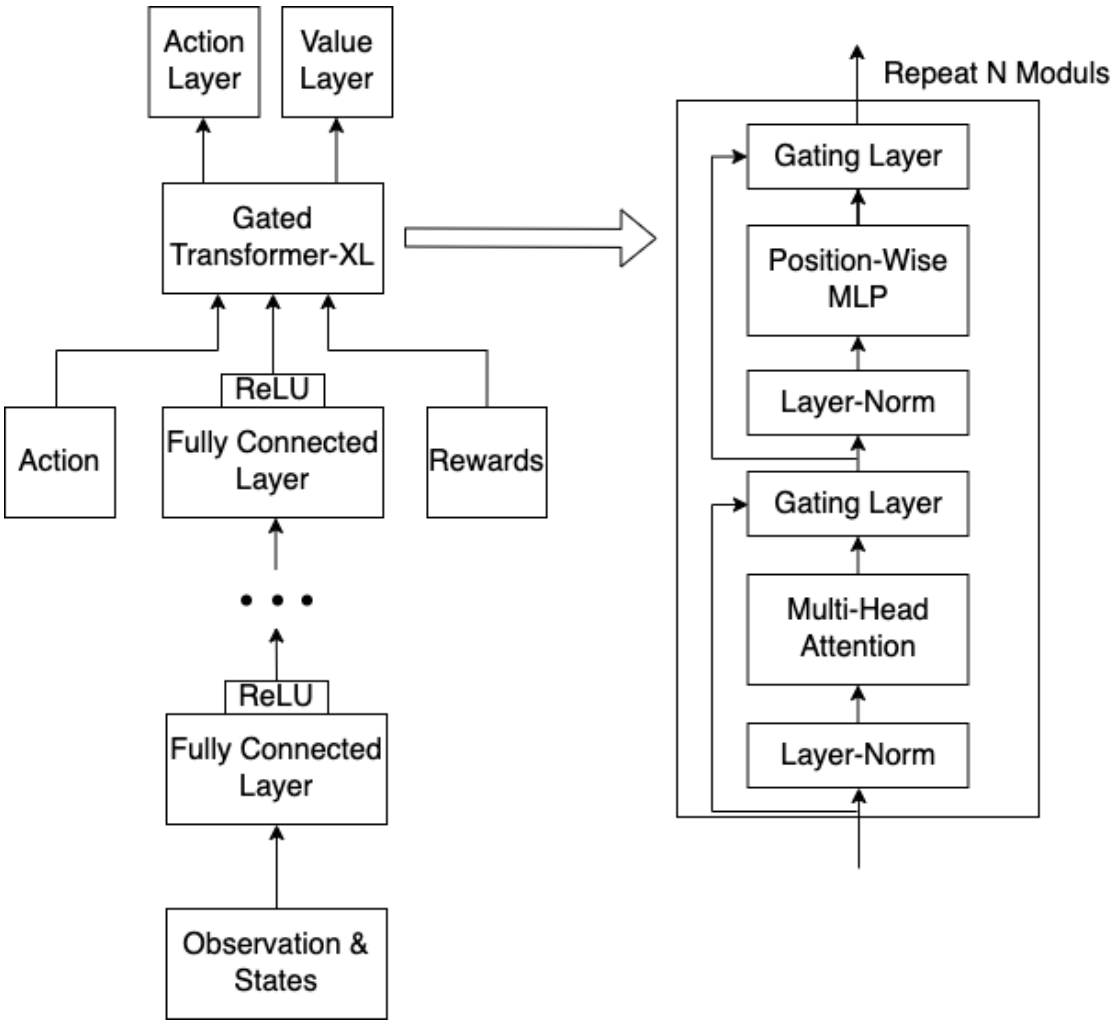


Figure 4.4: The network architecture of TF-DDRL framework

4.5 Performance Evaluation

This section introduces the experiment configuration, hyperparameters of the TF-DDRL, and the performance study.

4.5.1 Experiment Setup

We discuss the specification of our practical edge-cloud environment, details of employed real IoT applications, and baseline techniques.

Practical Experiment Environment

To reflect the heterogeneous computing environments, a practical experiment environment, containing IoT devices, edge servers, and cloud servers, is established. Besides, to build a multi-cloud computing environment, we used three instances from the Nectar Cloud infrastructure (All AMD EPYC with 2 cores @2.0GHz, 8GB RAM; 4cores @2.0GHz, 16GB RAM; 8 cores @2.0GHz, 32GB RAM), one instance from AWS Cloud (Intel Xeon with 1 core @2.4GHz, 1GB RAM), and one instance from Microsoft Azure Cloud (Intel Xeon with 1 core @2.3GHz, 1GB RAM).

In the edge computing environment, we used one RPi 3B (Pi OS, Broadcom BCM2837 with 4 cores @1.2GHz, 1GB RAM), one Macbook Pro (macOS, M1 Pro with 8 cores, 16GB RAM), and one Dell laptop (Linux, Intel Core i7 with 8 cores @2.3GHz, 16GB RAM). Also, as IoT devices, we have used webcams, IP cameras, and docker containers that stream pre-recorded video files.

Moreover, we used the Victorian Default Offer[1] (i.e., 0.2871 AUD/kWh) in Australia as the electricity price, and the official price of AWS[2] and Microsoft Azure[3] cloud servers (i.e., 0.1296 AUD/hour for m6a.large, 0.2592 AUD/hour for m6a.xlarge, 0.5184 AUD/hour for m6a.2xlarge, 0.0174 AUD/hour for t2.micro, and 0.0156 AUD/hour for B1s) to calculate the monetary cost in the experiment. In our environment, the servers exhibit the following average latency and bandwidth (data rate): the latency between the IoT device and the Nectar cloud server ranges from 6-12ms, with a bandwidth between 14-20MB/s; between the IoT device and the AWS cloud server, the latency ranges from 15-25ms, with a bandwidth between 15-22MB/s; between the IoT device and the Microsoft Azure cloud server, the latency ranges from 7-15ms, with a bandwidth between 15-21MB/s; the latency between the IoT device and the edge servers ranges from 1-6ms, with a bandwidth between 130-140 MB/s. The energy consumed to execute applications on servers is monitored using the eco2AI library [246]. Moreover, the transmission power $W^{tr}(\mathcal{C}_i^j)$ of servers is obtained similar to [243], [244], and $W^{tr}(\mathcal{C}_i^j)$ is set between 0.75-1 watt for edge servers and between 3-5 watt for cloud servers. However, these

[1]https://www.esc.vic.gov.au/electricity-and-gas/prices-tariffs-and-benchmarks/victorian-default-offer

[2]https://aws.amazon.com/pricing

[3]https://azure.microsoft.com/pricing

parameters can be adjusted.

Furthermore, in Eq. 4.16, w_1, w_2, and w_3 are set to 0.33, indicating that the importance of response time, energy consumption, and monetary cost are considered equal.

Sample IoT Applications

To evaluate TF-DDRL's performance, we utilized four types of IoT applications, featuring real-time and/or non-real-time capabilities. Real-time functionality allows applications to process live streams, while non-real-time functionality facilitates the processing of pre-recorded video files. These applications, adhering to a sensor-actuator architecture, are dynamically distributed across the heterogeneous IoT devices in both edge and cloud environments. Each application can operate on multiple devices simultaneously, creating a realistic and diverse application scheduling environment. Additionally, all applications offer an adjustable parameter known as the *application label*, which determines the resolution of the video. The applications are detailed below:

- *Face Detection* [178]: Identifies human faces in real-time, marking them with squares in the video. This application is implemented using the OpenCV[4].

- *Color Tracking* [178]: Traces colors in a video stream in real-time. Users have the flexibility to dynamically configure target colors using the application's GUI. This application is developed using OpenCV[4].

- *Face And Eye Detection* [178]: Alongside identifying human faces in real-time, it detects human eyes. This application is developed using OpenCV[4].

- *Video OCR* [44]: Retrieves textual content from pre-recorded video and presents it to the user. It is designed to automatically filter keyframes for efficient processing. This application is developed using Googles Tesseract-OCR Engine[5].

[4]https://github.com/opencv/opencv
[5]https://github.com/tesseract-ocr/tesseract

Baseline Techniques

To evaluate the performance of TF-DDRL, we implemented five additional DRL techniques, including centralized and distributed, as outlined below:

- *IMPALA* [93]: It is a distributed DRL technique and is designed for large-scale environments. TF-DDRL is based on the architecture of IMPALA to enable high scalability in highly distributed environments.

- *ApeX-DQN* [92]: It is an improved DRL technique based on DQN that introduces a distributed learning architecture, adopted by Wen et al. [236] for scheduling problems.

- *A3C* [88]: It is one of the most adapted techniques in the distributed DRL field for scheduling problems. It has been used by many works in the current literature, including [237], [238], [239], [240], and [241]. It combines the Actor-Critic method with the concept of concurrent execution. We extend this technique to solve the proposed optimization problem in the heterogeneous edge and cloud computing environment.

- *D3QN-RNN*: Many works ([231], [191], [234], [140], [76], and [189]) use DQN-based DRL techniques. We extend the foundation of DQN, incorporating the Dueling architecture [74] to decompose the Q values into state and advantage values for a more accurate estimation of the relative value of actions. Also, we introduce Double DQN [73], employing two independent neural networks to estimate target Q-values to address the overestimation during training. Moreover, RNN is used in this technique.

- *SAC* [84]: It is a centralized DRL technique and is used by Zheng et al. [193]. It combines the Actor-Critic method with entropy regularization, encouraging exploration, and enhancing the stability of learning. It is extended to address our problem within the heterogeneous edge and cloud computing environment.

4.5.2 Technique Hyperparameters

The network architecture of TF-DDRL is depicted in Fig. 4.4. In our implementation, we used three fully connected layers, followed by two Gated Transformer-XL-based attention layers, and then two additional fully connected layers for generating action logits and the value function. Furthermore, we performed a grid search to fine-tune the hyperparameters. Accordingly, we set the learning rate (lr) to 0.001 and the discount factor (γ) to 0.99. Also, the $\bar{c}$ and $\bar{\rho}$, governing the V-trace performance, are both set to 1 for optimal results. Table 4.3 provides a summary of the hyperparameter settings. Moreover, we conducted hyperparameter tuning for the baseline techniques to ensure a fair assessment of their performance, as presented in Table 4.4.

Table 4.3: The hyperparameters setting for TF-DDRL

TF-DDRL Hyperparameter	**Value**
Fully Connected Layers	3
Hidden Layer Units	[256, 256, 128]
Activation Function	ReLU
Learning Rate lr	0.001
Discount Factor γ	0.99
Transformer Unit Number	2
Transformer Head Number	4
Transformer Head Dimension	32
Transformer Position-wise MLP Dimension	32
Optimization Method	Adam

Table 4.4: Hyperparameters of baseline techniques

Hyperparameters	**ApeX-DQN**	**A3C**	**D3QN-RNN**	**SAC**
Fully Connected Layers	3	3	3	3
Hidden Layer Units	[256,256,128]	[256,256,128]	[256,256,128]	[256,256,128]
Activation Function	ReLU	TanH	ReLU	ReLU
Learning Rate	0.001	0.001	0.001	0.0001
Discount Factor	0.99	0.9	0.99	0.99

4.5.3 Performance Study

The results of our extensive experiments are shown below.

PER and Transformer Analysis

This experiment studies the performance of TF-DDRL compared to native IMPALA. We employ the four applications detailed in Section 4.5.1 for training. The results are provided exclusively for weighted costs.

Figure 4.5 shows the outcome of TF-DDRL under various model configurations. Without the use of both PER and Transformer, the native IMPALA requires approximately 90 iterations to converge to the optimal solution discovered in the experiment. The convergence speed slightly improves when only PER is employed. However, with the exclusive employment of the Transformer, TF-DDRL demonstrates a significant acceleration in convergence speed, reaching the experiment's optimal solution in around 50 iterations. When both PER and Transformer are used concurrently, TF-DDRL converges in approximately 40 iterations.

These results clearly highlight the advantages of integrating both PER and Transformer within TF-DDRL. While incorporating the Transformer could potentially introduce challenges such as instability and slower convergence due to increased model complexity, the V-trace correction mechanism ensures stable learning during the distributed training process. Additionally, PER further accelerates learning by prioritizing important experiences. This synergistic combination enables TF-DDRL to find better scheduling solutions more efficiently compared to native IMPALA.

Cost vs Policy Update Analysis

This experiment analyzes the performance of TF-DDRL in various iterations during policy updates. For training purposes, we utilize four applications as detailed in Section 4.5.1, configuring the resolution as 480. The results, showing the policy cost versus updating iteration, are presented in Fig. 4.6.

As shown in Fig. 4.6, the optimization costs of all techniques decrease with the in-

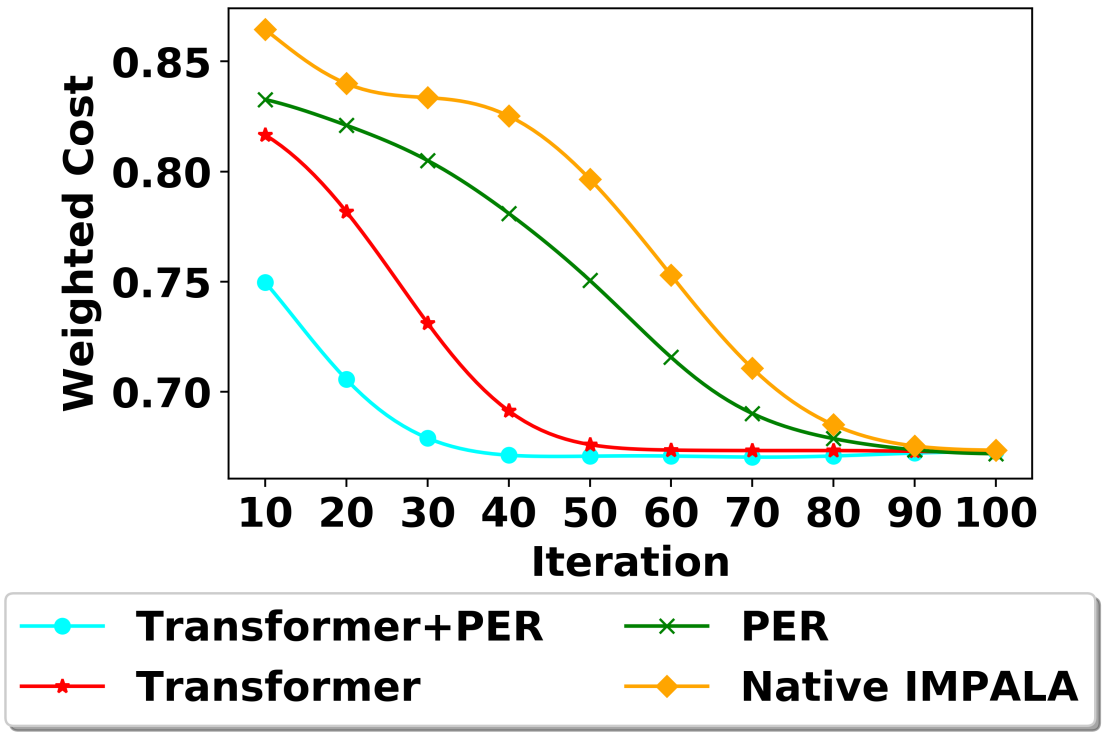
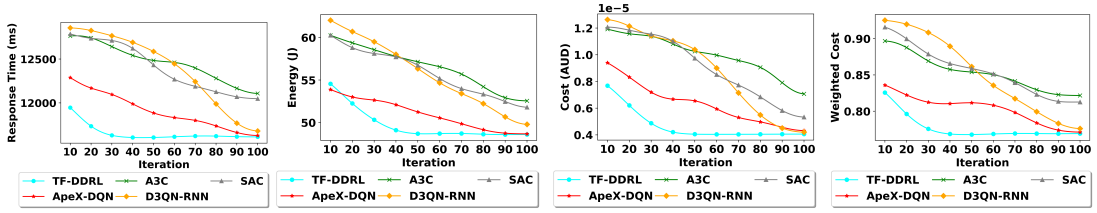


Figure 4.5: PER and Transformer analysis

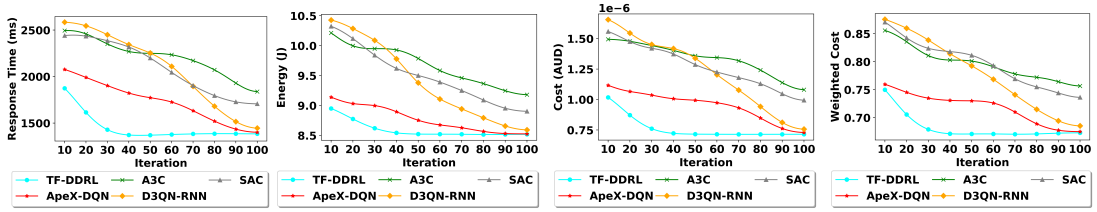
creasing number of iterations in different scenarios. However, under different optimization objectives, TF-DDRL shows a faster convergence compared to other techniques. It converges to the best scheduling solution discovered during training in approximately 40 iterations. ApeX-DQN exhibits a slower convergence speed than TF-DDRL but eventually converges to the optimal scheduling solution in 90 iterations. D3QN-RNN converges to the best scheduling solution under monetary cost optimization (Fig. 4.6c). Although the costs of A3C and SAC decrease continuously during training, neither of them converges to the optimal solution within 100 iterations.

During evaluation, the resolution is adjusted to 240, altering the IoT application's demands for computing resources compared to the training phase. The results, showing the optimization cost versus the policy update for various algorithms, are presented in Fig. 4.7. It is obvious that similar to the results obtained during the training phase, compared with other techniques, TF-DDRL demonstrates better performance in response time, energy consumption, monetary cost, and weighted cost in the evaluation phase.



(a) Response time **(b)** Energy Consumption **(c)** Monetary Cost **(d)** Weighted cost

Figure 4.6: Cost vs policy update analysis - training phase



(a) Response time **(b)** Energy Consumption **(c)** Monetary Cost **(d)** Weighted cost

Figure 4.7: Cost vs policy update analysis - evaluation phase

Also, after 100 iterations of updates for all baseline techniques, none of them can achieve results superior to TF-DDRL. This indicates that TF-DDRL not only converges faster, with significantly less time compared to other techniques but also provides better scheduling results. Except for the ApeX-DQN technique, A3C, D3QN-RNN, and SAC do not converge to the optimal scheduling solution in 100 iterations. Overall, compared to ApeX-DQN results, which is the only baseline technique that converges to the optimal scheduling solution found in the evaluation phase across all optimization objectives, TF-DDRL achieves average performance gains of 60%, 51%, 56%, and 58% in response time, energy consumption, monetary cost, and weighted cost, respectively.

These performance advantages of TF-DDRL can be attributed to several key technical designs. The Transformer, with its multi-head attention mechanism and position-wise MLP layers, effectively captures complex dependencies between state features and provides strong non-linear modeling capabilities, enabling better generalization to different resource configurations. The PER mechanism enables more efficient experience sampling by prioritizing informative experiences based on TD errors, significantly reducing exploration costs compared to uniform sampling methods used in baseline ap-

proaches. Additionally, the combination of distributed experience sharing and V-trace off-policy correction ensures efficient utilization of collected experiences while maintaining training stability, addressing the limitations of techniques like A3C that rely on local experiences. These design elements collectively contribute to TF-DDRL's superior convergence speed and scheduling performance in dynamic edge and cloud environments.

Scalability Analysis

This experiment investigates the impact of different numbers of servers on the scheduling technique for IoT applications. The number of available servers directly impacts the complexity of IoT application scheduling problems, as a higher number of servers leads to a larger action space. To evaluate the scalability performance of TF-DDRL, the experiment uses varying numbers of servers (e.g., 5, 10, 15, 20, 25, 30). Also, other settings are consistent with Section 4.5.3. Due to space constraints and the fact that the results for response time, energy consumption, and monetary cost follow the same patterns as weighted costs, only the results for weighted costs are presented.

Figure 4.8 shows the weighted cost optimization results obtained by various techniques after 100 iterations, considering the growth of candidate servers. As the number of servers increases, TF-DDRL consistently outperforms other techniques, converging more rapidly towards superior solutions. This shows that as the system scales up, TF-DDRL demonstrates superior scalability, enabling it to make more effective application scheduling decisions in fewer iteration cycles. In the baseline techniques, ApeX-DQN outperforms other techniques, although weighted costs eventually continue to increase with the growth of available servers.

The superior scalability of TF-DDRL can be attributed to its architectural advantages in handling large-scale environments. The Transformer's self-attention mechanism efficiently processes the increasing state space by dynamically focusing on relevant server features, while its position-wise MLP provides the necessary modeling capacity for complex server relationships. Moreover, the distributed experience collection combined with PER ensures efficient exploration of the expanded action space, as it prioritizes

experiences that are most informative for learning optimal scheduling policies. These design elements allow TF-DDRL to maintain its performance even as the server count increases, whereas baseline methods struggle with the exponentially growing state-action space.

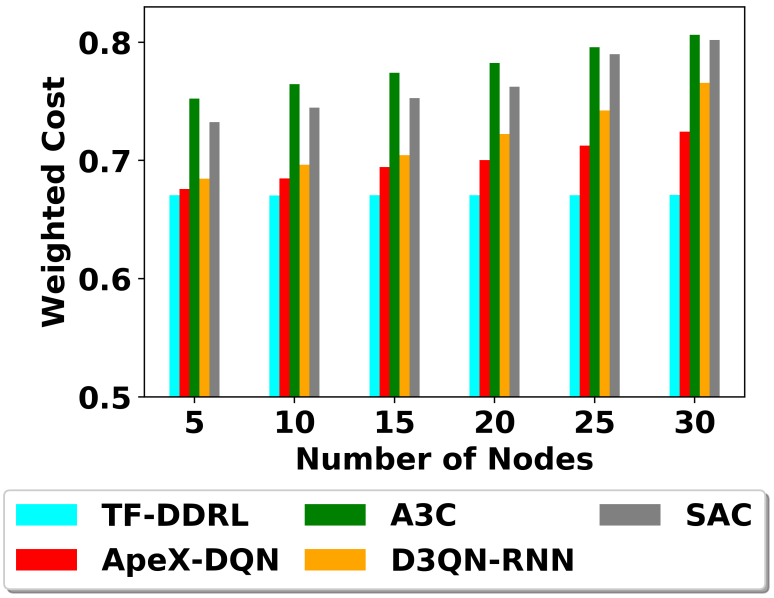


Figure 4.8: Scalability analysis

Greenhouse Gas Emission Analysis

This experiment examines the impact of various scheduling techniques based on Greenhouse Gas Emission (GHE). We specifically analyze electricity generation patterns in Australia[6], the US[7], and Germany[8], considering the associated GHE of various sources involved in electricity production[9]. The total GHE is defined as the sum of GHE from the production of electricity from each source [247], shown below:

$$GHE = EC * \sum_{i \in Sources} (U_i * P_i), \tag{4.46}$$

[6]https://www.energy.gov.au/data/electricity-generation
[7]https://www.eia.gov/tools/faqs
[8]https://www.umweltbundesamt.de/themen/co2-emissionen-pro-kilowattstunde-strom-stiegen-in
[9]https://world-nuclear.org/information-library/energy-and-the-environment/carbon-dioxide-emissions-from-electricity.aspx

where EC represents the total electricity consumed, U_i represents the amount of greenhouse gas emitted per unit of electricity produced using source i, and P_i represents the proportion of the source i in producing electricity.

Figure 4.9 presents the GHE associated with different scheduling techniques based on electricity generation in different countries. Notably, TF-DDRL exhibits the lowest GHE, while A3C has more GHE compared to other techniques. Also, the GHE based on the US power generation pattern is substantially higher than that of Australia and Germany. This discrepancy is due to the prevalence of fossil sources, including coal and natural gas, in the US electricity generation pattern. The experiment results show that TF-DDRL can effectively reduce GHE, which can contribute to collective efforts to address climate change, alleviate the impacts of global warming, and foster a healthier and more sustainable natural environment.

TF-DDRL achieves lower GHE through several key technical advantages in its scheduling decisions. The Transformer architecture enables more precise modeling of the relationship between server power consumption patterns and application characteristics, leading to more energy-efficient task scheduling. Additionally, the PER mechanism helps identify and prioritize experiences that lead to energy-saving scheduling strategies. The distributed learning framework further allows TF-DDRL to explore and learn from a diverse range of energy-efficient scheduling patterns. These capabilities result in more intelligent resource utilization and reduced energy waste, thereby effectively decreasing GHE across different power generation patterns.

Speedup Analysis

With the similar experimental configuration outlined in Section 4.5.3, we explore the speedup performance of various techniques. We define the reference time, denoted as $Time_r$, as the time required for the weighted cost of TF-DDRL with an Actor to reach a value of 0.76. Designating 0.76 as the reference weighted cost is motivated by the fact that this particular value serves as the smallest weighted cost that can be obtained by all baseline techniques. Additionally, we define $Time_t$ as the time required by each technique to attain the reference weighted cost. So, the speedup SPU for each technique

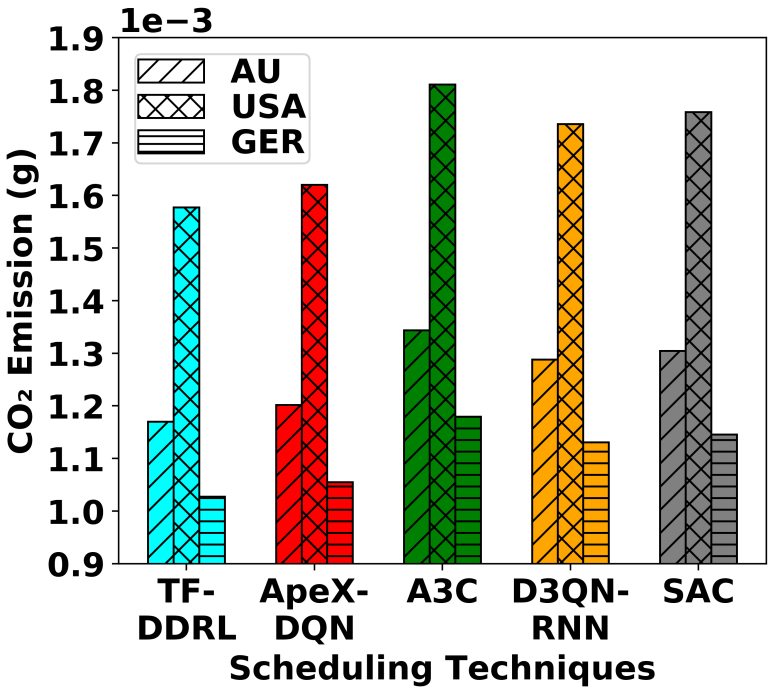


Figure 4.9: GHE analysis

is defined as follows:

$$SPU = \frac{Time_r}{Time_t}. \tag{4.47}$$

The speedup results for all techniques are illustrated in Fig. 4.10. The results demonstrate that TF-DDRL outperforms A3C, D3QN-RNN, and SAC by 7 to 11 times, and it is over 40% faster than ApeX-DQN. This significant speedup advantage can be attributed to several key designs of TF-DDRL. The Transformer's parallelism enables efficient processing of state information. The PER mechanism further accelerates learning by focusing computational resources on the most informative experiences. Moreover, TF-DDRL's V-trace correction method effectively addresses the policy lag between the Actor and Learner, providing more stable and efficient training. These architectural advantages collectively enable TF-DDRL to achieve faster learning and better adaptation to dynamic edge and cloud computing environments.

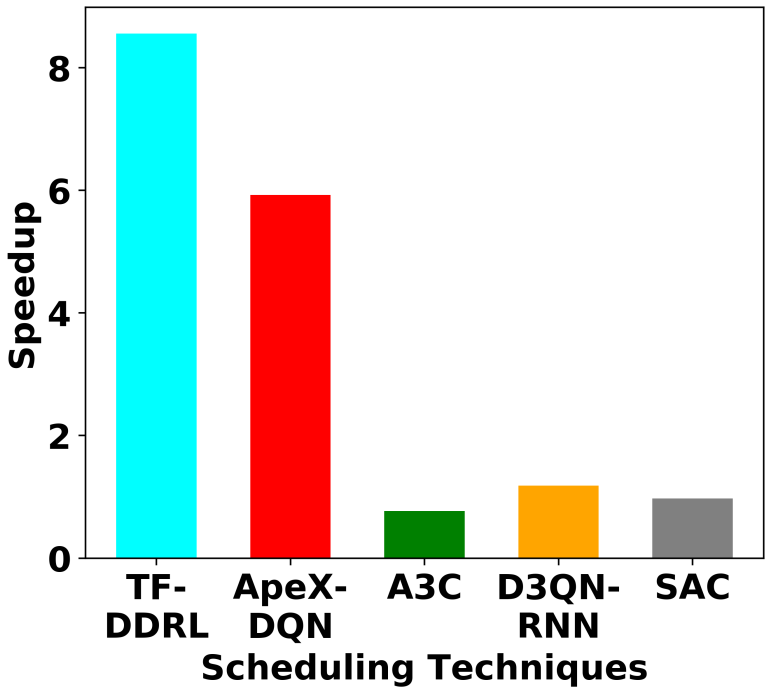


Figure 4.10: Speedup analysis

Scheduling Overhead (SCO) Analysis

This experiment investigates the SCO of each technique. We use the same environment settings in Section 4.5.3. For each technique, we run 100 iterations, each containing four IoT applications. Also, the average SCO is defined as $Time_a = \frac{Time_o}{100}$, where $Time_o$ denotes the total overhead of the technique to schedule the IoT applications.

Figure 4.11 illustrates the $Time_a$ within the 95% confidence interval for various techniques during the scheduling of IoT applications. The scheduling overhead of TF-DDRL is lower compared to ApeX-DQN, D3QN-RNN, and SAC, but higher than A3C. The higher overhead is expected due to the employment of Transformer layers and PER mechanism. However, this trade-off between overhead and performance is well justified by TF-DDRL's significantly better scheduling decisions and faster convergence. Thus, in heterogeneous edge and cloud computing environments, TF-DDRL proves to be more efficient in scheduling IoT applications.

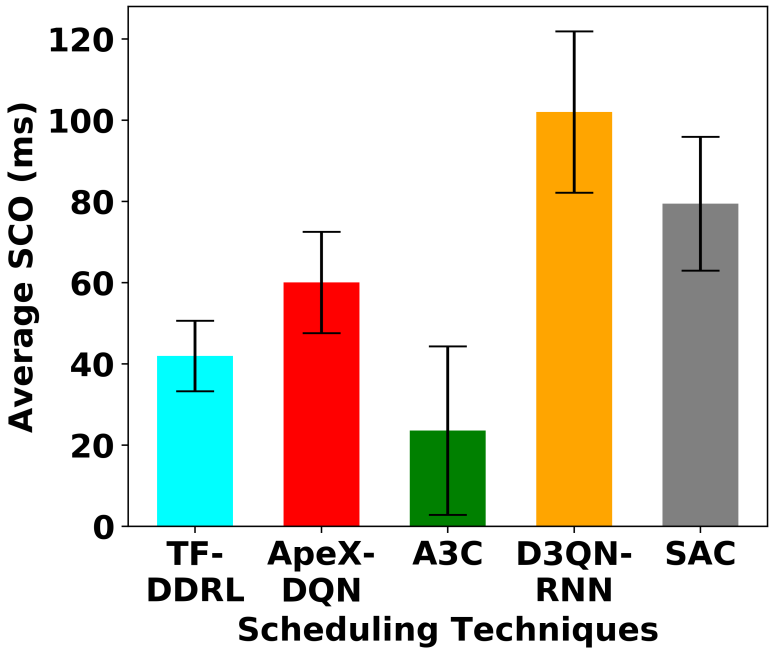


Figure 4.11: SCO analysis

4.6 Summary

In this chapter, we proposed a distributed DRL technique, named TF-DDRL, designed to solve DAG-based IoT application scheduling in highly heterogeneous and dynamic edge and cloud computing environments. We formulated the IoT application scheduling problem as an optimization problem and then transformed it into an MDP model, aiming to minimize response time, energy consumption, monetary cost, and weighted cost. We proposed the TF-DDRL, which follows Actor-Critic architecture, incorporating PER and Transformer techniques to decrease exploration costs and enhance convergence speed. TF-DDRL allows multiple parallel and scalable Actors to work simultaneously and share experience trajectories with the Learner, enabling more effective and efficient learning. Also, we used the V-trace off-policy correction method to solve discrepancies between Learner and Actor policies. As demonstrated by extensive experiments, in highly stochastic and heterogeneous computing environments, TF-DDRL possesses better scalability and adaptability, compared to its counterparts. The results indicate that TF-DDRL outperforms other DRL-based approaches, demonstrating performance

improvements of up to 60%, 51%, 56%, and 58% in terms of response time, energy consumption, monetary cost, and weighted cost, respectively.

Chapter 5

ReinFog: A Deep Reinforcement Learning Empowered Framework for Resource Management in Edge and Cloud Computing Environments

The expanding IoT landscape demands effective server deployment strategies for real-time processing and energy efficiency. However, heterogeneous and dynamic applications complicate these requirements. To address these challenges, we propose ReinFog, a modular distributed framework that leverages Deep Reinforcement Learning (DRL) for adaptive resource management across edge/-fog/cloud environments. ReinFog enables practical development and deployment of both centralized and distributed DRL techniques, supporting native and library-based implementations for diverse IoT scheduling objectives. It provides flexible deployment configurations, allowing customizable placement and scaling of DRL Learners and Workers in large-scale systems. Additionally, we introduce MADCP, a novel Memetic Algorithm for DRL Component Placement that synergizes Genetic Algorithm, Firefly Algorithm, and Particle Swarm Optimization. Experimental results demonstrate substantial performance improvements: 45% reduction in response time, 39% in energy consumption, and 37% in weighted cost, while maintaining minimal scheduling overhead. ReinFog exhibits strong scalability, incurring only 0.3-second additional startup time and approximately 2 MB extra RAM per Worker when scaling. Furthermore, MADCP accelerates DRL convergence rates by up to 38%.

This chapter is derived from:

- **Zhiyu Wang**, Mohammad Goudarzi, and Rajkumar Buyya, "ReinFog: A Deep Reinforcement Learning Empowered Framework for Resource Management in Edge and Cloud Computing Environments", *Journal of Network and Computer Applications (JNCA)*, Volume 242, Article 104250, Elsevier, 2025.

5.1 Introduction

The evolution of computing paradigms has led to the emergence of edge/fog and cloud computing as complementary approaches to address the growing demands of modern applications [13]. Edge/fog computing refers to the deployment of computational resources closer to data sources, such as IoT devices, sensors, and gateways, to reduce latency and improve real-time processing capabilities [248]. This distributed network of nodes performs data processing, storage, and decision-making near the data origin, minimizing the need to send large volumes of data to distant servers. By processing tasks locally or in proximity, edge/fog computing helps to reduce latency and enhance the responsiveness of time-sensitive applications. In contrast, cloud computing offers centralized resources in remote data centers, providing vast storage and processing capabilities ideal for handling large-scale data and complex computations, albeit with potentially higher latency for real-time applications [249].

The synergy between edge/fog and cloud computing creates a powerful infrastructure capable of supporting various applications with varying requirements. This integrated approach allows critical applications to be processed closer to the data source for faster response times, while less time-sensitive and resource-intensive applications can be offloaded to the cloud. Such a hybrid model is particularly crucial in the context of the Internet of Things (IoT), where applications are growing at an unprecedented rate [16]. This has led to a significant increase in data generation and processing demands, necessitating efficient deployment strategies that can effectively leverage the computational capabilities across edge/fog and cloud environments [250].

As IoT applications continue to evolve and expand, they present unique challenges for resource management and scheduling [9]. The heterogeneity of IoT devices and edge/fog and cloud environments, ranging from resource-constrained sensors to powerful cloud servers, is characterized by varying hardware capabilities (e.g., processing power, memory, and energy constraints), network conditions (e.g., latency, bandwidth, and reliability), and application requirements (e.g., real-time processing versus compute-intensive tasks) [251]. This diverse landscape makes rule-based resource scheduling strategies ineffective [30]. Moreover, the dynamic nature of IoT workloads

and network conditions requires adaptive resource management solutions capable of responding to rapid and unpredictable changes [252]. The exponential growth in the number of IoT devices and applications further compounds these issues, demanding highly scalable management approaches. Additionally, IoT applications often have conflicting requirements, such as minimizing latency while maximizing energy efficiency [253], necessitating complex multi-objective optimization strategies.

Traditional heuristic or rule-based approaches to resource management often rely on static optimization or pre-defined rules, which can be effective in predictable environments but fall short in adapting to the rapidly changing conditions of IoT ecosystems [19]. These methods struggle to optimize multiple objectives such as minimizing response time and reducing energy consumption [26], particularly as the complexity of decision-making increases exponentially with the scale of the system [254]. For instance, in real-time video processing applications deployed across edge/fog and cloud nodes, the system must continuously adapt to fluctuating bandwidth, latency, and processing power [176]. Heuristic methods may struggle to balance these dynamic trade-offs, often resulting in suboptimal resource scheduling and increased system latency. In contrast, Deep Reinforcement Learning (DRL) techniques offer a more adaptive and scalable solution by continuously learning and optimizing resource management decisions based on real-time feedback from the environment [255]. This enables DRL-based approaches to dynamically schedule IoT applications, predict future workloads, and efficiently utilize available resources, outperforming traditional methods in complex and unpredictable IoT environments.

While DRL techniques demonstrate significant potential in addressing dynamic resource management challenges, to the best of our knowledge, there is currently no framework that comprehensively integrates both centralized and distributed DRL techniques for IoT application scheduling in edge/fog and cloud computing environments. Existing frameworks primarily rely on rule-based and heuristic methods. This critical gap is reflected in two key aspects. First, current solutions do not provide mechanisms to simultaneously accommodate centralized and distributed DRL techniques, which is essential for efficient resource management across dynamic and stochastic edge/fog and cloud environments. Second, existing frameworks cannot support both native DRL tech-

nique implementations and external DRL library integrations, limiting the flexibility and adaptability of DRL-based solutions in heterogeneous computing environments. These limitations highlight the urgent need for a unified framework that can effectively leverage various DRL techniques for IoT application scheduling in edge/fog and cloud computing environments.

To address these challenges, we propose ReinFog, a novel framework that harnesses the power of DRL for adaptive resource management in edge/fog and cloud computing environments. To the best of our knowledge, ReinFog is the first framework that comprehensively integrates mechanisms for the integration of both centralized and distributed DRL techniques for IoT application scheduling, while supporting both native DRL implementations and external DRL library integrations through a modular and extensible design. To address the dynamic nature of IoT ecosystems, ReinFog incorporates multiple DRL techniques, both centralized and distributed, to adapt to rapidly changing workloads and network conditions, enabling real-time, intelligent IoT application scheduling decisions that optimize multiple objectives simultaneously. ReinFog supports native DRL technique implementations, enabling researchers to design and develop centralized and distributed DRL techniques, specifically designed for edge/fog and cloud computing environments. Also, recognizing that each DRL library incorporates a set of specific DRL techniques, ReinFog offers mechanisms to seamlessly integrate external DRL libraries. This dual capability enhances the flexibility and adaptability of DRL-based solutions in heterogeneous computing environments, allowing users to leverage familiar tools and accelerate their development processes. Accordingly, to facilitate the implementation of both native and library-based DRL techniques, ReinFog adopts a modular design that supports easy extension and customization. This design separates DRL components into DRL Workers for environment interaction and DRL Learners for policy optimization. To accommodate the diverse requirements of large-scale distributed systems, ReinFog supports customizable deployment configurations for DRL techniques. This design allows for flexible configuration of DRL Learners and DRL Workers, enabling users to tailor deployments according to specific system architectures and performance needs. Such flexibility is crucial for effectively managing resources and ensuring optimal performance in complex IoT environments with varying

scales and topologies. Moreover, as the efficient execution of different DRL techniques requires interaction among multiple DRL-related components, it is crucial to place these components on appropriate nodes. To optimize the DRL component placement, we propose a novel Memetic Algorithm for DRL Component Placement in ReinFog, named MADCP. MADCP combines the Genetic Algorithm (GA)'s robust exploration capabilities, Firefly Algorithm (FA)'s ability to fine-tune local search, and Particle Swarm Optimization (PSO)'s efficient global optimization to efficiently place DRL components across heterogeneous computing nodes. This algorithm enhances ReinFog's ability to quickly adapt to changing environmental conditions and optimize resource utilization before the start of DRL training processes.

The key contributions of our chapter are as follows:

- We propose ReinFog, a containerized and modular framework for DRL-based resource management in edge/fog and cloud environments. It offers mechanisms to support both centralized and distributed DRL techniques. Also, it enables the integration of both native DRL techniques and external DRL libraries.
- We design customizable deployment configurations for DRL techniques in ReinFog, allowing flexible configuration of DRL Learners and DRL Workers in large-scale distributed systems.
- We propose a novel Memetic Algorithm for DRL Component Placement in ReinFog, named MADCP, combining GA, FA, and PSO for efficient DRL component placement.
- We conduct extensive practical experiments evaluating ReinFog's performance across various aspects. It demonstrates that ReinFog is a lightweight and scalable framework capable of effectively scheduling IoT applications under diverse optimization objectives.

5.2 Related Work

IoT application scheduling and resource management in edge, fog, and cloud environments have attracted significant research attention. Existing approaches can be broadly

categorized into two groups: (i) algorithmic techniques that focus on optimizing scheduling decisions using heuristics, meta-heuristics, or DRL, and (ii) system-level software frameworks that aim to support the practical deployment and management of IoT applications. In this section, we first review algorithmic techniques, including both heuristic/meta-heuristic and DRL-based methods. Then, we summarize relevant software frameworks and highlight their limitations in handling dynamic, large-scale, and heterogeneous environments. Finally, we provide a comparative analysis to clearly position the originality and technical contributions of our proposed ReinFog framework.

5.2.1 Algorithmic techniques for IoT Scheduling

A wide range of algorithmic techniques have been proposed to address the challenges of scheduling and resource management in IoT-enabled edge, fog, and cloud environments. These techniques can be broadly classified into heuristic/meta-heuristic methods and machine learning-based techniques. In terms of heuristic techniques, Wu et al. [183] modeled IoT application scheduling in edge and fog environments as a Directed Acyclic Graph (DAG), using EDA and partitioning to queue IoT applications and assign servers. Ali et al. [186] proposed an NSGA2-based technique for minimizing the total computation time and system cost of IoT application scheduling in heterogeneous fog cloud computing environments. Hoseiny et al. [185] proposed a GA-based technique to minimize computation time and energy consumption in heterogeneous fog-cloud IoT application scheduling. While these heuristic methods perform well in specific scenarios, they often lack adaptability to dynamic environments. In recent years, machine learning techniques, particularly DRL, have gained significant attention for resource management in edge/fog and cloud computing environments, owing to their adaptability and capacity for continuous learning in dynamic scenarios. Huang et al. [191] applied a Deep Q-Network (DQN)-based approach to address resource allocation problems within edge computing environments. Zheng et al. [193] proposed a Soft Actor-Critic (SAC)-based technique to solve the optimization problem of computational offloading and resource allocation in collaborative vehicle networks. Siyadatzadeh et al. [256] proposed ReLIEF, which employs Q-Learning to manage resources in fog-based IoT systems. Wang et al.

[26] proposed DRLIS, which leverages Proximal Policy Optimization (PPO) to optimize system load balancing and response time in edge and fog computing environments. Zou et al. [89] proposed A3C-DO, which utilizes the Asynchronous Advantage Actor-Critic (A3C) technique to manage resources in edge computing environments. Liu et al. [257] proposed an A3C-based approach for edge computing in the smart vehicles domain. Wang et al. [54] proposed TF-DDRL, which is based on Importance Weighted Actor-Learner Architectures (IMPALA) to schedule IoT applications under three optimization objectives. Table 5.1 presents a qualitative analysis of existing techniques proposed for IoT application scheduling. While prior studies focus on DRL-based scheduling, none provide a unified software framework that facilitates the implementation of both centralized and distributed DRL techniques, which is essential for experimental and practical deployment. These limitations are overcome by our proposed ReinFog software framework.

Table 5.1: A qualitative analysis of existing techniques for IoT application scheduling

Work	Architectural Properties				Algorithm Properties			Evaluation
	IoT Device Layer		Edge/Cloud Layer		Main Technique		Multiple Optimization Objectives	
	Real Applications	Request Type	Computing Environment	Heterogeneity				
[183]	◐	Homogeneous	Edge and Cloud	Heterogeneous	Metaheuristic Algorithms	EDA	✓	Simulation
[186]	○	Homogeneous	Edge and Cloud	Heterogeneous		NSGA2	✓	Simulation
[185]	○	Homogeneous	Edge and Cloud	Heterogeneous		GA	×	Simulation
[191]	◐	Heterogeneous	Edge	Homogeneous	Centralized DRL	DQN	✓	Simulation
[193]	◐	Homogeneous	Edge	Homogeneous		SAC	×	Simulation
[256]	◐	Heterogeneous	Edge	Heterogeneous		Q-Learning	✓	Simulation
[26]	●	Heterogeneous	Edge and Cloud	Heterogeneous		PPO	✓	Practical
[89]	◐	Homogeneous	Edge	Heterogeneous	Distributed DRL	A3C	✓	Simulation
[257]	○	Homogeneous	Edge and Cloud	Heterogeneous		A3C	×	Simulation
[54]	●	Heterogeneous	Edge and Cloud	Heterogeneous		IMPALA	✓	Practical
●: Real IoT Application and Deployment, ◐: Simulated IoT Application, ○: Random								

5.2.2 Frameworks for Resource Management

Building upon these techniques, researchers have developed several frameworks for resource management in edge/fog and cloud computing environments. Many of these frameworks employ heuristic or meta-heuristic techniques for making resource management decisions. For instance, Yigitoglu et al. [258] developed Foggy, a container-enabled framework supporting policy and rule-based scheduling of containerized IoT applications with dependent tasks. Similarly, Merlino et al. [259] proposed a framework allow-

ing policy-driven vertical and horizontal task offloading. Yousefpour et al. [260] introduced FogPlan, which employs greedy algorithms to minimize IoT application response time, while Ghosh et al. [261] developed Mobi-IoST, using a probabilistic approach for IoT application scheduling. Deng et al. [44] created FogBus2, introducing a GA-based IoT application scheduling technique, and Pallewatta et al. [262] proposed MicroFog, integrating multiple heuristic algorithms to enhance IoT application scheduling flexibility. In recent years, some frameworks have started to incorporate reinforcement learning techniques. Toosi et al. [263] developed GreenFog, which combines linear programming optimization with the Multi-Armed Bandit approach for energy consumption reduction. Similarly, Nkenyereye et al. [264] proposed CEIF, which adopted Deep Q-Learning for resource management in edge computing environments.

5.2.3 Summary and Technical Comparison with Existing Frameworks

Table 5.2 identifies the main properties of the related frameworks and compares them with ReinFog. Environment Support indicates the computing environments supported by each framework. DRL-Integrated Framework indicates whether the framework is comprehensively integrated with DRL capabilities, encompassing multiple DRL techniques and providing support for extensibility. The DRL Capabilities section is a specific contribution of ReinFog. These capabilities enable more adaptive and intelligent resource management by leveraging DRL to dynamically optimize IoT application scheduling in real time. It is further divided into three sub-categories. Mechanism indicates whether the framework supports the integration of native DRL techniques or importing external libraries. Architecture shows whether the framework supports centralized and distributed DRL techniques. Finally, DRL Component Placement shows whether the framework can automatically optimize the placement of DRL components. The Generic Capabilities section represents general requirements for frameworks in edge/-fog or cloud computing environments. These are commonly expected features that any robust framework should provide to ensure flexibility, adaptability, and ease of integration across various platforms and environments, especially in addressing the challenges posed by heterogeneous systems. It is further divided into five sub-categories. Multi-

platform Support shows whether the framework can operate across diverse heterogeneous hardware and software platforms. Container Support refers to the ability to use containerization technologies. Scalability, Configurability, and Extensibility assess the framework's ability to be scaled, customized, and incorporate new features.

ReinFog offers significant advantages over related frameworks across multiple dimensions. Many existing frameworks rely on traditional heuristic ([258], [259], [260], [261], [262]) or meta-heuristic ([44]) techniques for IoT application scheduling, which often lack adaptability to manage dynamic and complex computing environments. Although some recent frameworks have started incorporating basic and single reinforcement learning techniques, such as Multi-Armed Bandit ([263]) or Deep Q-Learning ([264]), these frameworks do not offer mechanisms for integration/development of centralized and distributed DRL techniques. Accordingly, these frameworks struggle with the implementation of efficient and scalable DRL techniques for large-scale deployments. To the best of our knowledge, ReinFog is the first resource management framework that enables the integration of both centralized and distributed DRL mechanisms for IoT application scheduling across edge/fog and cloud environments. These mechanisms enable the integration/development of a wide range of centralized and distributed techniques such as PPO [40] and IMPALA [93]. Besides, ReinFog offers interfaces for the integration of both native and library-based DRL techniques. Notably, ReinFog introduces DRL component placement, a novel feature to customize and optimize the placement of DRL components using multiple meta-heuristic algorithms and proposed MADCP. These comprehensive features and capabilities make ReinFog a versatile platform for researchers, enabling them to either utilize built-in DRL techniques or extend its mechanisms for various resource management scenarios in edge/fog and cloud computing environments. In terms of generic capabilities, ReinFog also offers multi-platform support, a feature lacking in several frameworks (e.g., [260], [261], [263]). Besides, ReinFog is designed with scalability, configurability, and extensibility in mind, addressing limitations found in many existing frameworks ([258], [259], [260], [261], [263], [264]).

Table 5.2: A qualitative comparison of related software frameworks with ReinFog

Work	Environment Support	DRL-Integrated Framework	DRL Capabilities					Generic Capabilities				
			Mechanism		Architecture		DRL Components Placement	Multi-platform Support	Container Support	Scalability	Configurability	Extensibility
			Native	Library	Centralized	Distributed						
[258]	Edge/Fog, Cloud	×	×	×	×	×	×	✓	✓	✓	×	×
[259]	Edge/Fog, Cloud	×	×	×	×	×	×	✓	✓	△	✓	×
[260]	Edge/Fog, Cloud	×	×	×	×	×	×	×	✓	△	△	✓
[261]	Edge/Fog, Cloud	×	×	×	×	×	×	×	×	×	×	×
[44]	Edge/Fog, Cloud	×	×	×	×	×	×	✓	✓	✓	✓	✓
[262]	Edge/Fog, Cloud	×	×	×	×	×	×	×	✓	✓	✓	△
[263]	Edge/Fog	×	×	×	×	×	×	×	✓	✓	△	×
[264]	Edge/Fog	×	×	×	×	×	×	✓	✓	✓	△	×
ReinFog	Edge/Fog, Cloud	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

✓: Fully Supported, △: Partially Supported, ×: Not Supported

5.3 ReinFog Framework Architecture

This section introduces the ReinFog framework, outlining its hardware environment and software architecture. We propose a multi-layered structure that supports heterogeneous IoT, edge/fog, and cloud environments, and detail the overall architecture of our framework.

5.3.1 Hardware Environment

ReinFog is designed to operate across a heterogeneous multi-layered hardware environment, as illustrated in Fig. 5.1. This environment encompasses three primary layers: Cloud, Edge/Fog, and IoT, each with distinct characteristics and roles in the overall system.

Cloud Layer

The Cloud Layer represents the highest tier of computing resources in the ReinFog hardware environment. It consists of high-performance servers provided by different cloud service providers such as Amazon Web Services (AWS), Microsoft Azure, and Nectar. These cloud environments offer scalable computing power and storage capabilities, high reliability and availability, and advanced services for data analytics and machine learn-

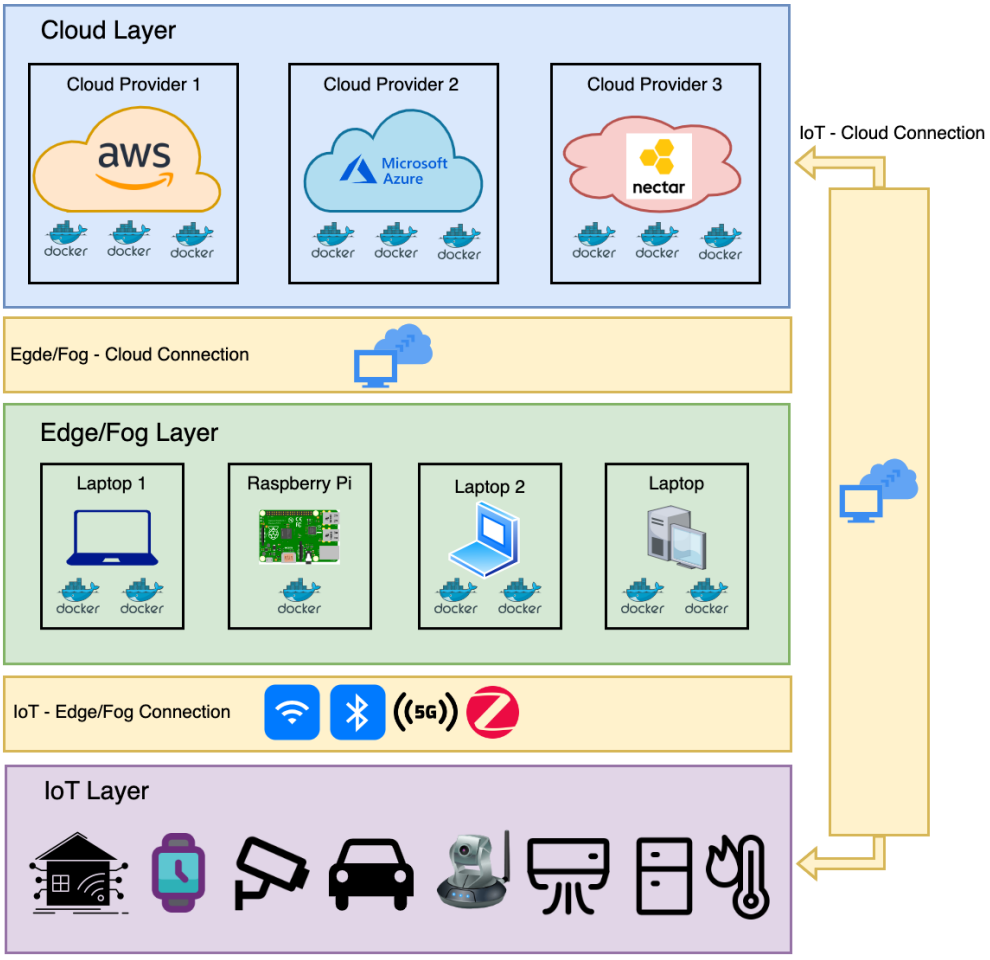


Figure 5.1: Heterogeneous multi-layered hardware environment for ReinFog

ing. The cloud layer supports containerization for consistent deployment of ReinFog components and typically handles computationally intensive tasks, large-scale data processing, and long-term data storage.

Edge/Fog Layer

The Edge/Fog Layer serves as an intermediate computing tier between the cloud and IoT devices. This layer comprises various computing devices, including laptops, desktops with varying computational capabilities, and single-board computers (e.g., Raspberry Pi). These devices are strategically positioned closer to the data source, enabling reduced latency for time-sensitive applications, local data processing and filtering, and

improved privacy and security by keeping sensitive data local. The Edge/Fog layer connects to the Cloud layer via Internet connections, allowing for seamless data transfer and task offloading when needed. It also supports containerization for flexible deployment of ReinFog software components, playing a crucial role in facilitating real-time applications and reducing the computational burden on both the cloud and IoT devices.

IoT Layer

The IoT Layer comprises the various end devices and sensors that collect data and interact with the physical environment. This layer includes smart home devices (e.g., thermostats, security cameras), wearable devices, industrial sensors and actuators, connected vehicles, and environmental monitoring sensors. These devices are characterized by limited computational resources and power constraints, with direct interaction with the physical world through sensing and actuation. For connectivity, these IoT devices employ various short-range communication protocols (e.g., WiFi, Bluetooth, Zigbee, 5G) to connect with the Edge/Fog layer, while Internet-based connections facilitate communication with the Cloud layer. The IoT layer is the primary source of data in the ReinFog ecosystem, driving the need for efficient resource management and IoT application scheduling. We assume each IoT application in this layer can consist of one or multiple interdependent IoT tasks that need to be efficiently scheduled.

5.3.2 Software Architecture

The software architecture of ReinFog is designed to efficiently manage and schedule IoT tasks in heterogeneous edge/fog and cloud environments. As illustrated in Figure 5.2, the framework consists of two primary subsystems, along with user interfaces for IoT application submission.

Task Execution Subsystem

This subsystem forms the operational core of ReinFog, which is responsible for managing the execution of IoT applications across the distributed environment. It comprises:

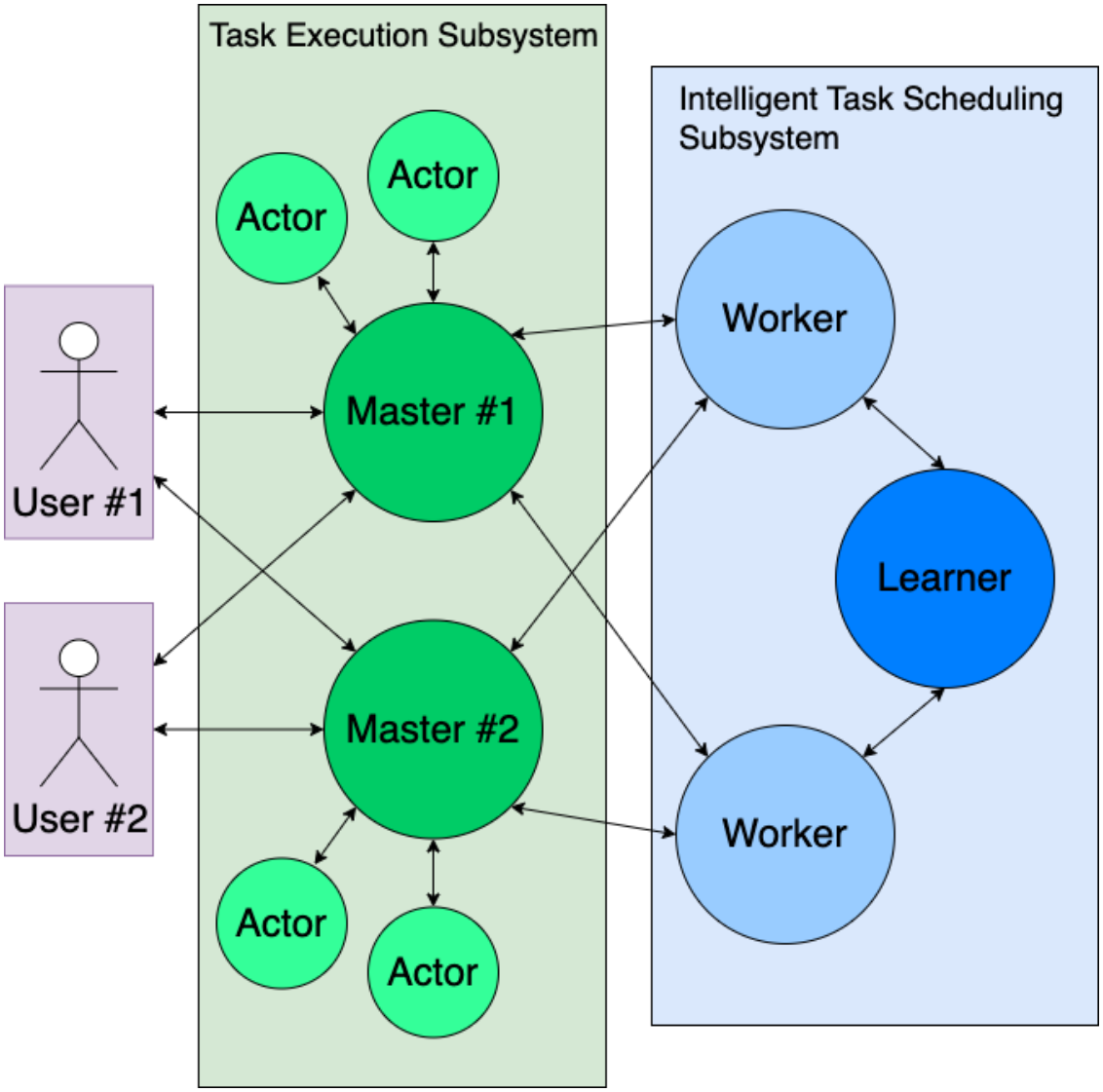


Figure 5.2: High-level software architecture of ReinFog

- **Master:** Acting as the central coordinator, the Master manages overall system operations. It receives IoT application requests, analyzes task dependencies, and constructs directed acyclic graphs (DAGs) to represent application structures. The Master interacts with the Intelligent Task Scheduling Subsystem for optimal scheduling decisions, considering both task dependencies and resource availability. It oversees execution, ensuring correct task order. The architecture supports multiple Master instances for scalability and fault tolerance.

- **Actor:** Distributed across various nodes in the system, Actors are responsible for executing assigned tasks and managing local resources. They implement the actual task execution, monitor local system performance, and handle data transfer between tasks when necessary. Actors continuously report resource utilization, task progress, and completion status back to their associated Master. This real-time feedback enables dynamic resource management and adaptive IoT task scheduling, allowing the system to respond efficiently to changing workloads and environmental conditions.

The Task Execution Subsystem handles critical functions such as application analysis, task distribution, execution monitoring, and result aggregation, ensuring efficient utilization of available resources across the heterogeneous computing environment.

Intelligent Task Scheduling Subsystem

This subsystem represents ReinFog's core innovation, leveraging DRL to make intelligent IoT scheduling decisions. It consists of:

- **Worker:** Workers gather system state information from the Task Execution Subsystem and generate IoT task scheduling decisions based on the learned policy. They communicate these decisions back to the Task Execution Subsystem for implementation. Workers continuously refine their decision-making process, adapting to the system's dynamic nature through distributed learning.
- **Learner:** Centrally located, the Learner aggregates experiences or gradients from Workers and optimizes the global DRL technique. The Learner periodically distributes the refined policy to all Workers, ensuring system-wide improvement in decision-making. This continuous optimization enables ReinFog to adapt to changing conditions and enhance overall IoT task scheduling efficiency.

The Intelligent Task Scheduling Subsystem continuously learns from the system's performance, adapting to changing conditions over time. This adaptive approach allows ReinFog to efficiently handle the dynamic nature of IoT workloads and the heterogeneity of computing resources.

ReinFog Operation Workflow

In the ReinFog framework, the workflow begins when users submit IoT applications through dedicated interfaces. These applications typically comprise multiple interdependent tasks that require coordinated execution. Upon receiving user submissions, the Master in the Task Execution Subsystem serves as the first point of contact, processing the applications by analyzing task dependencies and constructing DAGs to represent the execution structure. The Master then forwards scheduling requests to the Worker in the Intelligent Task Scheduling Subsystem, along with the system states and task characteristics. Based on this real-time information, the Worker generates IoT task scheduling decisions using DRL policies and sends these decisions back to the Master. Upon receiving these decisions, the Master coordinates its managed Actors to execute the specific tasks, with Actors handling the actual task execution and providing status feedback. Throughout this process, the Learner in the Intelligent Task Scheduling Subsystem works in parallel to optimize the DRL policies globally based on execution results and system states, continuously distributing improved policies to all Workers to enhance the system's scheduling efficiency and adaptability.

This architecture allows ReinFog to balance the load across available resources and optimize overall system performance. By integrating DRL-based decision-making with traditional task execution mechanisms, ReinFog can effectively adapt to the complex and dynamic nature of modern IoT, edge/fog and cloud computing environments, providing a robust and efficient solution for resource management and IoT application scheduling.

5.4 ReinFog Design and Implementation

To achieve adaptive resource management in edge/fog and cloud environments, we implement the ReinFog framework by extending the core components of the FogBus2 framework and implementing and integrating the new Intelligent Task Scheduling Subsystem. We chose FogBus2 as the foundation for ReinFog because it already implements many of the basic functionalities required in our Task Execution Subsystem.

FogBus2 is a lightweight and distributed container-based framework for integrating heterogeneous IoT systems with edge/fog and cloud environments. It comprises five main components that align well with our Task Execution Subsystem requirements:

- **User:** handles environmental data collection and actuation control, similar to our user interface for IoT application submission.
- **Master:** manages IoT applications and scheduling, which forms the basis of our Master component in the Task Execution Subsystem.
- **Actor:** performs host resource profiling, aligning with our Actor component's responsibilities.
- **Task Executor:** executes submitted IoT applications, fitting directly into our execution model.
- **Remote Logger:** provides persistent log storage, supporting our system's monitoring and analysis needs.

By leveraging FogBus2, we are able to focus our efforts on developing our DRL-based scheduling subsystem to handle complex, dependency-aware IoT applications. This approach allowed us to build upon a proven foundation while integrating our novel Intelligent Task Scheduling Subsystem with its DRL capabilities. The overall design of ReinFog, showing both the extended FogBus2 components and our new DRL components, is illustrated in Fig. 5.3.

5.4.1 ReinFog DRL Components

In this section, we present the design and organization of our DRL components: the DRL Learner and the DRL Worker. Each component is modularly designed and encompasses several sub-components and modules, which interact through well-defined APIs and internal messages. The overall design is depicted in Fig. 5.4.

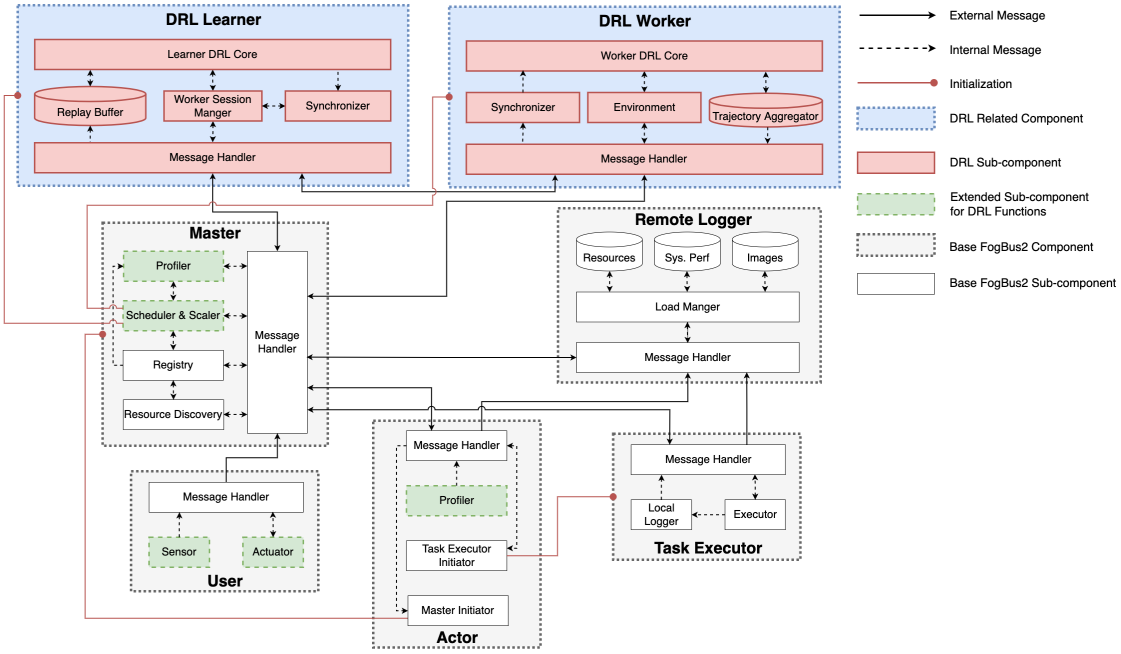


Figure 5.3: ReinFog design overview

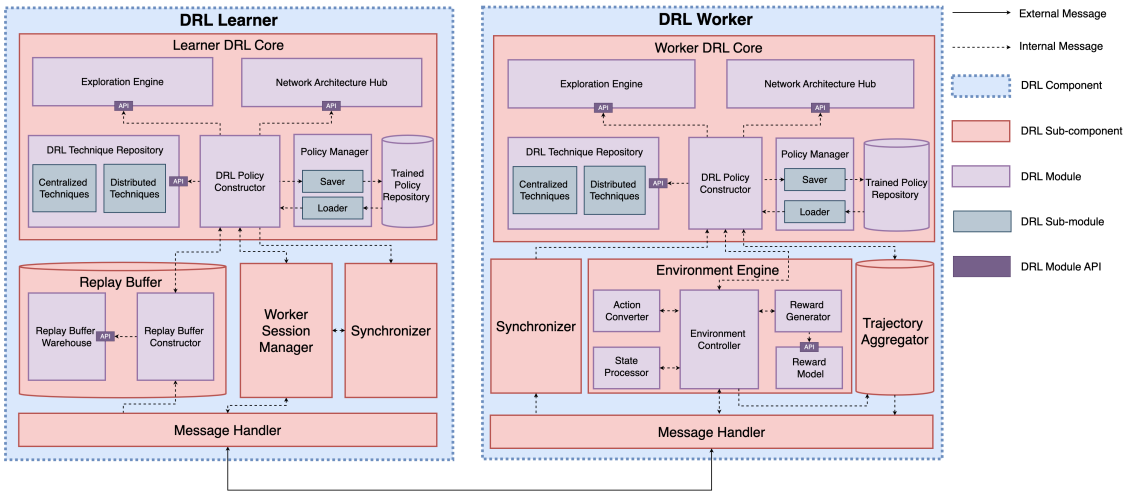


Figure 5.4: Distributed DRL components design

DRL Learner

The DRL Learner component is responsible for training the DRL techniques and managing the learning process. It comprises the following sub-components:

Learner DRL Core: The Learner DRL Core is central to policy learning and optimization. It includes the following modules:

1. **Exploration Engine:** This module implements and manages exploration strategies during the training process to balance exploration and exploitation. It incorporates established methods such as ϵ-greedy and OrnsteinUhlenbeck process noise, while also providing an extensible mechanism for integrating additional exploration algorithms. The Exploration Engine exposes a well-defined API, allowing other components to interact with it efficiently. Through this API, it dynamically interacts with the Network Architecture Hub and the DRL Technique Repository via the DRL Policy Constructor, enabling it to access and utilize current technique states and network architectures. This API-driven design enhances modularity and facilitates easy integration of new exploration strategies.

2. **Network Architecture Hub:** This module manages and maintains base neural networks for various DRL techniques. It supports a wide range of architectures including:

   - Deep Neural Networks (DNNs)
   - Recurrent Neural Networks (RNNs)
   - Long Short-Term Memory Networks (LSTMs)
   - Transformers

   These diverse architectures enable the hub to accommodate different types of tasks. The module exposes APIs for querying and updating network architectures, facilitating seamless integration and dynamic adaptability. Designed with extensibility in mind, it allows for easy incorporation of new neural network architectures and policy optimization methods in the future.

3. **DRL Technique Repository:** This module efficiently manages and maintains various DRL techniques. It is divided into Centralized Techniques and Distributed Techniques sub-modules, accommodating different user requirements and computational architectures. The repository provides two integration mechanisms.

The native integration mechanism enables direct implementation of DRL techniques within the framework, offering optimal performance and full customization flexibility. ReinFog already includes native implementations of several representative DRL techniques, such as DQN, PPO, A3C, and IMPALA. In parallel, the library-based integration mechanism enables seamless incorporation of external DRL libraries through standardized interfaces. ReinFog has already integrated techniques including SAC and R2D2 via the Ray library. This dual-mechanism design allows developers to either build custom DRL techniques within the framework or directly leverage existing implementations, ensuring flexibility, maintainability, and performance consistency across heterogeneous computing environments.

4. **DRL Policy Constructor:** This module is central to building and optimizing DRL policies. It orchestrates interactions with multiple modules through well-defined APIs to create and refine effective DRL strategies. The Constructor leverages the Network Architecture Hub's API for base network construction and accesses the DRL Technique Repository via API for policy development. It interacts with the Exploration Engine through its API to balance exploration and exploitation during the learning process. For policy persistence, the Constructor integrates with the Policy Manager, enabling saving and loading of policies. To enhance learning efficiency, it collaborates with the Replay Buffer for experience sampling. The DRL Policy Constructor also coordinates parallel learning processes via the Worker Session Manager and ensures policy synchronization through the Synchronizer.

5. **Policy Manager:** This module is responsible for comprehensive DRL policy administration. It consists of two key sub-modules: the Saver and the Loader. The Saver sub-module ensures version-controlled, persistent storage of optimized policies in the Trained Policy Repository. The Loader sub-module, on the other hand, retrieves policies from the Repository as needed for further refinement or evaluation. The Policy Manager interfaces directly with the DRL Policy Constructor, enabling seamless integration of policy persistence operations within the overall learning process.

6. **Trained Policy Repository:** This module functions as a centralized storage for optimized and validated DRL policies. It interfaces directly with the Policy Manager's Saver and Loader sub-modules, enabling efficient storage and retrieval of policies. The Repository maintains a structured archive of policies, supporting version control and rapid access for DRL experiments. It enhances the overall efficiency and effectiveness of policy management in the ReinFog framework.

Replay Buffer: The Replay Buffer enhances learning efficiency and stability by storing and reusing experiences. It consists of two key modules:

1. **Replay Buffer Warehouse:** This module stores agent-environment interaction experiences. It features both random and reservoir sampling buffers. Random sampling helps break temporal correlations in the data, while reservoir sampling maintains a fixed-size buffer suitable for streaming or unbounded data. The Warehouse also supports flexible integration of additional sampling algorithms.

2. **Replay Buffer Constructor:** This module manages the creation, maintenance, and updating of buffers. It interfaces with the Replay Buffer Warehouse through its API to facilitate the storage, retrieval, and sampling of experiences. It also interacts directly with the DRL Policy Constructor, supporting efficient experience utilization in the learning process.

Worker Session Manager: The Worker Session Manager is responsible for managing communication across multiple DRL Workers in a distributed environment. It handles thread creation, maintenance, and DRL Worker communication, ensuring fast and reliable transmission of data and instructions. This sub-component plays a crucial role in coordinating parallel learning processes and maintaining efficient distributed operations within the ReinFog framework.

Synchronizer: The Synchronizer is tasked with synchronizing training across the distributed learning environment. It coordinates policy updates, gradients, and parameters between the DRL Learner and DRL Workers, maintaining a unified learning environment. Through close collaboration with the Worker Session Manager, the Synchronizer

ensures consistency in the learning process, facilitating effective distributed learning in ReinFog.

Message Handler: The Message Handler manages inter-component communications within the ReinFog framework. It receives and processes incoming messages, efficiently routing them to the appropriate internal sub-components. It serves as a central communication hub, facilitating effective information exchange between various components of the ReinFog framework. By ensuring smooth and organized message flow, the Message Handler maintains system coherence and optimizes overall operational efficiency.

DRL Worker

The DRL Worker component is responsible for interacting with the environment, collecting and processing data to support policy learning and optimization within the ReinFog framework. It shares core functionalities with the DRL Learner, with the Worker DRL Core mirroring the architecture of the Learner DRL Core to ensure consistent functionality and coordination. Despite these similarities, the DRL Worker incorporates several unique sub-components: an Environment Engine for direct interaction with the learning environment, a Trajectory Aggregator for efficient collection and processing of experience data, and a Synchronizer adapted for DRL Worker-specific synchronization tasks, functioning differently from the Synchronizer in the DRL Learner. In centralized learning scenarios, the DRL Worker can assume the role of the DRL Learner, enabling direct policy updates for DRL techniques. This comprehensive design enables flexible deployment of the DRL Worker in both centralized and distributed learning contexts, enhancing the overall adaptability and efficiency of the ReinFog framework.

Environment Engine: The Environment Engine manages interactions between the learning components and the environment through several key modules:

1. **State Processor:** This module transforms raw environmental data into a format suitable for decision-making and learning processes.

2. **Action Converter:** This module translates generated actions into environment-specific commands, ensuring proper execution of decisions.

3. **Reward Generator:** This module processes reward parameters and utilizes the Reward Model's API to calculate rewards.

4. **Reward Model:** This module defines reward calculation methods, maintains consistency in reward generation, and supports the extension of additional reward functions.

5. **Environment Controller:** This module oversees the overall interaction process, managing the flow of actions and states, and coordinating the integration of other modules.

This modular design ensures a smooth and continuous interaction cycle, facilitating efficient learning and adaptation within the ReinFog framework.

Trajectory Aggregator: The Trajectory Aggregator is responsible for collecting and processing trajectories generated from interactions with the environment. These trajectories, consisting of sequences of states, actions, and rewards, encapsulate experiential data over time. The aggregator maintains a well-structured and accessible repository of these experiences for training and evaluation purposes. In specific techniques such as A3C, where gradients are calculated within DRL Workers, the Trajectory Aggregator collects and transmits these gradients to the DRL Learner for updating the global policy. Moreover, in centralized learning scenarios where policies are optimized within the DRL Worker component, the Trajectory Aggregator can adapt to function as a Replay Buffer, storing and sampling trajectory data to support policy training and updates. This versatile design enables the Trajectory Aggregator to support various learning paradigms and techniques within the ReinFog framework, enhancing flexibility and efficiency in different operational contexts.

Synchronizer: The Synchronizer in the DRL Worker maintains consistency between local and global policies in distributed learning scenarios. It periodically obtains the latest policy parameters from the DRL Learner and updates the local policies accordingly.

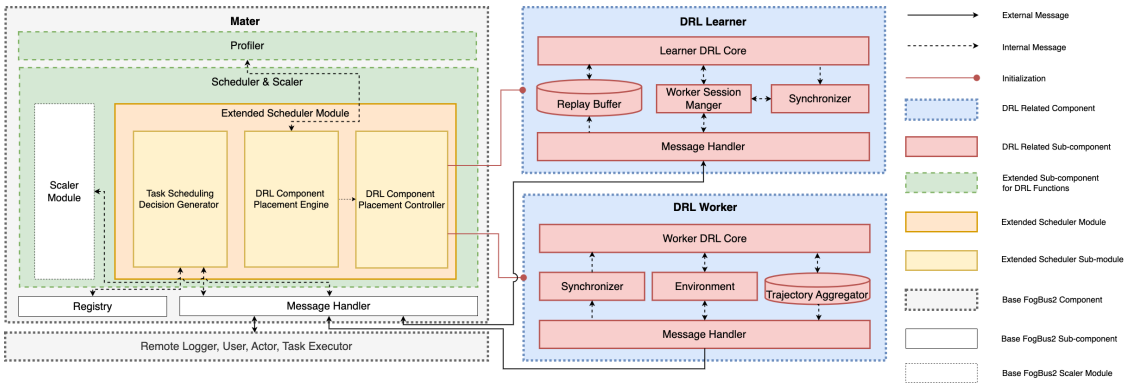


Figure 5.5: Extended FogBus2 scheduler module

This mechanism facilitates efficient knowledge sharing, allowing DRL Workers to make decisions based on up-to-date global knowledge while contributing to the overall learning process. The Synchronizer is essential for balancing local exploration and global exploitation within the ReinFog framework.

5.4.2 Extended FogBus2 Sub-components

To enable FogBus2 to work seamlessly with DRL capabilities, we have extended several of its sub-components. This section introduces these extended sub-components, as illustrated in Fig. 5.3.

Extended Scheduler & Scaler

The Scheduler & Scaler sub-component has been extended to incorporate DRL capabilities, with the primary focus on enhancing the Scheduler Module. This extension enables coordination with the newly introduced DRL components. Fig. 5.5 illustrates the detailed structure of the Extended Scheduler Module within the Scheduler & Scaler sub-component.

DRL Component Placement Engine: The DRL Component Placement Engine generates strategies for placing DRL Learners and DRL Workers. It incorporates several algorithms:

- **GA**: Inspired by natural selection, GA evolves a population of potential solutions over generations.
- **FA**: Based on the flashing behavior of fireflies, FA uses attraction and movement towards brighter solutions.
- **PSO**: Mimicking the social behavior of bird flocking, PSO updates solutions based on personal and global best experiences.
- **MADCP**: Our proposed algorithm that combines GA, FA, and PSO to leverage their respective strengths for more efficient DRL component placement.

In addition, the Engine supports the integration of additional algorithms to address diverse DRL component placement requirements and optimization objectives.

DRL Component Placement Controller: The DRL Component Placement Controller retrieves placement strategies from the DRL Component Placement Engine and applies them to place DRL Learners and DRL Workers. This sub-module ensures that the DRL components are properly set up according to the generated strategies, facilitating the efficient start of the learning process.

Task Scheduling Decision Generator: The Task Scheduling Decision Generator communicates with DRL components to obtain IoT task scheduling decisions. It then manages the deployment of these tasks based on the received decisions. This sub-module bridges the gap between DRL components and practical task execution within the ReinFog framework.

Extended Profiler

The Profiler has been extended in both the Master and Actor components to provide more comprehensive system monitoring. In the Master, the extended Profiler now collects and analyzes system-wide performance metrics, providing a holistic view of the entire distributed environment. This includes monitoring resource utilization patterns, network latencies, and overall system throughput. In the Actor, the Profiler has been

augmented to gather more detailed node-specific information, such as CPU and memory usage, task execution times, and local network conditions. These enhancements enable more accurate modeling of the system state, which is crucial for the DRL components to make informed IoT task scheduling decisions.

Extended User's Sensor and Actuator

The User component's Sensor and Actuator have been extended to enhance data collection and performance monitoring capabilities, supporting the integration of DRL in ReinFog. These extensions focus on gathering more detailed information about IoT device operations and application performance. Key enhancements include the ability to monitor detailed energy consumption patterns of IoT devices, fine-grained operational timing such as task execution and completion times, as well as device-specific characteristics like processing capabilities and storage capacity. Additionally, the extensions provide real-time data on network conditions and connectivity status of IoT devices.

5.4.3 Centralized and Distributed Deployment

ReinFog supports both centralized and distributed deployment for DRL techniques. This section details how ReinFog components are utilized in these two deployment patterns, with specific examples to illustrate the implementation of each deployment.

Centralized DRL Techniques Deployment

Centralized DRL techniques deployment is essential in ReinFog for scenarios where the environment complexity is moderate and the system can benefit from simplified architecture and reduced communication overhead. In centralized deployment, a single DRL Worker handles both learning and decision-making processes, suitable for scenarios with relatively stable workload patterns or when system resources are limited.

Centralized DRL Deployment Scheme in ReinFog: In centralized deployment, the learning process is conducted directly within the DRL Worker, eliminating the need for

a separate DRL Learner. The Environment Engine collects environmental data from the ReinFog Master and processes them for training. The Trajectory Aggregator will assume the role of the Replay Buffer, which stores and samples experiences for policy optimization. The Worker DRL Core contains all the necessary modules for both learning and decision-making. This design centralizes all environmental data collection and policy updating within the DRL Worker, creating a self-contained unit that operates independently.

Sample Illustration: To demonstrate the centralized deployment, we utilize DQN [72] as an example. DQN is a widely applied DRL technique that combines Q-learning with DNNs, employing a primary network for action selection and a separate target network for value estimation to stabilize training and improve learning efficiency in sequential decision-making problems [265]. In our implementation, the DRL Worker's Environment Engine receives environmental data from the ReinFog Master as a scheduling request and processes them into suitable formats for DQN utilization. The Worker DRL Core maintains two neural networks: the primary network for action selection and the target network for stable Q-value estimation. Actions are selected based on the current Q-values and sent to the Master for IoT task scheduling. The scheduling results and rewards are processed by the Environment Engine and stored in the Trajectory Aggregator. The Worker DRL Core then samples mini-batches of experiences directly from the Trajectory Aggregator and performs Q-learning updates.

Distributed DRL Techniques Deployment

Distributed DRL techniques deployment is essential in ReinFog for addressing complex, large-scale environments where centralized approaches may be inefficient. This deployment pattern can harness the computational capabilities of heterogeneous edge/fog and cloud resources through multiple DRL Workers collaborating with a single or multiple DRL Learners, enabling parallel training and distributed learning to deal with dynamic workloads that demand rapid adaptation.

Distributed DRL Deployment Scheme in ReinFog: In distributed deployment, ReinFog employs multiple DRL Workers collaborating with a single or multiple DRL Learners to enable parallel training and decision-making. Each DRL Worker operates independently, with its Environment Engine collecting and processing environmental data from the Master and its DRL Core generating local decisions based on the current policy. The Trajectory Aggregator in each DRL Worker collects and processes local experiences for transmission. The Worker Session Manager coordinates the communication flow, enabling DRL Workers to transmit their processed experiences to the central DRL Learner. The DRL Learner stores these experiences in its Replay Buffer. The Learner DRL Core then performs global policy optimization using experiences stored and sampled from the Replay Buffer. After policy updates, the Synchronizer ensures policy consistency by distributing the latest global policy to all DRL Workers. This distributed architecture enables scalable learning across multiple nodes while maintaining policy coherence.

Sample Illustration: To illustrate the distributed deployment, we present IMPALA as an example. IMPALA is a highly scalable distributed DRL technique that efficiently handles large-scale learning by employing multiple actors running in parallel with a centralized learner, using importance sampling and V-trace off-policy correction to maintain stability in the learning process. In our implementation of IMPALA, multiple DRL Workers operate in parallel, each with its Environment Engine processing environmental data received from the ReinFog Master. The Worker DRL Core maintains local copies of the actor and critic networks for policy representation. Based on these local networks, the Worker DRL Core generates actions for IoT task scheduling. The scheduling results and rewards are processed locally by the Environment Engine, and each DRL Worker's Trajectory Aggregator collects these experiences. The collected experiences are then transmitted to the DRL Learner through worker sessions managed by the Worker Session Manager, and stored in the Replay Buffer. The Learner DRL Core maintains the global actor and critic networks for policy optimization. It samples experiences from the Replay Buffer, computes importance sampling ratios to address the discrepancy between behavior and target policies, and employs V-trace for off-policy correction to update these global networks. The updated global policy parameters are then dis-

tributed to all DRL Workers through the Synchronizer, ensuring consistency across the distributed system.

5.4.4 Native and Library-based Integration

ReinFog supports two primary mechanisms for integrating DRL techniques: native integration and library-based integration. The native integration mechanism enables direct implementation of DRL techniques within the framework, offering optimal performance and customization capabilities. The library-based integration mechanism allows seamless incorporation of external DRL libraries, providing access to established implementations while maintaining framework compatibility. This dual integration approach ensures that ReinFog can accommodate diverse research needs while leveraging existing DRL solutions. In this section, we detail both integration mechanisms and provide examples of their implementation within ReinFog.

Native DRL Techniques Integration

The native integration mechanism is essential to ReinFog as it offers substantial advantages in performance, maintainability, and flexibility. Directly implementing DRL techniques within the framework can optimize performance, particularly in dynamic edge/-fog and cloud environments where real-time responsiveness is crucial. This mechanism also minimizes dependencies on external libraries, enhancing stability and compatibility across various deployment scenarios. Furthermore, native integration provides researchers with the flexibility to implement and integrate their own custom DRL techniques, allowing for tailored solutions to specific research problems.

Native Integration Mechanism: To support native integration of DRL techniques, ReinFog provides a comprehensive framework through well-defined interfaces and modular design, to offer maximum flexibility while maintaining consistency and efficiency. The Network Architecture Hub is responsible for neural network architecture definition and management. It maintains a collection of base network architectures (e.g., DNNs, RNNs, LSTMs, Transformers) and allows researchers to define custom architec-

tures through a consistent interface. Building upon these network architectures, the DRL Technique Repository is responsible for managing DRL techniques, providing an extensible interface through a base class. This base class defines essential abstract methods including policy network initialization, action selection, loss computation, and parameter updates. These abstractions enable researchers to implement new DRL techniques. ReinFog's Exploration Engine supports native integration through a pluggable exploration strategy interface, offering built-in configurability for common strategies (e.g., ϵ-greedy and OrnsteinUhlenbeck process noise) while enabling custom strategy implementation. The Replay Buffer enables custom buffer implementation through a standard interface. It provides several built-in buffer types (e.g., random sampling and reservoir sampling) while allowing researchers to define flexible sampling strategies and integrate specialized buffer types.

Sample Illustration: To demonstrate the native integration mechanism, we describe how to implement a customized version of IMPALA that leverages ReinFog's advanced features. The Network Architecture Hub is utilized to construct two Transformer-based neural networks. The actor network processes the input states and outputs action probabilities for IoT task scheduling decisions, while the critic network evaluates these states to guide the learning process. IMPALA's core algorithm is implemented by extending the base class in the DRL Technique Repository, where the V-trace off-policy correction and importance sampling calculations are defined. The Exploration Engine is configured to use Ornstein-Uhlenbeck noise for action exploration. The Replay Buffer is set up with reservoir sampling to efficiently store and manage experiences. Through these configurations and implementations, IMPALA operates as a fully functional distributed DRL technique within the ReinFog framework.

Library-based DRL Techniques Integration

The library-based integration mechanism serves as another crucial feature of ReinFog, offering development efficiency, algorithmic diversity, and research flexibility. By importing well-established DRL libraries, researchers can leverage proven algorithms without reimplementation, significantly reducing development time while ensuring reliable

performance. This mechanism also enriches ReinFog with diverse DRL techniques, enabling it to handle various scheduling scenarios in heterogeneous edge/fog and cloud environments. Moreover, it empowers researchers to integrate and experiment with different DRL libraries based on their specific research requirements.

Library-based Integration Mechanism: To support external DRL libraries integration, ReinFog provides a standardized interface through the DRL Technique Repository. This interface defines essential methods for bridging external libraries with ReinFog's environment. Specifically, the interface allows researchers to implement state preprocessing to convert ReinFog's system state representations into formats compatible with external libraries, action translation to map library-generated actions back to ReinFog's scheduling decisions, and reward signal adaptation to ensure proper learning feedback. The DRL Technique Repository manages these transformations, enabling external DRL techniques to operate seamlessly within ReinFog's resource management framework while maintaining their original implementations.

Sample Illustration: To demonstrate the library-based integration mechanism, we provide an example of integrating Recurrent Replay Distributed DQN (R2D2) [94] from the Ray[1] library. Ray library is chosen for its high-performance distributed computing features and comprehensive DRL techniques suite. R2D2 is a distributed DRL technique that extends DQN by incorporating RNNs and a replay system to handle partial observability and temporal dependencies in sequential decision-making problems. Through the interface provided in the DRL Technique Repository, we implement state preprocessing to convert ReinFog's scheduling environment states (e.g., node resources, IoT task characteristics, and network conditions) into R2D2's required tensor format. The action translation mechanism transforms R2D2s output probabilities into concrete scheduling decisions within ReinFog, effectively guiding the scheduling of IoT tasks across available nodes. For reward signal adaptation, the implementation processes ReinFog's performance metrics (e.g., scheduling result, response time, energy consumption) into a scalar reward value suitable for R2D2's learning process. These implementations enable

[1]https://www.ray.io/

R2D2 to effectively learn and make scheduling decisions within ReinFog while preserving its original recurrent replay-based learning mechanism.

5.5 MADCP: A Memetic Algorithm for DRL Component Placement

In ReinFog, effective DRL-based IoT application scheduling requires careful placement of various components (e.g., DRL Learners and DRL Workers) across different nodes in the heterogeneous computing environment. Poor DRL component placement decisions can lead to increased communication overhead, inefficient resource utilization, and degraded learning performance. To solve this challenge, ReinFog offers a mechanism for DRL component placement. While traditional meta-heuristic algorithms could be applied to this placement problem, they show specific limitations. GA provides robust exploration capabilities but may converge slowly in complex solution spaces [266]. FA excels at local search refinement but can be trapped in local optima [267]. PSO offers efficient global search but may lack fine-tuning abilities in local regions [268]. To address this critical placement challenge while overcoming these algorithmic limitations, we propose MADCP, a Memetic Algorithm for DRL Component Placement that combines the strengths of GA, FA, and PSO. In ReinFog, MADCP is integrated into the DRL Component Placement Engine of the extended Scheduler module (see Fig. 5.5). It is invoked before DRL training to determine efficient placement of DRL Learners and Workers across nodes.

In this section, we first define the optimization problem we are addressing along with the MADCP. We also explain how our proposed MADCP combines the strengths of three methods: GA, FA, and PSO. In addition, we present a comprehensive analysis of the computational complexity of MADCP, examining its initialization and iterative optimization phases.

5.5.1 Optimization Problem Definition

The MADCP is designed to address a complex optimization problem in distributed computing environments. The primary goal is to efficiently place DRL components across heterogeneous computing nodes to optimize overall system performance.

Problem Formulation

Consider a set of DRL components $\mathcal{C} = \{C_1, C_2, \ldots, C_m\}$ that need to be assigned to a set of heterogeneous computing nodes $\mathcal{N} = \{N_1, N_2, \ldots, N_n\}$. Each node has different computational capabilities, memory sizes, and energy consumption rates.

Each component C_i has a computational requirement U_i (e.g., CPU cycles), a memory requirement M_i, and a deadline or time constraint D_i. Similarly, each node N_j is characterized by a computational capacity P_j, and available memory A_j.

Objective Function

The objective is to find an optimal assignment of DRL components to computing nodes that minimizes the total operation time and energy consumption while meeting all time and resource constraints. We define the objective function as:

$$\begin{aligned} \text{Minimize} \quad F = \sum_{i=1}^{m} \sum_{j=1}^{n} x_{ij} \big(& \omega_1 \cdot O(C_i, N_j) \\ & + \omega_2 \cdot E(C_i, N_j) \big), \end{aligned} \tag{5.1}$$

Here, x_{ij} is a binary variable indicating whether component C_i is assigned to node N_j:

$$x_{ij} = \begin{cases} 1, & \text{if component } C_i \text{ is assigned to node } N_j, \\ 0, & \text{otherwise.} \end{cases} \tag{5.2}$$

$O(C_i, N_j)$ is the operation time of component C_i on node N_j, $E(C_i, N_j)$ is the energy consumed by node N_j during the operation time $O(C_i, N_j)$, and ω_1 and ω_2 are weighting factors balancing the importance of operation time and energy consumption.

Constraints

The optimization problem is subject to the following constraints:

Resource Constraints: For each node N_j, the total computational and memory requirements of the assigned components should not exceed its capacity:

$$\sum_{i=1}^{m} x_{ij} U_i \leq P_j, \quad \forall N_j \in \mathcal{N}, \tag{5.3}$$

$$\sum_{i=1}^{m} x_{ij} M_i \leq A_j, \quad \forall N_j \in \mathcal{N}. \tag{5.4}$$

Deadline Constraints: Each component must complete the operation within its deadline:

$$O(C_i, N_j) \leq D_i, \quad \forall C_i \in \mathcal{C}, \quad \forall N_j \in \mathcal{N} \text{ where } x_{ij} = 1. \tag{5.5}$$

Assignment Constraints: Each component is assigned to exactly one node:

$$x_{ij} \in \{0, 1\}, \quad \forall C_i \in \mathcal{C}, \quad \forall N_j \in \mathcal{N}, \tag{5.6}$$

$$\sum_{j=1}^{n} x_{ij} = 1, \quad \forall C_i \in \mathcal{C}. \tag{5.7}$$

Mathematical Model

Combining the objective function and the constraints, the optimization problem can be formulated as:

$$\text{Minimize} \quad F = \sum_{i=1}^{m}\sum_{j=1}^{n} x_{ij}\Big(\omega_1 \cdot O(C_i, N_j) + \omega_2 \cdot E(C_i, N_j)\Big) \tag{5.8}$$

$$\text{Subject to} \quad \sum_{i=1}^{m} x_{ij} U_i \leq P_j, \quad \forall N_j \in \mathcal{N} \tag{5.9}$$

$$\sum_{i=1}^{m} x_{ij} M_i \leq A_j, \quad \forall N_j \in \mathcal{N} \tag{5.10}$$

$$O(C_i, N_j) \leq D_i, \quad \forall C_i \in \mathcal{C}, \quad \forall N_j \in \mathcal{N} \text{ where } x_{ij} = 1 \tag{5.11}$$

$$x_{ij} \in \{0, 1\}, \quad \forall C_i \in \mathcal{C}, \quad \forall N_j \in \mathcal{N} \tag{5.12}$$

$$\sum_{j=1}^{n} x_{ij} = 1, \quad \forall C_i \in \mathcal{C} \tag{5.13}$$

This problem belongs to a class of combinatorial optimization challenges, where the objective is to efficiently allocate resources under multiple constraints, making it computationally intensive (NP-hard). The complexity arises from the exponential number of possible component-to-node assignments as the number of components and nodes increases. Traditional exact algorithms become impractical for large-scale instances because they require prohibitive computational time to explore all possible solutions. Moreover, single-method heuristic algorithms often fail to adequately balance exploration and exploitation in the vast solution space, leading to suboptimal results.

5.5.2 MADCP

Given the limitations of existing methods in addressing large-scale, NP-hard optimization problems, we propose a novel memetic algorithm named MADCP to find near-

optimal solutions within reasonable computational time for DRL components (e.g., DRL Learners and DRL Workers) placement. MADCP synergistically combines the strengths of three optimization algorithms:

1. **GA:** Provides robust exploration capabilities by simulating the process of natural selection, allowing the algorithm to broadly traverse the solution space and maintain genetic diversity among solutions.

2. **FA:** Enhances local search by modeling the flashing behavior of fireflies, enabling fine-tuning in promising areas and improving exploitation of the solution space.

3. **PSO:** Contributes efficient global optimization and rapid convergence by simulating social behavior patterns of organisms, allowing particles (solutions) to adjust their positions based on personal and global best experiences.

By integrating these algorithms, MADCP achieves a balanced search strategy that mitigates the shortcomings of individual techniques, leading to improved performance in solving complex optimization problems. Algorithm 5.1 provides a detailed description of MADCP.

In the initialization phase (Lines 1–6), the algorithm begins by generating the set of components and nodes based on the provided parameters. The function **CreateComponents** initializes the components $\mathcal{C}$, each with specific computational and memory requirements, as well as deadlines. Similarly, **CreateNodes** initializes the nodes $\mathcal{N}$, each characterized by their computational capacities and available memory. An initial population of candidate solutions is generated using **GeneratePopulation**, where each individual represents a potential assignment of components to nodes. For the PSO phase, velocities for each individual are initialized using **InitializeVelocities**. The personal best solutions *personalOpt* are initially set to the current population, and the global best solution *globalOpt* is determined by evaluating the fitness of each individual using **CalculateFitness** and selecting the one with the highest fitness value.

The algorithm then enters the main loop (Lines 7–28), iterating over a predefined number of generations to evolve the population towards optimality. At the beginning of each generation, the fitness of each individual in the population is calculated using

Algorithm 5.1: MADCP

Input:
compParameters – parameters defining the components
nodeParameters – parameters defining the nodes
populationSize – number of individuals in the population
generations – number of iterations
numOperations – number of genetic operations per generation
mutationRate – probability of mutation
FAParameters – parameters (α, β, γ) for FA
PSOParameters – parameters (w, c_1, c_2) for PSO
Output: Optimal assignment of components to nodes

```
/* Initialization                                                    */
components ← CreateComponents(compParameters)
nodes ← CreateNodes(nodeParameters)
population ← GeneratePopulation(populationSize, components, nodes)
velocities ← InitializeVelocities(populationSize, components)
personalOpt ← population
globalOpt ← arg max_{p∈population} CalculateFitness(p, components)
for i ← 1 to generations do
    /* Fitness Evaluation                                            */
    foreach p ∈ population do
        fitnessValues[p] ← CalculateFitness(p, components)
    end foreach
    newPopulation ← ∅
    /* Genetic Algorithm Operations                                  */
    for j ← 1 to numOperations do
        /* Selection                                                 */
        parent1, parent2 ← SelectParents(population, fitnessValues)
        /* Crossover                                                 */
        child1, child2 ← Crossover(parent1, parent2)
        /* Mutation                                                  */
        child1 ← Mutate(child1, mutationRate, nodes)
        child2 ← Mutate(child2, mutationRate, nodes)
        Add child1 and child2 to newPopulation
    end for
    population ← newPopulation
    /* Firefly Algorithm Movement                                    */
    population ← FireflyMovement(population, components, α, β, γ)
    /* Particle Swarm Optimization Update                            */
    population, velocities ← PSOUpdate(population, velocities, personalOpt, globalOpt, w, c1, c2)
    /* Update Personal and Global Best Solutions                     */
    foreach p ∈ population do
        if CalculateFitness(p, components) > CalculateFitness(personalOpt[p], components) then
            personalOpt[p] ← p
        end if
    end foreach
    globalOpt ← UpdateGlobalOpt(population, globalOpt, components)
end for
return globalOpt
```

CalculateFitness (Lines 8–10). The fitness function is designed based on the objective function defined in the optimization problem, considering execution time and energy consumption.

In the Genetic Algorithm operations (Lines 12–18), selection, crossover, and mutation are performed to generate a new population. Pairs of parent individuals are selected from the current population based on their fitness values using **SelectParents** (Line 13). The selected parents undergo crossover using **Crossover** (Line 14) to produce offspring, combining parts of the parents' solutions to create new individuals and promote exploration of the solution space. The offspring are then subjected to mutation using **Mutate** (Lines 15–16), introducing random changes to some genes (component assignments) with a probability defined by the mutation rate. This helps maintain genetic diversity and prevents premature convergence. The newly created children are added to form a new population (Line 19), replacing the old one.

After the Genetic Algorithm operations, the Firefly Algorithm is applied to the population using **FireflyMovement** (Line 20). Each individual moves towards better solutions based on attractiveness and randomization parameters, enhancing local search by exploiting promising regions in the solution space.

Subsequently, the Particle Swarm Optimization phase updates the positions and velocities of individuals using **PSOUpdate** (Line 21). Velocities are updated based on the inertia weight, cognitive coefficient, and social coefficient, guiding individuals towards their personal best and the global best solutions. Positions are then updated based on the new velocities, contributing to global optimization and convergence of the algorithm.

The personal best solutions *personalOpt* are updated by comparing the current fitness of each individual with their personal best fitness (Lines 22–26); if an individual has achieved a better fitness, its personal best is updated accordingly. The global best solution *globalOpt* is updated by identifying the best individual in the current population based on fitness evaluations (Line 27). After completing all generations, the algorithm returns the global best solution *globalOpt* (Line 29), which represents the optimal assignment of components to nodes found by MADCP.

5.5.3 Computational Complexity Analysis

This section provides a detailed analysis of the computational complexity of the proposed MADCP. The complexity analysis is divided into two main parts: the initialization phase and the iterative optimization phase. By analyzing the time complexity of each phase, we derive the overall complexity of the algorithm.

Initialization Phase

The initialization phase consists of several operations, including the creation of components and nodes, population generation, velocity initialization, and initial fitness evaluation:

- **Component and Node Creation**: The components and nodes are created based on the input parameters, where M represents the number of components, and N denotes the number of nodes. The complexity of this step is $O(M)$ for component creation and $O(N)$ for node creation, respectively.
- **Population Generation**: The initial population, with size P, is generated based on the number of components and nodes. Because each individual in the population represents a mapping between components and nodes, the complexity of this step is $O(P \times M)$.
- **Velocity Initialization**: The initial velocity of each individual is set based on the number of individuals in the population. The velocity initialization is performed by iterating over each component, so the complexity of this step is $O(P \times M)$.
- **Fitness Evaluation**: The initial fitness of each individual in the population is calculated based on component assignments. The complexity for calculating the fitness of a single individual is $O(M)$. Therefore, evaluating the fitness of the entire population results in a complexity of $O(P \times M)$.

Combining the above components, the total complexity of the initialization phase can be expressed as:

$$O(M + N + P \times M). \tag{5.14}$$

Iterative Optimization Phase

The iterative optimization phase consists of fitness evaluation, GA operations (selection, crossover, and mutation), FA-based movement, PSO update, and the update of personal and global best solutions.

- **Fitness Evaluation**: In each generation, the fitness of every individual in the population is evaluated. The time complexity for evaluating the fitness of the entire population is $O(P \times M)$.
- **Genetic Algorithm Operations**: This step includes parent selection, crossover, and mutation:
  - *Selection*: Assuming a selection mechanism such as roulette wheel selection, the complexity is $O(P)$.
  - *Crossover*: The crossover operation generates two offspring, with a complexity of $O(M)$.
  - *Mutation*: Each offspring undergoes mutation with a complexity of $O(M)$.

  Assume the GA operations are performed $P/2$ times per generation, the total complexity for GA operations is:

$$O\left(\frac{P}{2} \times (P + 2 \times M)\right) = O\left(P^2 + P \times M\right). \tag{5.15}$$

- **Firefly Algorithm Movement**: The FA-based movement operation involves comparing and updating the position of every individual relative to other individuals in the population. The complexity of this step is $O(P^2 \times M)$.
- **Particle Swarm Optimization Update**: The PSO update step adjusts the position and velocity of each individual in the population. The complexity of this step is $O(P \times M)$.
- **Update of Personal and Global Best Solutions**: This step involves comparing the fitness of each individual in the population with its historical best solution and

updating the global best solution if necessary. The complexity for updating the personal and global best solutions is $O(P \times M)$.

The overall complexity of a single iteration is given by:

$$\begin{aligned} &O(P \times M) + O\left(P^2 + P \times M\right) + \\ &O(P^2 \times M) + O(P \times M) + O(P \times M), \end{aligned} \tag{5.16}$$

which can be simplified to:

$$O(P^2 \times M). \tag{5.17}$$

Assume the number of generations is G, the total complexity of the iterative optimization phase is:

$$O(G \times P^2 \times M). \tag{5.18}$$

Total Complexity

By combining the complexity of the initialization phase and the iterative optimization phase, the overall time complexity of the MADCP is:

$$O(M + N + (P \times M) + (G \times P^2 \times M)). \tag{5.19}$$

In most practical scenarios, the complexity of the iterative optimization phase $O(G \times P^2 \times M)$ dominates the overall complexity, especially for larger values of G, P, or M. Therefore, the overall time complexity of the MADCP can be expressed as:

$$O(N + G \times P^2 \times M). \tag{5.20}$$

The complexity analysis shows that although MADCP combines GA, FA, and PSO, the overall complexity does not significantly increase. Specifically, GAs selection, crossover, and mutation operations have a complexity of $O(P^2 + P \times M)$, and PSOs update operations have a complexity of $O(P \times M)$, both of which are relatively low and do not heavily impact the total complexity. The FAs movement operation, with a complexity of $O(P^2 \times M)$, becomes the dominant factor as the population size increases. Instead,

MADCP leverages GA to enhance population diversity, preventing premature convergence; FA to improve local search capabilities; and PSO to guide global convergence. This results in a superior optimization performance without adding substantial computational overhead. In essence, MADCP achieves a balanced integration of the strengths of each algorithm while maintaining manageable complexity, making it a highly effective optimization algorithm for complex optimization problems.

5.6 Performance Evaluation

This section presents a comprehensive evaluation of ReinFog's performance in addressing the complex IoT application scheduling problem across heterogeneous edge and cloud environments. In our performance evaluation, we adhered to established benchmarking practices to ensure the reliability of our results. We repeated each experiment 10 times per configuration to account for the variability in measurements. Moreover, we carefully controlled the experimental environment to minimize external disturbances and ensure consistent conditions for all tests. The final results presented are the averages of these repeated experiments, ensuring that they accurately reflect the typical system performance.

5.6.1 Experiments Setup

We conduct our experiments in a heterogeneous computing environment comprising cloud, edge/fog, and IoT layers. The cloud layer consists of a multi-cloud setup including one AWS VM (Intel Xeon, 1 core @ 2.4GHz, 1GB RAM), one Azure VM (Intel Xeon, 1 core @ 2.3GHz, 1GB RAM), and 30 Nectar Cloud VMs (AMD EPYC; 16 VMs with 2 cores @ 2.0GHz, 8GB RAM; 10 VMs with 4 cores @ 2.0GHz, 16GB RAM; 4 VMs with 8 cores @ 2.0GHz, 32GB RAM). The edge/fog layer consists of an Apple Macbook Pro (macOS, Apple M1 Pro, 8 cores, 16GB RAM), a Dell laptop (Linux, Intel Core i7, 8 cores @ 2.3GHz, 16GB RAM), and a Raspberry Pi 3B (Raspberry Pi OS, Broadcom BCM2837, 4 cores @ 1.2GHz, 1GB RAM). Devices in the IoT layer are equipped with 2 cores @ 3.2GHz and 4GB RAM. The network configuration is characterized by the following latency/band-

width metrics: IoT to Nectar 8-12ms/14-18MB/s, IoT to AWS 16-22ms/16-20MB/s, IoT to Azure 8-15ms/15-22MB/s, and IoT to edge/fog 1-4ms/130-150MB/s. We deploy four IoT applications in our experiments: Face Detection, Color Tracking, Face and Eye Detection, and Video Optical Character Recognition. These applications feature adjustable resolution and support both real-time and non-real-time processing.

Among the implemented/integrated DRL techniques, we use the following representative techniques to conduct our experiments. The configuration of each technique is discussed below:

- **IMPALA [93]**: IMPALA is a distributed DRL technique natively supported by ReinFog. We choose it for its exceptional scalability and efficiency in handling distributed learning scenarios [93]. In our experiments, it uses OrnsteinUhlenbeck noise exploration strategy and the reservoir sampling replay buffer, and its network structure introduces Transformers. The DRL component placement algorithm is set to MADCP.

- **A3C [88]**: A3C is a distributed DRL technique natively supported by ReinFog. We choose it for its asynchronous parallel learning architecture that enables efficient policy optimization through multiple independent actors operating concurrently [88]. In our experiments, it uses ϵ-greedy exploration strategy, and its network structure introduces LSTMs. The DRL component placement algorithm is set to FA.

- **PPO [40]**: PPO is a centralized DRL technique natively supported by ReinFog. We choose it for its stable policy optimization approach that uses a clipped objective function to prevent excessive policy updates while maintaining high sample efficiency [269]. In our experiments, it uses OrnsteinUhlenbeck noise exploration strategy and the reservoir sampling replay buffer, and its network structure introduces LSTMs. The DRL component placement algorithm is set to PSO.

- **DQN [72]**: DQN is a centralized DRL technique natively supported by ReinFog. We choose it as it is a fundamental and widely-adopted DRL technique that effectively combines Q-learning with DNNs through experience replay and target

networks to stabilize training [265]. In our experiments, it uses ϵ-greedy exploration strategy and the random sampling replay buffer, and its network structure introduces RNNs. The DRL component placement algorithm is set to GA.

- **R2D2 [94]**: R2D2 is a distributed DRL technique imported by ReinFog from the Ray library. We choose it for its distributed architecture that extends DQN with RNNs to handle temporal dependencies while maintaining efficient parallel learning [94]. In our experiments, the DRL component placement algorithm is set to MADCP.

- **SAC [84]**: SAC is a centralized DRL technique imported by ReinFog from the Ray library. We choose it for its maximum entropy approach that provides automatic tuning to balance exploration and exploitation, leading to robust and stable learning performance [84]. In our experiments, the DRL component placement algorithm is set to PSO.

- **OHNSGA [44]**: OHNSGA is a GA-based meta-heuristic algorithm natively supported by FogBus2. We extended this technique to support multi-objective optimization for comparison with DRL techniques.

5.6.2 Hyperparameters Settings

We conduct an extensive grid search to fine-tune the hyperparameters for each technique used in our experiments. The grid search involves systematically exploring various combinations of hyperparameters within predefined ranges to identify the optimal configuration for each technique.

For the neural network architecture, following the guidance from [270] and [271], we experiment with different numbers of fully connected layers (ranging from 2 to 5) and various combinations of hidden layer units (16, 32, 64, 128, 256). We find that a 3-layer architecture with [256, 256, 128] hidden units provided the best balance between model complexity and performance across most techniques.

The choice of activation function can significantly impact the learning dynamics. While ReLU (Rectified Linear Unit) is found to be effective for most techniques due to

its non-linearity and reduced likelihood of vanishing gradients, we observe that TanH (Hyperbolic Tangent) works better for A3C, possibly due to its bounded output range.

Following the guidance from [93], [88], [40], [75], and [84], learning rates are tuned within the range of 0.0001 to 0.01, with most techniques performing optimally around 0.001. The discount factor, which balances immediate and future rewards, is tuned within the range of 0.8 to 0.99. Most techniques show optimal performance with a discount factor of 0.99, while A3C performed better with a slightly lower value of 0.9.

For the OHNSGA algorithm, following the guidance from [179] and [180], which is not a DRL technique, we focus on tuning the population size and number of generations. The population size is varied from 50 to 500, and the number of generations is tested in the range of 50 to 200. After extensive experimentation, we find that a population size of 200 and 100 generations provides a good trade-off between solution quality and computational time. Table 5.3 presents the optimal hyperparameters identified for each technique.

Table 5.3: Technique hyperparameters

Hyperparameters	IMPALA	A3C	PPO	DQN	R2D2	SAC	OHNSGA
Fully Connected Layers	3	3	3	3	3	3	-
Hidden Layer Units	[256,256,128]	[256,256,128]	[256,256,128]	[256,256,128]	[256,256,128]	[256,256,128]	-
Activation Function	ReLU	TanH	ReLU	ReLU	ReLU	ReLU	-
Learning Rate	0.001	0.001	0.001	0.01	0.01	0.0001	-
Discount Factor	0.99	0.9	0.99	0.99	0.99	0.99	-
Population Size	-	-	-	-	-	-	200
Generations Number	-	-	-	-	-	-	100

In addition to the DRL techniques and OHNSGA, we also fine-tune the parameters for the DRL component placement algorithms: MADCP, GA, FA, and PSO. For MADCP, we tune the following parameters: population size (from 50 to 500), number of generations (from 50 to 200), crossover rate (from 0.6 to 1.0), light absorption coefficient (γ) (from 0.1 to 1.0), and inertia weight (w) (from 0.4 to 0.9). For GA, following the guidance from [272], we tune the following parameters: population size (from 50 to 500), number of generations (from 50 to 200), crossover rate (from 0.6 to 1.0), mutation rate (from 0.01 to 0.05). For FA, following the guidance from [273], we focus on tuning: number of fireflies (from 20 to 200), randomization parameter (α) (from 0.1 to 1.0), attractiveness coefficient (β) (from 0.1 to 1.0), and light absorption coefficient (γ) (from 0.1 to 1.0). For PSO, following the guidance from [274], parameters are optimized as follows: swarm

size (from 20 to 200), cognitive coefficient (c_1) (from 1.5 to 2.5), social coefficient (c_2) (from 1.5 to 2.5), and inertia weight (w) (from 0.4 to 0.9).

After extensive experimentation, we identify the optimal parameter settings for each DRL component placement algorithm when paired with different DRL techniques. For MADCP, the optimal settings are population size: 200, number of generations: 100, crossover rate: 0.8, light absorption coefficient (γ): 0.5, and inertia weight (w): 0.7 when used with IMPALA. When paired with R2D2, most parameters remain the same, except for a slight change in inertia weight (w) to 0.8. For GA, which is used with DQN, the optimal parameters are population size: 200, number of generations: 100, crossover rate: 0.9, and mutation rate: 0.05. For FA, when used with A3C, the optimal settings are number of fireflies: 100, randomization parameter (α): 0.2, attractiveness coefficient (β): 0.8, and light absorption coefficient (γ): 0.1. When used with PPO, only γ changes to 0.2, while other parameters remain the same. PSO maintains the same parameters (swarm size: 100, cognitive coefficient (c_1): 2.0, social coefficient (c_2): 2.0, and inertia weight (w): 0.7) in both SAC and PPO.

These fine-tuned parameters are used consistently across all experiments to ensure fair comparison. It's worth noting that while these hyperparameters yield the best overall performance in our experimental setup, the optimal configuration may vary depending on the specific characteristics of the deployment environment.

5.6.3 Evaluation Metrics

In our experiments, we focus on three key metrics: Response Time ($\mathcal{RT}$), Energy Consumption ($\mathcal{EC}$), and Weighted Cost ($\mathcal{WC}$). These metrics are applied to a set of IoT applications $\mathcal{A}$, where each application $\mathcal{A}_i$ is modeled as a Directed Acyclic Graph (DAG) consisting of dependent tasks $\mathcal{T}_i^j$, scheduled across a set of nodes $\mathcal{N}$. $\mathcal{SC}$ denotes the set of scheduling configurations for these applications.

Response Time ($\mathcal{RT}$): This metric represents the total time required to process all applications. It is calculated as:

$$\mathcal{RT}(\mathcal{SC}) = \sum_{i=1}^{|\mathcal{A}|} \sum_{j=1}^{|\mathcal{A}_i|} \left(\mathcal{RT}(\mathcal{SC}_i^j) \times \mathcal{CP}(\mathcal{T}_i^j) \right), \tag{5.21}$$

where $\mathcal{SC}_i^j$ refers to the scheduling configuration for task $\mathcal{T}_i^j$, and $\mathcal{CP}(\mathcal{T}_i^j)$ is a binary variable that indicates whether the task lies on the critical path of the application. The critical path determines the minimum time required for the application to complete.

Energy Consumption ($\mathcal{EC}$): This metric measures the total energy consumed during the execution of the applications:

$$\mathcal{EC}(\mathcal{SC}) = \sum_{i=1}^{|\mathcal{A}|} \sum_{j=1}^{|\mathcal{A}_i|} \mathcal{EC}(\mathcal{SC}_i^j), \tag{5.22}$$

where $\mathcal{EC}(\mathcal{SC}_i^j)$ represents the energy consumed for task $\mathcal{T}_i^j$ under the given scheduling configuration $\mathcal{SC}_i^j$.

Weighted Cost ($\mathcal{WC}$): Weighted Cost ($\mathcal{WC}$) is a composite metric that balances both response time and energy consumption. It is defined as:

$$\mathcal{WC}(\mathcal{SC}) = w_1 \times Norm(\mathcal{RT}(\mathcal{SC})) + w_2 \times Norm(\mathcal{EC}(\mathcal{SC})), \tag{5.23}$$

where w_1 and w_2 are weight parameters used to control the relative importance of Response Time ($\mathcal{RT}$) and Energy Consumption ($\mathcal{EC}$), respectively, with $w_1 + w_2 = 1$. The function $Norm(x)$ performs normalization of a value x relative to a predefined baseline to eliminate the differences in units and scales between Response Time ($\mathcal{RT}$) and Energy Consumption ($\mathcal{EC}$). This ensures that both metrics contribute equally to the Weighted Cost ($\mathcal{WC}$) calculation, independent of their absolute values or units. In our experiments, we set $w_1 = 0.5$ and $w_2 = 0.5$, giving equal priority to both factors.

Based on the metrics, the reward function is defined as the negative Response Time ($\mathcal{RT}$), Energy Consumption ($\mathcal{EC}$), and Weighted Cost ($\mathcal{WC}$) if the application is success-

fully executed, or a penalty if it fails. The optimization goal is to minimize the three metrics, respectively, while satisfying the corresponding constraints related to application requirements and node processing capabilities.

5.6.4 ReinFog Framework Performance Analysis

In this section, we conduct a comprehensive evaluation of ReinFog's framework performance. Among the related works, we choose FogBus2 as our baseline framework for its most comprehensive generic capabilities for resource management framework and similar modular architecture. We first analyze the startup time of different components in ReinFog. We then evaluate their RAM usage, demonstrating the lightweight nature of our framework despite its DRL capabilities. Next, we assess the environmental impact by evaluating carbon dioxide (CO_2) emissions across the electricity generation patterns in different regions. Finally, we examine the scalability of the framework by analyzing its performance with varying numbers of DRL Workers.

Framework Startup Time Analysis

In this experiment, we compare the startup time of ReinFog and FogBus2 across three key components: Master, Actor, and User. As shown in Fig. 5.6, the startup time of the Actor (0.89s) and User (0.47s) components in ReinFog are nearly identical to those in FogBus2. However, the Master component in ReinFog requires approximately 0.25 seconds more startup time (1.26s vs 1.01s) compared to FogBus2's Master component, due to the initialization of DRL components. Despite this slight increase in the Master's startup time, the overall impact on system performance is minimal, demonstrating ReinFog's efficient design in integrating DRL capabilities while maintaining reasonable startup overhead.

Framework RAM Usage Analysis

In this experiment, we evaluate the RAM usage of ReinFog and FogBus2 across the Master, Actor, and User components. As illustrated in Fig. 5.7, the RAM usage of the User

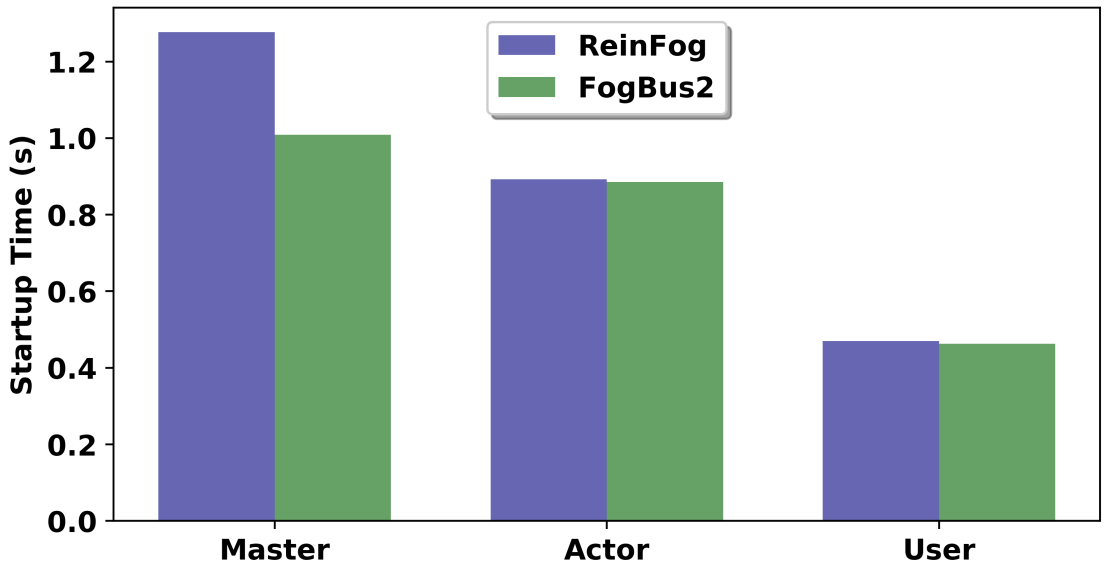


Figure 5.6: Startup time comparison between ReinFog and FogBus2 components

(81MB) and Actor (61MB) components exhibits minimal differences between ReinFog and FogBus2. The Master component in ReinFog utilizes approximately 5MB of additional memory compared to FogBus2 (59MB versus 54MB) due to the DRL components integration. This modest increase in RAM usage demonstrates that ReinFog maintains a lightweight design while incorporating advanced DRL capabilities.

Framework Carbon Dioxide Emissions Analysis

In this experiment, we evaluate the average CO_2 emissions of ReinFog across all integrated DRL techniques and compare it with FogBus2 during one hour of IoT application processing. To enable a fair comparison, we extended FogBus2's scheduler with energy optimization capabilities. The primary objective is to assess whether the integration of DRL components in ReinFog introduces significant environmental overhead despite its energy optimization capabilities. The analysis considers the distinct electricity generation patterns of Australia, the United States of America (USA), and Germany to provide a comprehensive evaluation of environmental impact in regions with different energy source distributions. We record the electricity consumed and estimate the GHG

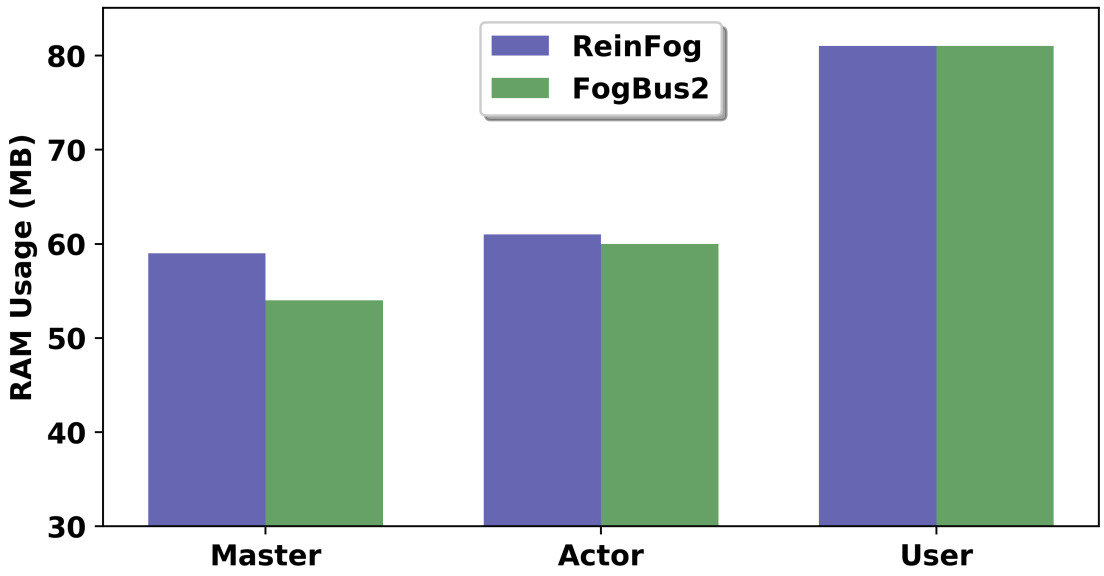


Figure 5.7: RAM usage comparison between ReinFog and FogBus2 components

emissions following the electricity generation patterns in Australia[2], the USA[3], and Germany[4], using the formula $GHG = E \times \sum_i (e_i \times p_i)$, where E represents the total electricity consumed, e_i denotes the emission factor, and p_i indicates the proportion of the source i (i.e., coal, gas) in producing electricity. As illustrated in Fig. 5.8, ReinFog consistently produces lower CO_2 emissions than FogBus2 across all regions. Specifically, in Australia, ReinFog emits 0.99g compared to FogBus2's 1.24g. The difference is more pronounced in the USA, where ReinFog generates 1.35g versus FogBus2's 1.67g. Germany shows the lowest emissions for both frameworks (ReinFog: 0.88g, FogBus2: 1.09g), attributable to its higher proportion of renewable energy sources. These results demonstrate that ReinFog not only provides advanced DRL capabilities but also maintains a lower carbon footprint compared to FogBus2, with the actual environmental impact varying based on regional energy policies and power grid compositions.

[2]https://www.energy.gov.au/data/electricity-generation
[3]https://www.eia.gov/tools/faqs
[4]https://www.umweltbundesamt.de/themen/co2-emissionen-pro-kilowattstunde-strom-stiegen-in

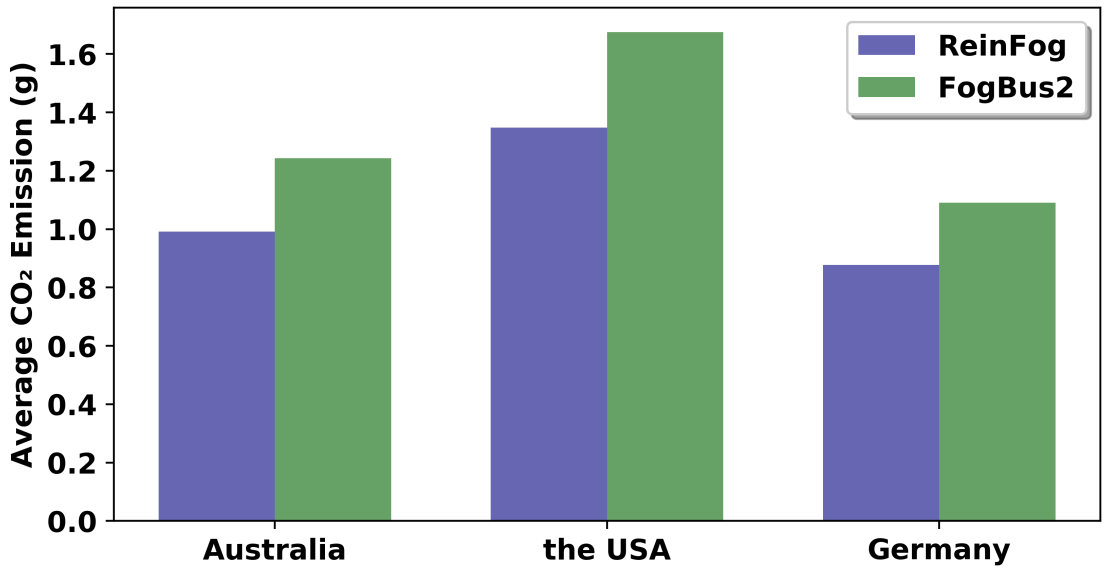


Figure 5.8: Hourly CO2 emissions comparison between ReinFog and FogBus2 across different regions

Framework Scalability Analysis

In this experiment, we evaluate the scalability of ReinFog by systematically varying the number of DRL Workers from 1 to 30. As illustrated in Fig. 5.9, as the number of DRL Workers increases, both framework RAM usage and Master startup time show controlled growth. Specifically, the total framework RAM usage exhibits a linear increase from 600MB with one DRL Worker to 660MB with 30 DRL Workers, representing only a 10% increase in memory consumption (i.e., roughly 2 MB of additional RAM usage per DRL Worker). Meanwhile, the Master startup time demonstrates a gradual increase from 1.25s to 1.56s, reflecting a modest rise of approximately 0.3s. These results indicate that ReinFog maintains efficient resource utilization and reasonable startup overhead even with a significant increase in the number of DRL Workers, demonstrating its excellent scalability for large-scale deployments.

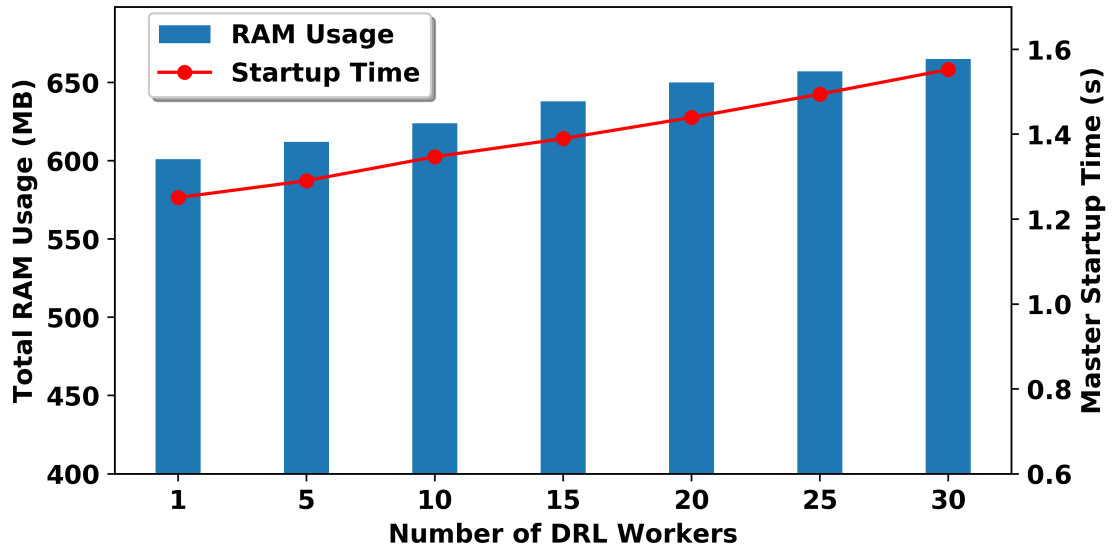


Figure 5.9: Impact of increasing DRL Workers on ReinFog RAM usage and startup time

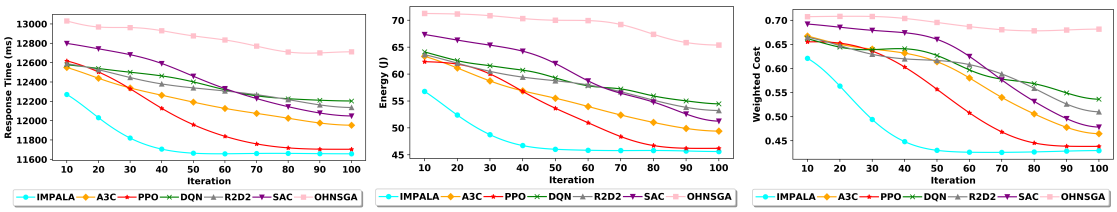
5.6.5 ReinFog DRL Scheduling Techniques Analysis

In this section, we evaluate the performance of different DRL scheduling techniques integrated into ReinFog by comparing them with OHNSGA [44], the meta-heuristic scheduling approach originally proposed in FogBus2. This allows us to directly assess the benefits of DRL-based scheduling over a well-established non-learning baseline within a consistent system environment. We first analyze the convergence performance of various scheduling techniques during both training and evaluation phases across multiple metrics, including response time, energy consumption, and weighted cost. We then examine the computational overhead introduced by different scheduling techniques. Following this, we evaluate the scalability of these techniques by testing their performance across different numbers of nodes. Finally, we assess the environmental impact of different techniques by analyzing the CO_2 emissions.

Scheduling Techniques Convergence Analysis

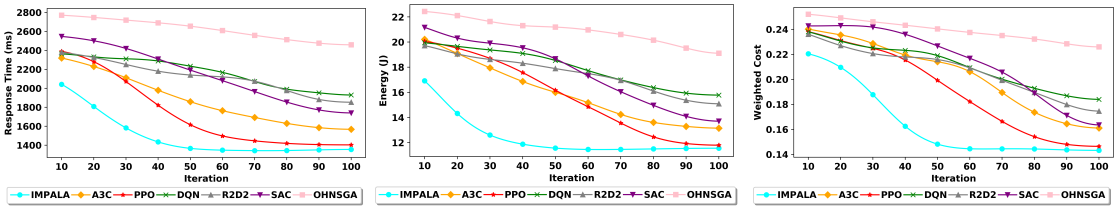
In this experiment, we evaluate and compare the convergence performance of various DRL techniques in ReinFog with OHNSGA during both training and evaluation phases.

The comparison is conducted across three metrics: response time, energy consumption, and weighted cost, as shown in Fig. 5.10 and Fig. 5.11. During the training phase, the video resolution is set to 480p, while in the evaluation phase, it is reduced to 240p to test the adaptability of different techniques. In the training phases, across all three metrics, IMPALA consistently demonstrates superior performance, achieving the fastest convergence (around iteration 40-50) and the lowest final values. PPO emerges as the second-best performer, followed by A3C with moderate performance. The imported techniques, R2D2 and SAC, converge more slowly, while DQN demonstrates relatively stable but the slowest convergence among all DRL techniques. OHNSGA, the meta-heuristic baseline from FogBus2, consistently performs the worst, showing minimal improvement over iterations and the highest final values. The evaluation phase maintains similar performance patterns but with overall lower metric values due to the reduced video resolution. In the evaluation phase, ReinFog's DRL techniques keep achieving significant improvements over OHNSGA, with reductions of up to 45% in response time, 39% in energy consumption, and 37% in weighted cost. The results demonstrate the robustness and generalization capability of ReinFog's DRL techniques, particularly IMPALA, in adapting to varying workload conditions. The significant performance gap between ReinFog's DRL techniques and FogBus2's OHNSGA validates the effectiveness of DRL-based approaches in dynamic IoT application scheduling scenarios.



(a) Response time **(b)** Energy consumption **(c)** Weighted cost

Figure 5.10: Convergence performance comparison of scheduling techniques during training phase



(a) Response time **(b)** Energy consumption **(c)** Weighted cost

Figure 5.11: Convergence performance comparison of scheduling techniques during evaluation phase

Scheduling Techniques Overhead Analysis

In this experiment, we compare the average scheduling overhead of different techniques in ReinFog, against OHNSGA, as shown in Fig. 5.12. Among ReinFog's native DRL techniques, DQN shows the lowest overhead at approximately 8ms, followed by PPO (11ms), while IMPALA and A3C both demonstrate an overhead of around 13ms. The imported techniques demonstrate higher overhead, with R2D2 at around 28ms and SAC at 25ms. FogBus2's OHNSGA exhibits the lowest overhead at 8ms. The results show that native integrated DRL techniques (IMPALA, A3C, PPO, DQN) consistently demonstrate lower overhead than external imported ones (R2D2, SAC), and centralized techniques (DQN, PPO) generally incur less overhead than distributed ones (IMPALA, A3C). While OHNSGA shows the lowest overhead, the superior convergence performance of ReinFog's DRL techniques justifies their moderate scheduling overhead.

Scheduling Techniques Scalability Analysis

In this experiment, we evaluate and compare the scalability of different scheduling techniques by varying the number of nodes from 5 to 30. As the response time and energy consumption metrics show similar trends, we present only the weighted cost results here. As shown in Fig. 5.13, IMPALA consistently achieves the lowest and most stable weighted cost (around 0.14) across all node configurations, showing excellent scalability. Other DRL techniques, including A3C, PPO, DQN, R2D2, and SAC, demonstrate moderate increases in weighted cost as the number of nodes grows. While their relative performance shows some variations across different scales, they all maintain signifi-

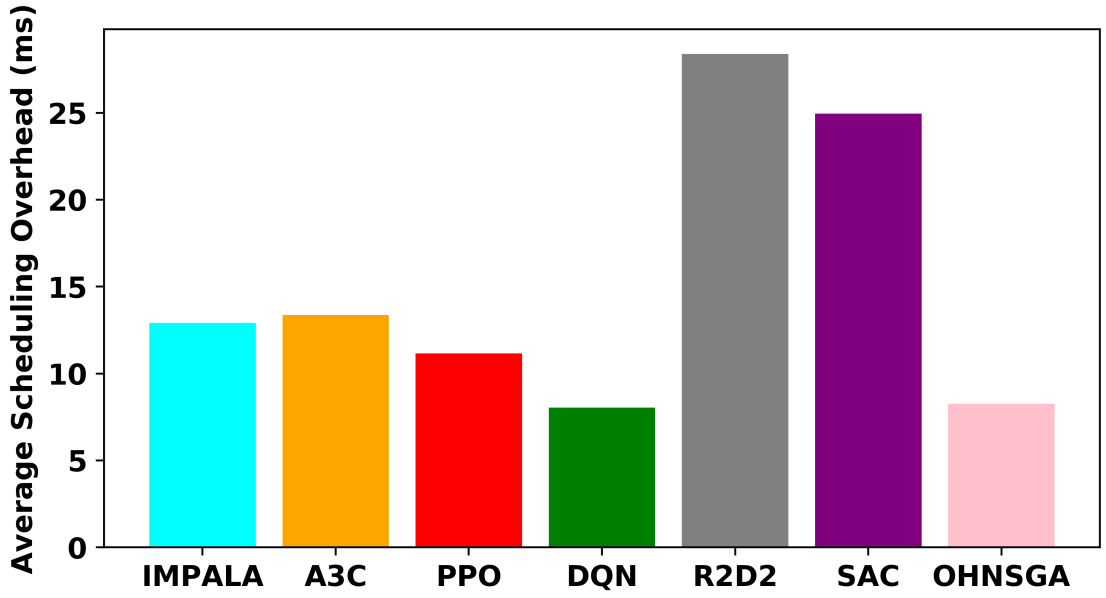


Figure 5.12: Average scheduling overhead comparison across different scheduling techniques

cantly better performance than OHNSGA. Specifically, A3C and PPO show better scalability with lower weighted costs, SAC maintains intermediate performance, while DQN and R2D2 exhibit relatively higher weighted costs as the system scales up. In contrast, OHNSGA exhibits the highest weighted cost across all scales, maintaining its weighted cost between 0.22 and 0.25 as the system scales up. This analysis clearly demonstrates that while some DRL techniques in ReinFog exhibit performance fluctuations, overall they demonstrate good scalability in scheduling IoT applications across different nodes. In contrast, traditional meta-heuristic methods like OHNSGA perform poorly across all environment scales.

Scheduling Techniques Sustainability Analysis

In this experiment, we evaluate the environmental sustainability of different scheduling techniques by comparing the CO_2 emissions when processing IoT applications for one hour. We estimate the CO_2 emissions in Australia, the USA, and Germany following the same approach described in Section 5.6.4. The results are shown in Fig. 5.14. In general,

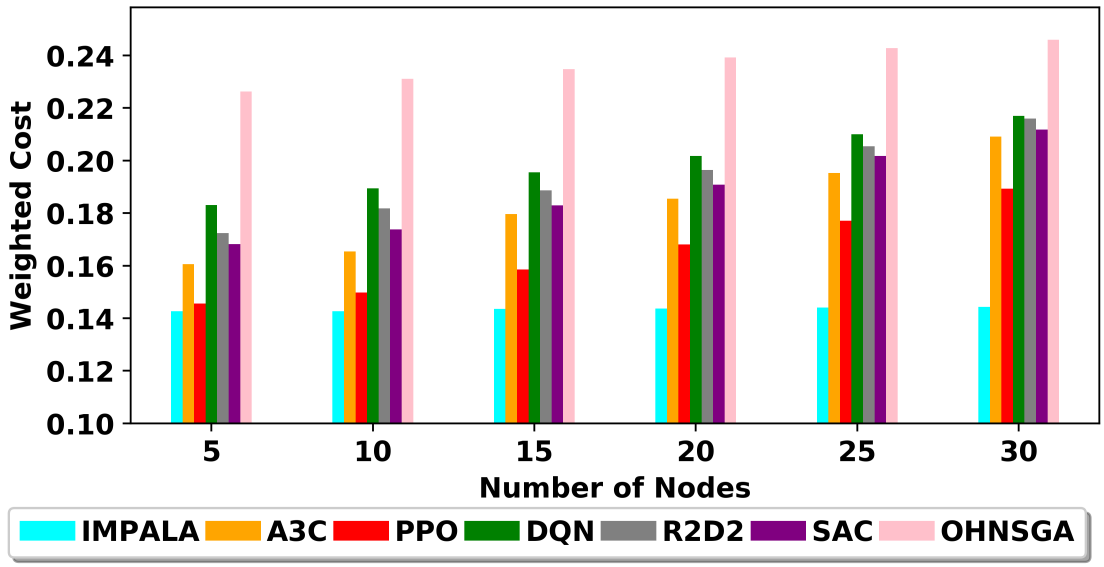


Figure 5.13: Impact of number of nodes on scheduling performance across different scheduling techniques

all DRL techniques demonstrate lower CO_2 emissions compared to OHNSGA. Among the DRL techniques, IMPALA shows the lowest emissions across all regions (0.84g in Australia, 1.14g in the USA, and 0.74g in Germany), while DQN exhibits the highest emissions among DRL techniques (1.08g in Australia, 1.45g in the USA, and 0.94g in Germany). OHNSGA produces significantly higher emissions (1.25g in Australia, 1.68g in the USA, and 1.10g in Germany), approximately 50% more than IMPALA. Notably, all techniques show consistently higher emissions in the USA while Germany demonstrates the lowest emissions among the three regions. This pattern can be attributed to differences in regional electricity generation patterns and their corresponding carbon intensities. These results demonstrate that the DRL techniques in ReinFog can better contribute to environmental sustainability compared to traditional meta-heuristic methods.

5.6.6 DRL Component Placement Algorithm Analysis

In this section, we evaluate the performance of our proposed MADCP in comparison with other DRL component placement algorithms, including GA, FA, PSO, and a ran-

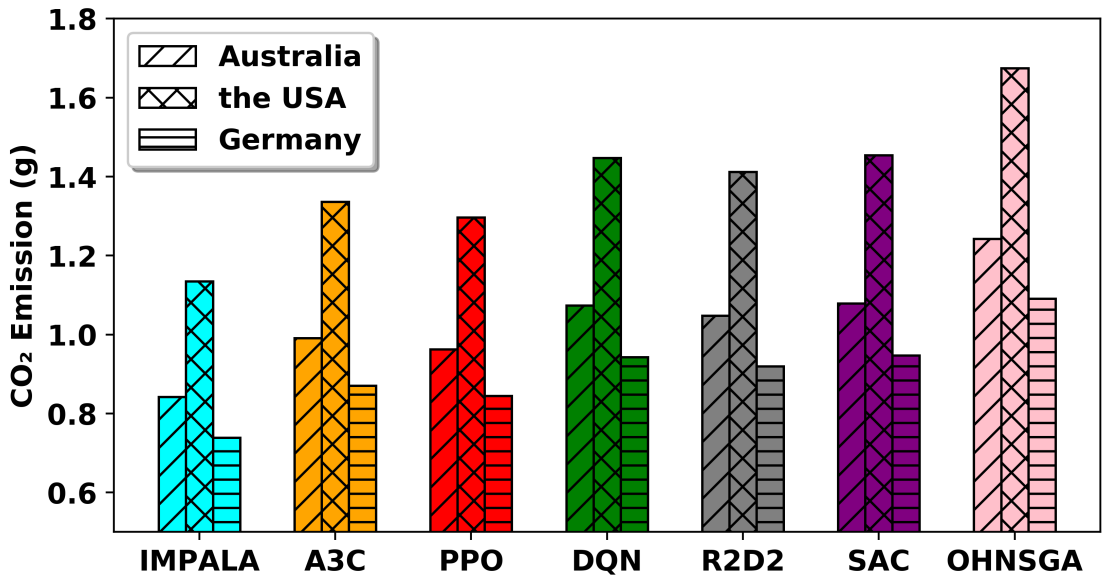
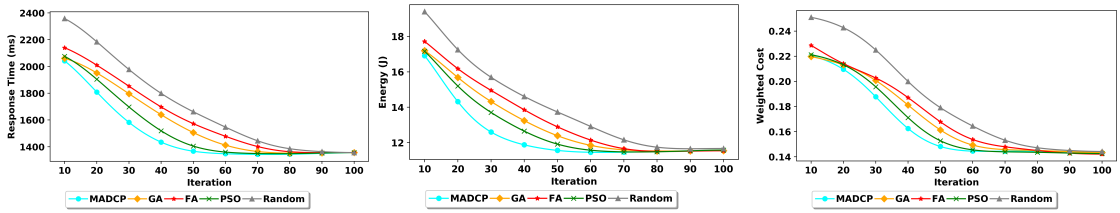


Figure 5.14: Hourly CO_2 emissions comparison of scheduling techniques across different regions

dom placement method. Given that IMPALA demonstrates superior performance in our previous convergence analysis, we employ it with various DRL component placement algorithms to assess their impact on overall performance. This study focuses on analyzing the effects of various DRL component placement algorithms on the convergence rate and scheduling overhead of IMPALA.

DRL Component Placement Algorithm Convergence Analysis

To assess the impact of different DRL component placement algorithms on IMPALA's performance, we conducted a convergence analysis across the three metrics: response time, energy consumption, and weighted cost. The results are demonstrated in Figure 5.15. MADCP consistently outperforms other DRL component placement algorithms across all metrics, achieving the fastest convergence rates, typically stabilizing after 50-60 iterations. PSO and GA follow closely, performing similarly and surpassing FA. Although the FA method converges slightly more slowly, it still outperforms the random placement method, which shows the poorest performance across all metrics and only stabilizes after 90 iterations. Notably, MADCP accelerates the convergence rate by up



(a) Response time **(b)** Energy consumption **(c)** Weighted cost

Figure 5.15: Impact of different DRL component placement algorithms on IMPALA's convergence performance

to 38% compared to random placement method. These results highlight MADCP's effectiveness in providing a strong starting point for DRL techniques. Its superior performance stems from combining the strengths of GA, FA, and PSO, leading to more efficient exploration of the solution space. This analysis also underscores the importance of intelligent placement of DRL components in DRL-based resource management frameworks.

DRL Component Placement Algorithm Overhead Analysis

In this experiment, we evaluate and compare the average scheduling overhead of IMPALA when employed with different DRL component placement algorithms. As shown in Fig. 5.16, MADCP demonstrates the lowest average overhead at approximately 13ms, followed by PSO at 15ms, GA at 16ms, and FA at 18ms. The random placement method shows the highest overhead at about 24ms, nearly twice that of MADCP. These results highlight MADCP's effectiveness in reducing computational overhead when used to place DRL components in the ReinFog framework.

5.7 Summary

This chapter proposed ReinFog, a novel framework leveraging DRL mechanisms and techniques for adaptive resource management in edge/fog and cloud computing environments. ReinFog addresses the challenge of efficiently scheduling heterogeneous IoT applications across diverse computing resources through its modular and extensi-

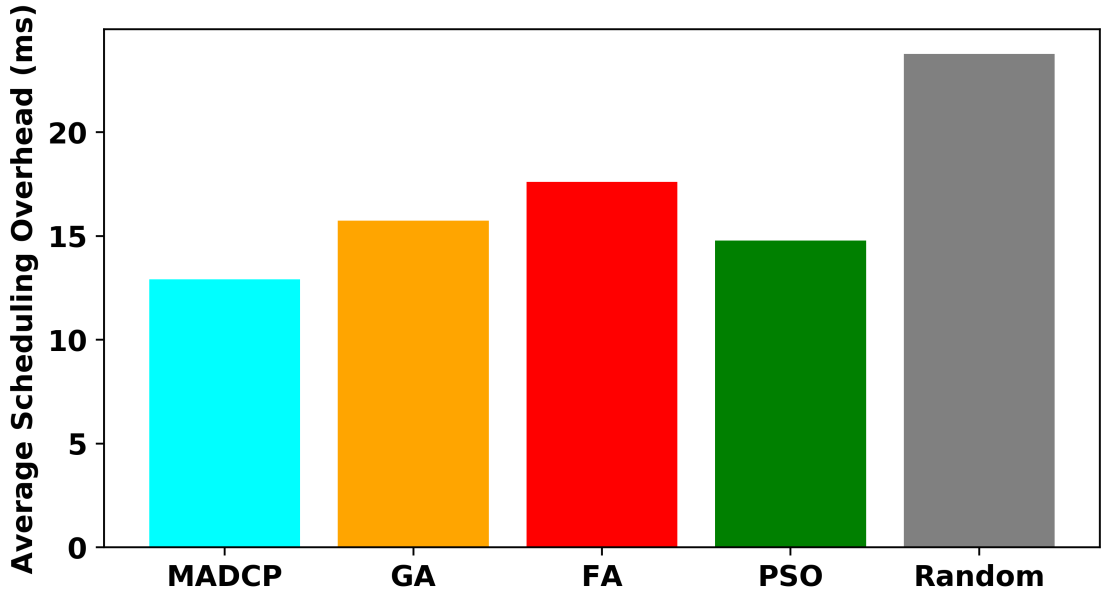


Figure 5.16: Impact of different DRL component placement algorithms on IMPALA's average scheduling overhead

ble DRL components. It offers capabilities to support centralized and distributed DRL techniques and allows integration of both native and library-based DRL techniques. It features customizable deployment configurations, allowing users to flexibly configure DRL Learners and Workers based on system requirements. Additionally, it incorporates MADCP, an efficient DRL component placement algorithm that dynamically optimizes the allocation of DRL Learners and Workers, enhancing DRL-based scheduling techniques performance in distributed environments. Our extensive experiments demonstrate that ReinFog is a lightweight and scalable framework capable of effectively scheduling IoT applications under diverse optimization objectives.

Chapter 6

A Knowledge Distillation-empowered Adaptive Federated Reinforcement Learning Framework for Multi-Domain IoT Applications Scheduling

The rapid proliferation of IoT applications across heterogeneous Cloud-Edge-IoT environments presents significant challenges in distributed scheduling optimization. Existing approaches struggle with fixed neural network architectures incompatible with computational heterogeneity, non-IID data distributions across scheduling domains, and insufficient cross-domain collaboration. To address these challenges, we propose KD-AFRL, a Knowledge Distillation-empowered Adaptive Federated Reinforcement Learning framework for multi-domain IoT scheduling. KD-AFRL introduces three core innovations: a resource-aware hybrid architecture generation mechanism enabling collaborative learning across heterogeneous devices with optimal resource utilization; a privacy-preserving environment-clustered federated learning approach for handling non-IID challenges; and an environment-oriented cross-architecture knowledge distillation mechanism for efficient knowledge transfer. Practical experiments demonstrate substantial improvements over the best baseline: 21% faster convergence and performance gains of 15.7%, 10.8%, and 13.9% in completion time, energy consumption, and weighted cost, respectively. Scalability experiments show that KD-AFRL achieves 3-5× better performance retention than existing solutions as the number of domains increases.

This chapter is derived from:

- **Zhiyu Wang**, Mohammad Goudarzi, Mingming Gong, and Rajkumar Buyya, "A Knowledge Distillation-empowered Adaptive Federated Reinforcement Learning Framework for Multi-Domain IoT Applications Scheduling", *IEEE Transactions on Mobile Computing (TMC)*, 2026 [In Press].

6.1 Introduction

The rapid proliferation of Internet of Things (IoT) applications has fundamentally transformed computing paradigms, creating unprecedented demands for intelligent resource management in heterogeneous Cloud-Edge-IoT environments [275]. Modern IoT deployments span multiple autonomous domainsfrom smart cities and industrial automation to healthcare monitoring and autonomous vehicleseach exhibiting distinct computational capabilities, workload characteristics, and operational constraints [276, 277]. These applications typically consist of interdependent tasks forming complex Directed Acyclic Graphs (DAGs), requiring sophisticated scheduling strategies to optimize completion time, energy consumption, and operational costs while respecting dependency constraints and resource limitations [278]. For example, in a multi-city smart transportation collaboration scenario, traffic management departments across different cities need to schedule various IoT applications such as video surveillance analysis, traffic flow prediction, and signal optimization, but each city faces distinctly different environmental characteristics: some cities have relatively stable traffic patterns and good network conditions, others face highly dynamic traffic flows and unstable network environments, while still others need to handle scheduling challenges under extreme weather conditions. By leveraging multi-domain collaborative learning, cities with diverse operational contexts can exchange and incorporate specialized scheduling insights, leading to more resilient, adaptive, and globally optimized IoT application performance across heterogeneous environments.

To address these complex scheduling optimization challenges, Deep Reinforcement Learning (DRL) has emerged as a promising solution for adaptive policy learning [279]. However, traditional centralized DRL scheduling faces significant scalability and adaptability challenges, struggling to capture the dynamic nature of multi-domain IoT ecosystems where resource availability, network conditions, and workload patterns fluctuate continuously across distributed domains [54]. These limitations become particularly pronounced when managing heterogeneous computational resources ranging from resource-constrained IoT devices to high-performance cloud servers. To overcome these limitations, distributed DRL has emerged to improve system scalability by distribut-

ing computation across multiple devices. However, existing distributed DRL scheduling methods suffer from several fundamental deficiencies. First, most methods employ fixed neural network architectures that cannot adapt to computational heterogeneity, resulting in resource mismatches across devices. Second, existing works operate under unrealistic IID data assumptions, neglecting the non-IID nature of real-world multi-domain deployments where different domains exhibit distinct environmental characteristics and workload patterns. Third, current distributed DRL methods lack effective cross-domain collaboration mechanisms, failing to exploit the potential for knowledge sharing between different domains.

Federated Learning (FL) offers a compelling solution for enabling collaborative learning across distributed domains while preserving data locality and privacy [280]. By allowing multiple domains to jointly train machine learning models without sharing raw data, FL addresses privacy concerns and reduces communication overhead associated with centralized approaches [281]. However, applying federated learning to multi-domain IoT scheduling introduces several fundamental challenges. First, the heterogeneity in computational capabilities across domains necessitates adaptive model architectures that can scale appropriately to device constraints while maintaining learning effectiveness [51]. Second, the non-IID nature of scheduling environments across domains can significantly degrade federated learning performance when domains with dissimilar characteristics attempt to share model parameters directly [22]. Third, domains with different computational capabilities often employ architectures of varying complexity, making direct parameter aggregation impossible and preventing resource-constrained domains from benefiting from knowledge acquired by more capable domains [282].

To address these challenges, we propose KD-AFRL, a Knowledge Distillation-empowered Adaptive Federated Reinforcement Learning framework that enables effective multi-domain IoT application scheduling. By introducing resource-aware adaptive architecture generation, privacy-preserving environment-clustered federated learning, and environment-oriented cross-architecture knowledge distillation, KD-AFRL enables heterogeneous devices, from resource-constrained IoT devices to powerful cloud servers, to collaboratively learn optimal scheduling policies despite non-IID data distributions across domains, while preserving data privacy and adapting to their computational constraints.

The main contributions of this chapter are fourfold:

- We design a resource-aware hybrid architecture generation mechanism that dynamically adapts model complexity to each device's computational capacity. Leveraging a dual-zone architecture that comprises shared foundational zones and personalized adaptation zones, the mechanism preserves federated learning compatibility across domains, enabling heterogeneous devices to participate in collaborative learning while maximizing optimal resource utilization.

- We propose a privacy-preserving environment-clustered federated learning mechanism that addresses non-IID challenges in multi-domain deployments. The mechanism leverages K-means clustering to group domains with similar environmental characteristics, facilitating targeted collaboration among compatible domains while mitigating negative transfer from dissimilar environments. To ensure data confidentiality, we incorporate ϵ-differential privacy throughout the clustering and federated aggregation pipeline, protecting sensitive operational information without compromising learning effectiveness.

- We propose an environment-oriented cross-architecture knowledge distillation mechanism that enables efficient knowledge transfer between heterogeneous models based on environmental similarities. Through temperature-regulated soft targets, the mechanism allows small models on resource-constrained devices to achieve competitive performance compared to large models on high-end devices.

- We conduct comprehensive practical evaluation across distributed scheduling domains with real Cloud-Edge-IoT infrastructure, demonstrating the effectiveness and scalability of KD-AFRL using diverse real-world IoT applications spanning different computational characteristics and resource requirements.

The rest of the chapter is organized as follows. Section 6.2 reviews related work. Section 6.3 presents the system model and problem formulation. Section 6.4 details the KD-AFRL framework. Section 6.5 presents experimental evaluation. Section 6.6 concludes the chapter.

6.2 Related Work

In this section, we review existing DRL techniques for IoT scheduling, categorizing them into centralized and distributed approaches, and identify research gaps through qualitative comparison.

6.2.1 Centralized DRL for IoT Scheduling

Tang et al. [283] proposed RASO based on Deep Q-Network (DQN) for collaborative task offloading in Mobile Edge Computing (MEC) networks. The method employs spatial indexing and fine-grained task recombination to minimize offloading delay and energy consumption. Zhu et al. [284] proposed a Proximal Policy Optimization (PPO)-based approach with hybrid actor-critic networks for joint wireless charging and computation offloading in wireless-powered multi-access edge computing (WP-MEC). The objective is to maximize utility characterized by wireless devices' residual energy and social relationship strength. Fan et al. [285] proposed a Softmax Deep Double Deterministic Policy Gradients (DDPG)-based resource orchestration scheme for vehicle collaborative networks. The method aims at minimizing total cost involving latency and energy consumption. Chen et al. [235] proposed DODQ based on DQN for cloud-edge computing environments. The approach models mobile applications as DAGs to adaptively handle dynamic resource changes and parallel task scheduling without presetting task priorities. Wang et al. [26] proposed DRLIS based on PPO for IoT application scheduling in heterogeneous edge/fog computing environments. The method optimizes response time and load balancing for DAG-based applications. Hsieh et al. [234] investigated the task assignment problem in cooperative MEC networks, developing and comparing Double-DQN, Policy Gradient, and Actor-Critic algorithms for task optimization. The results demonstrated that the Actor-Critic approach performed best in optimizing delay under dynamic MEC environments. Zhao et al. [233] proposed MESON based on DDPG for urban vehicular edge computing. The scheme incorporates vehicle mobility detection and task priority determination to minimize average response time and energy consumption. Chi et al. [286] proposed a scheme that combines Double Dueling DQN (D3QN) and prioritized experience for task offloading in edge-assisted Industrial

Internet of Things (IIoT). The scheme reduces average task cost and improves task completion rate by enhancing action selection accuracy and convergence speed.

6.2.2 Distributed DRL for IoT Scheduling

Wu et al. [287] proposed a distributed DQN-based algorithm with temporal convolution sequence network (TCSN) for proactive caching in 6G cloud-edge collaboration computing. The distributed approach maximizes edge hit ratio while minimizing content access latency and traffic cost. Zhao et al. [288] proposed ADTO, an Asynchronous Advantage Actor-Critic (A3C)-based solution for multi-hop task offloading in RSU-assisted Internet of Vehicles (IoV) networks. The approach establishes mobility models and forwarding vehicle selection mechanisms to minimize task delay. Wang et al. [54] proposed a Transformer-enhanced Distributed DRL technique (TF-DDRL) based on Importance Weighted Actor-Learner Architectures (IMPALA) for scheduling heterogeneous IoT applications in edge and cloud environments. The approach incorporates prioritized experience replay and off-policy correction to reduce response time, energy consumption, and monetary cost. Zhou et al. [289] proposed DRLCOSCM using A3C algorithm for three-tier mobile cloud-edge computing. The approach minimizes cloud service cost while meeting delay requirements of mobile users. Zhang et al. [27] proposed LsiA3CS based on A3C for task scheduling in IIoT. The approach incorporates Markov game modeling and heuristic guidance to reduce task completion times. Ju et al. [239] proposed an A3C-based energy-efficiency secure offloading (EESO) scheme for vehicular edge computing networks. The approach aims at minimizing system energy consumption while ensuring security. Liu et al. [257] proposed an A3C-based algorithm for collaborative task computing and on-demand resource allocation in vehicular edge computing. The approach maximizes system utility through optimal task and resource scheduling policy considering service migration and available vehicle resources. Shen et al. [124] proposed AFO, an asynchronous federated PPO-based task offloading algorithm for dependency-aware UAV-assisted vehicular networks. The approach enhances data diversity to minimize average task execution delay and energy consumption.

6.2.3 A Qualitative Comparison

To systematically analyze the existing literature and identify research gaps, we conduct a comprehensive qualitative comparison of related works presented in Table 6.1, evaluating them across four critical dimensions: application properties, system properties, technique properties, and evaluation methodology.

Comparative Analysis Dimensions

Application Properties evaluate workload complexity through task number and dependency. System Properties assess infrastructure scope including application types, computing environments, heterogeneity, and multi-domain support. Technique Properties analyzes algorithmic approaches (centralized vs. distributed), DRL techniques, optimization objectives, and adaptive architecture support. Evaluation Methodology distinguishes simulation-based from practical deployment evaluation.

Research Gap Identification

Based on our systematic analysis, we identify four fundamental research gaps that existing literature fails to address:

Gap 1 - Limited Application Realism: Only 5 works support task dependencies, and merely 2 evaluate real IoT applications, indicating a substantial disconnect between research assumptions and practical deployment requirements. Real-world IoT applications typically exhibit complex interdependencies and diverse computational characteristics that are not captured in simplified single-task or synthetic workload scenarios employed by most existing works.

Gap 2 - Multi-Domain Coordination: All existing works lack multi-domain support, despite real-world IoT deployments spanning multiple autonomous domains. This design limitation results in isolated scheduling systems, missing opportunities for cross-domain resource optimization and collaborative decision-making.

Gap 3 - Adaptive Architecture Generation: All existing works employ fixed neural network architectures, completely ignoring the substantial computational heterogene-

ity from IoT devices to cloud servers. This architectural rigidity leads to either severe resource underutilization on high-performance devices or computational overload on resource-constrained devices.

Gap 4 - Evaluation Limitations: 14 works rely solely on simulation-based evaluation, lacking practical deployment verification and failing to capture real-world operational complexities and uncertainties.

These identified gaps collectively motivate the development of KD-AFRL, which systematically addresses the core challenges of heterogeneous multi-domain IoT scheduling through resource-aware adaptive architecture generation, privacy-preserving environment-clustered federated learning, and environment-oriented cross-architecture knowledge distillation.

Table 6.1: A qualitative comparison of our work with existing related works

Work	Application Properties		System Properties					Technique Properties						Evaluation
	Task Number	Dependency	IoT Device Layer		Edge/Cloud Layer		Multi-Domain	Main Technique		Optimization Objectives			Resource-aware	
			Real Applications	Request Type	Computing Environment	Heterogeneity		Type	Algorithm	Time	Energy	Multi Objective	Adaptive Architecture	
Tang et al. [283]	Multiple	Independent	◐	Homogeneous	Edge	Heterogeneous	×	Centralized	DQN	✓	✓	✓	×	Simulation
Zhu et al. [284]	Single	Independent	○	Homogeneous	Edge	Homogeneous	×		PPO	×	✓	✓	×	Simulation
Fan et al. [285]	Single	Independent	◐	Homogeneous	Edge	Heterogeneous	×		DDPG	✓	✓	✓	×	Simulation
Chen et al. [235]	Multiple	Dependent	◐	Heterogeneous	Edge and Cloud	Heterogeneous	×		DQN	✓	×	×	×	Simulation
Wang et al. [26]	Multiple	Dependent	●	Heterogeneous	Edge and Cloud	Heterogeneous	×		PPO	✓	×	✓	×	Practical
Hsieh et al. [234]	Single	Independent	◐	Heterogeneous	Edge and Cloud	Heterogeneous	×		Actor-Critic	✓	×	×	×	Simulation
Zhao et al. [233]	Multiple	Dependent	◐	Heterogeneous	Edge	Heterogeneous	×		DDPG	✓	✓	✓	×	Simulation
Chi et al. [286]	Single	Independent	○	Homogeneous	Edge	Homogeneous	×		DQN	✓	✓	✓	×	Simulation
Wu et al. [287]	Single	Independent	◐	Homogeneous	Edge and Cloud	Homogeneous	×	Distributed	DQN	✓	×	✓	×	Simulation
Zhao et al. [288]	Single	Independent	○	Homogeneous	Edge and Cloud	Homogeneous	×		A3C	✓	×	×	×	Simulation
Wang et al. [54]	Multiple	Dependent	●	Heterogeneous	Edge and Cloud	Heterogeneous	×		IMPALA	✓	✓	✓	×	Practical
Zhou et al. [289]	Single	Independent	○	Homogeneous	Edge and Cloud	Homogeneous	×		A3C	✓	×	✓	×	Simulation
Zhang et al. [27]	Single	Independent	◐	Heterogeneous	Edge and Cloud	Heterogeneous	×		A3C	✓	×	×	×	Simulation
Ju et al. [239]	Single	Independent	◐	Heterogeneous	Edge	Heterogeneous	×		A3C	×	✓	×	×	Simulation
Liu et al. [257]	Single	Independent	◐	Homogeneous	Edge	Homogeneous	×		A3C	✓	×	×	×	Simulation
Shen et al. [124]	Multiple	Dependent	◐	Heterogeneous	Edge and Cloud	Heterogeneous	×		PPO + Fed Learning	✓	✓	✓	×	Simulation
KD-AFRL	Multiple	Dependent	●	Heterogeneous	Edge and Cloud	Heterogeneous	✓		Actor-Critic + Fed Learning + KD	✓	✓	✓	✓	Practical

●: Real IoT Application and Deployment, ◐: Simulated IoT Application, ○: Random

6.3 System Model and Problem Formulation

This section first describes the topology of the Cloud-Edge-IoT multi-domain computing architecture. Next, we tackle the scheduling of IoT applications by formulating it as an optimization problem, aiming at reducing application completion time, system energy consumption, and weighted cost.

6.3.1 System Model

Fig. 6.1 depicts a hierarchical Cloud-Edge-IoT computing architecture with distributed scheduling domains. Our system comprises a heterogeneous computing infrastructure spanning cloud servers, edge nodes, and IoT devices, collectively forming a continuous computing environment.

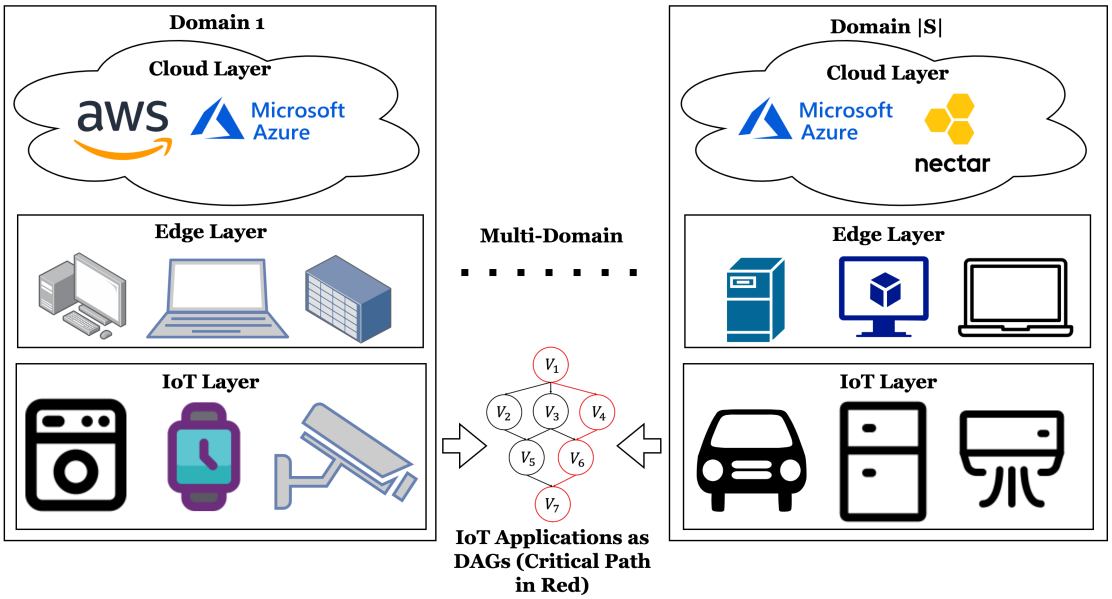


Figure 6.1: The hierarchical Cloud-Edge-IoT computing architecture with distributed scheduling domains.

The computing infrastructure consists of $|\mathcal{N}|$ servers, defined as $\mathcal{N} = \{\mathcal{N}_k | 1 \leq k \leq |\mathcal{N}|\}$. To account for server heterogeneity, each server $\mathcal{N}_k$ is characterized by specific resource capabilities, including available CPU frequency $Freq(\mathcal{N}_k)$ measured in MHz and available memory $Ram(\mathcal{N}_k)$ measured in GB. The network connectivity between servers is defined by propagation time $\mathcal{P}_{\mathcal{N}_l,\mathcal{N}_k}$ (ms) and data transmission rate $\mathcal{B}_{\mathcal{N}_l,\mathcal{N}_k}$ (b/s).

A distinguishing feature of our system model is the distribution of scheduling responsibility across multiple domains. We define a set of schedulers $\mathcal{S} = \{\mathcal{S}_m | 1 \leq m \leq |\mathcal{S}|\}$, where each scheduler $\mathcal{S}_m$ operates in a distinct environment with unique characteristics. Each scheduler $\mathcal{S}_m$ is responsible for managing a subset of computational resources $\mathcal{N}_m \subseteq \mathcal{N}$ and handling workloads within its domain. These domains may be

geographically distributed, administratively separated, or functionally specialized, each exhibiting unique environmental dynamics that influence scheduling decisions.

Within each distributed domain, we consider a set of IoT applications $\mathcal{A} = \{\mathcal{A}_i | 1 \leq i \leq |\mathcal{A}|\}$ that require execution across the available resources. Each application $\mathcal{A}_i$ consists of multiple interdependent tasks denoted as $\mathcal{A}_i = \{\mathcal{A}_i^j | 1 \leq j \leq |\mathcal{A}_i|\}$. As illustrated in Fig. 6.1, each application is modeled as a DAG, where vertices $\mathcal{V}_j = \mathcal{A}_i^j$ represent individual tasks, and edges $\mathcal{E}_{j,k}$ represent data dependencies between tasks $\mathcal{V}_j$ and $\mathcal{V}_k$, indicating that successor tasks can only begin execution after their predecessors complete. The critical path, denoted as $CP(\mathcal{A}_i)$ and highlighted in red, represents the path with the highest cumulative cost from entry to exit tasks.

6.3.2 Problem Formulation

With multiple scheduling domains managing different subsets of resources, the scheduling process is distributed across various schedulers. For each task $\mathcal{A}_i^j$, we define its scheduling configuration as a tuple:

$$\mathcal{C}_i^j = (\mathcal{N}_k, \mathcal{S}_m), \quad k \in \{1, \ldots, |\mathcal{N}|\}, \quad m \in \{1, \ldots, |\mathcal{S}|\}, \tag{6.1}$$

where $\mathcal{N}_k$ denotes the server assigned to execute the task, and $\mathcal{S}_m$ represents the scheduler responsible for making this assignment. This configuration must satisfy the domain constraint: $\mathcal{N}_k \in \mathcal{N}_m$, ensuring that schedulers only allocate resources within their authority.

The scheduling configuration $\mathcal{C}_i$ for the application $\mathcal{A}_i$ encompasses all task-level configurations and is defined as:

$$\mathcal{C}_i = \{\mathcal{C}_i^j | 1 \leq j \leq |\mathcal{A}_i|\}, \tag{6.2}$$

where $|\mathcal{A}_i|$ represents the total number of tasks in application $\mathcal{A}_i$.

The execution model for applications preserves the dependency constraints represented in the DAG structure. Each task cannot begin execution until all its predecessor tasks complete. We use $PR(\mathcal{A}_i^j)$ to denote the set of predecessor tasks of task $\mathcal{A}_i^j$ and use

$CP(\mathcal{A}_i^j)$ to indicate whether task $\mathcal{A}_i^j$ is located on the critical path of the application $\mathcal{A}_i$.

Application Completion Time Model

Given the scheduling configuration $\mathcal{C}_i^j = (\mathcal{N}_k, \mathcal{S}_m)$ for task $\mathcal{A}_i^j$, we define the task completion time model $TCT(\mathcal{C}_i^j)$ comprising two primary components: the communication latency model $T^{cl}(\mathcal{C}_i^j)$ and the processing duration model $T^{pd}(\mathcal{C}_i^j)$:

$$TCT(\mathcal{C}_i^j) = T^{cl}(\mathcal{C}_i^j) + T^{pd}(\mathcal{C}_i^j). \tag{6.3}$$

The communication latency model $T^{cl}(\mathcal{C}_i^j)$ represents the maximum time required for data dependencies to be satisfied before task execution:

$$T^{cl}(\mathcal{C}_i^j) = \max_{\mathcal{A}_i^k \in PR(\mathcal{A}_i^j)} T^{cl}_{\mathcal{C}_i^k, \mathcal{C}_i^j}, \tag{6.4}$$

where $T^{cl}_{\mathcal{C}_i^k, \mathcal{C}_i^j}$ indicates the time needed to transfer data from the server executing predecessor task $\mathcal{A}_i^k$ to the server executing task $\mathcal{A}_i^j$. This time depends on both the data transfer time $T^{dt}_{\mathcal{C}_i^k, \mathcal{C}_i^j}$ and the network propagation time $\mathcal{P}_{\mathcal{N}_l, \mathcal{N}_k}$ between the respective servers:

$$T^{cl}_{\mathcal{C}_i^k, \mathcal{C}_i^j} = \begin{cases} T^{dt}_{\mathcal{C}_i^k, \mathcal{C}_i^j} + \mathcal{P}_{\mathcal{N}_l, \mathcal{N}_k} & \text{if servers differ,} \\ 0 & \text{if same server,} \end{cases} \tag{6.5}$$

where $\mathcal{N}_l$ and $\mathcal{N}_k$ are the servers in configurations $\mathcal{C}_i^k$ and $\mathcal{C}_i^j$ respectively. The data transfer time $T^{dt}_{\mathcal{C}_i^k, \mathcal{C}_i^j}$ is calculated as:

$$T^{dt}_{\mathcal{C}_i^k, \mathcal{C}_i^j} = \frac{DV_{\mathcal{C}_i^k, \mathcal{C}_i^j}(\mathcal{A}_i^j)}{\mathcal{B}_{\mathcal{N}_l, \mathcal{N}_k}}, \tag{6.6}$$

where $DV_{\mathcal{C}_i^k, \mathcal{C}_i^j}(\mathcal{A}_i^j)$ denotes the data volume for task $\mathcal{A}_i^j$ transmitted from the server in configuration $\mathcal{C}_i^k$ to the server in configuration $\mathcal{C}_i^j$, and $\mathcal{B}_{\mathcal{N}_l, \mathcal{N}_k}$ represents the data transmission rate between these servers.

The processing duration model $T^{pd}(\mathcal{C}_i^j)$ defines the time required to execute task $\mathcal{A}_i^j$

on the assigned server and is calculated as:

$$T^{pd}(\mathcal{C}_i^j) = \frac{CC(\mathcal{A}_i^j)}{Freq(\mathcal{N}_k)}, \tag{6.7}$$

where $CC(\mathcal{A}_i^j)$ denotes the computational complexity (in required CPU cycles) for executing task $\mathcal{A}_i^j$ and $Freq(\mathcal{N}_k)$ represents the CPU frequency of the assigned server $\mathcal{N}_k$ in configuration $\mathcal{C}_i^j$.

The completion time $CT(\mathcal{C}_i)$ for the entire application $\mathcal{A}_i$ is expressed as:

$$CT(\mathcal{C}_i) = \sum_{j=1}^{|\mathcal{A}_i|} (TCT(\mathcal{C}_i^j) \times CP(\mathcal{A}_i^j)), \tag{6.8}$$

where $CP(\mathcal{A}_i^j)$ is the critical path indicator: 1 if task $\mathcal{A}_i^j$ is on the critical path of application $\mathcal{A}_i$, and 0 otherwise.

For each scheduler $\mathcal{S}_m$, let $\mathcal{A}_{\mathcal{S}_m} = \{\mathcal{A}_i | \mathcal{A}_i \text{ is assigned to } \mathcal{S}_m\}$ denote the set of applications assigned to that scheduler. The global optimization objective is to minimize the sum of application completion times across all scheduling domains:

$$\min_{\mathcal{C}} \sum_{m=1}^{|\mathcal{S}|} \sum_{\mathcal{A}_i \in \mathcal{A}_{\mathcal{S}_m}} CT(\mathcal{C}_i), \tag{6.9}$$

where $\mathcal{C}$ represents the collective scheduling decisions across all domains.

System Energy Consumption Model

The energy consumption function $E(\cdot)$ characterizes the total energy required to execute applications across heterogeneous computing resources in a distributed environment. For a task $\mathcal{A}_i^j$ with scheduling configuration $\mathcal{C}_i^j = (\mathcal{N}_k, \mathcal{S}_m)$, the total energy consumption $E(\mathcal{C}_i^j)$ consists of two components: the processing energy $E^{pd}(\mathcal{C}_i^j)$ consumed during task execution, and the communication energy $E^{cm}(\mathcal{C}_i^j)$ consumed when transmitting data to successor tasks:

$$E(\mathcal{C}_i^j) = E^{pd}(\mathcal{C}_i^j) + (E^{cm}(\mathcal{C}_i^j) \times ED(\mathcal{A}_i^j)), \tag{6.10}$$

where $ED(\mathcal{A}_i^j)$ is a binary indicator: 0 if $\mathcal{A}_i^j$ is a terminal task with no successors, and 1 otherwise.

The processing energy model $E^{pd}(\mathcal{C}_i^j)$ quantifies the energy consumed by the server when executing the task:

$$E^{pd}(\mathcal{C}_i^j) = T^{pd}(\mathcal{C}_i^j) \times P^{pd}(\mathcal{N}_k), \tag{6.11}$$

where $T^{pd}(\mathcal{C}_i^j)$ is the processing duration obtained from the completion time model, and $P^{pd}(\mathcal{N}_k)$ represents the power consumption rate of server $\mathcal{N}_k$ during computation.

The communication energy model $E^{cm}(\mathcal{C}_i^j)$ accounts for the energy expended when transmitting output data to the servers hosting successor tasks:

$$E^{cm}(\mathcal{C}_i^j) = \sum_{\mathcal{A}_i^l \in SU(\mathcal{A}_i^j)} \frac{DV_{\mathcal{C}_i^j, \mathcal{C}_i^l}(\mathcal{A}_i^j)}{\mathcal{B}_{\mathcal{N}_k, \mathcal{N}_q}} \times P^{cm}(\mathcal{N}_k) \times \delta(\mathcal{N}_k, \mathcal{N}_q), \tag{6.12}$$

where:

- $SU(\mathcal{A}_i^j)$ denotes the set of successor tasks of task $\mathcal{A}_i^j$
- $\mathcal{C}_i^l = (\mathcal{N}_q, \mathcal{S}_n)$ is the scheduling configuration of a successor task $\mathcal{A}_i^l$
- $DV_{\mathcal{C}_i^j, \mathcal{C}_i^l}(\mathcal{A}_i^j)$ represents the volume of data transmitted
- $\mathcal{B}_{\mathcal{N}_k, \mathcal{N}_q}$ is the data transmission rate between servers
- $P^{cm}(\mathcal{N}_k)$ denotes the power consumption rate of server $\mathcal{N}_k$ during data transmission
- $\delta(\mathcal{N}_k, \mathcal{N}_q)$ is a binary indicator: 0 if $\mathcal{N}_k$ and $\mathcal{N}_q$ are the same server, and 1 otherwise

The transmission power $P^{cm}(\mathcal{N}_k)$ can be modeled as a constant value for each server type or as a dynamic parameter that varies based on network conditions and transmission load.

The total energy consumption $E(\mathcal{C}_i)$ for executing application $\mathcal{A}_i$ is calculated by

summing the energy consumption of all constituent tasks:

$$E(\mathcal{C}_i) = \sum_{j=1}^{|\mathcal{A}_i|} E(\mathcal{C}_i^j). \tag{6.13}$$

For each scheduler $\mathcal{S}_m$, the global energy optimization objective is to minimize the sum of application energy consumption across all scheduling domains:

$$\min_{\mathcal{C}} \sum_{m=1}^{|\mathcal{S}|} \sum_{\mathcal{A}_i \in \mathcal{A}_{\mathcal{S}_m}} E(\mathcal{C}_i), \tag{6.14}$$

where $\mathcal{C}$ represents the collective scheduling decisions across all domains.

Weighted Cost Model

To address the multi-objective nature of scheduling in distributed environments, we define a weighted cost model that balances application completion time and system energy consumption. For each application $\mathcal{A}_i$ with scheduling configuration $\mathcal{C}_i$, the weighted cost model $J(\mathcal{C}_i)$ is defined as:

$$J(\mathcal{C}_i) = \alpha_{cost} \times \frac{CT(\mathcal{C}_i) - CT^{min}}{CT^{max} - CT^{min}} + (1 - \alpha_{cost}) \times \frac{E(\mathcal{C}_i) - E^{min}}{E^{max} - E^{min}}, \tag{6.15}$$

where CT^{min} and CT^{max} represent the minimum and maximum achievable completion times, E^{min} and E^{max} represent the minimum and maximum achievable energy consumption values, and $\alpha_{cost} \in [0, 1]$ is the weight parameter that controls the trade-off between completion time and energy efficiency. Normalization is necessary because completion time and energy consumption typically have different scales and units.

From a system-wide perspective, the global weighted cost optimization objective is to minimize the sum of weighted costs across all scheduling domains:

$$\min_{\mathcal{C}} \sum_{m=1}^{|\mathcal{S}|} \sum_{\mathcal{A}_i \in \mathcal{A}_{\mathcal{S}_m}} J(\mathcal{C}_i). \tag{6.16}$$

This optimization problem is subject to the following constraints:

$$\text{s.t.} \quad C1: \ |\{\mathcal{N}_k | (\mathcal{N}_k, \mathcal{S}_m) = \mathcal{C}_i^j\}| = 1, \ \forall \mathcal{C}_i^j \in \mathcal{C}_i \tag{6.17}$$

$$C2: \ DV_{\mathcal{C}_i^k, \mathcal{C}_i^j}(\mathcal{A}_i^j), \mathcal{B}_{\mathcal{N}_l, \mathcal{N}_k} > 0, \ \forall \mathcal{N}_l, \mathcal{N}_k \in \mathcal{N}, \quad \forall \mathcal{A}_i^j \in \mathcal{A}_i \tag{6.18}$$

$$C3: \ Freq(\mathcal{N}_k), Ram(\mathcal{N}_k) > 0, \ \forall \mathcal{N}_k \in \mathcal{N} \tag{6.19}$$

$$C4: \ \sum_{\mathcal{A}_i \in \mathcal{A}_{\mathcal{S}_m}} \sum_{\mathcal{A}_i^j \in \mathcal{A}_i} Ram(\mathcal{A}_i^j) \times SO(\mathcal{A}_i^j, \mathcal{N}_k) < Ram(\mathcal{N}_k), \ \forall \mathcal{N}_k \in \mathcal{N}_m, \forall \mathcal{S}_m \in \mathcal{S} \tag{6.20}$$

$$C5: \ TCT(\mathcal{A}_i^k) \geq TCT(\mathcal{A}_i^j) + T^{cl}_{\mathcal{C}_i^j, \mathcal{C}_i^k}, \forall \mathcal{A}_i^j \in PR(\mathcal{A}_i^k) \tag{6.21}$$

$$C6: \ 0 \leq \alpha_{cost} \leq 1 \tag{6.22}$$

Constraint $C1$ ensures that each task is assigned to exactly one server. $C2$ specifies that data volume and bandwidth must be positive for all task communications. $C3$ defines lower bounds for server resources (CPU frequency and RAM). $C4$ ensures that every server has sufficient RAM to process all tasks scheduled on it, where $SO(\mathcal{A}_i^j, \mathcal{N}_k)$ equals 1 if task $\mathcal{A}_i^j$ is scheduled on server $\mathcal{N}_k$ and 0 otherwise. $C5$ enforces precedence constraints, ensuring that a task can only start after its predecessors complete and the necessary data is transferred. Finally, $C6$ restricts the weight parameter to values between 0 and 1.

This optimization problem presents significant challenges due to its non-convex nature, time-varying constraints, and heterogeneous multi-domain environment. Traditional optimization methods and heuristic algorithms struggle with these complexities, particularly when adapting to dynamic resource availability and handling cross-domain interactions [290]. DRL offers a promising alternative by adapting to changing conditions without requiring complete system knowledge. This approach allows us to formulate scheduling as a sequential decision process that naturally balances immediate and long-term performance goals.

6.3.3 Deep Reinforcement Learning Formulation

To solve the optimization problem formulated in Section 6.3.2, we reformulate it as a DRL problem where each scheduler $\mathcal{S}_m$ learns an optimal scheduling policy through in-

teractions with its environment. We model this as a Markov Decision Process (MDP) defined by the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$, where $\mathcal{S}$ represents the state space, $\mathcal{A}$ is the action space, $\mathcal{P}$ denotes the state transition probability function that determines how the system state evolves after executing an action, defined as $P(s_{t+1}|s_t, a_t) = Pr[S_{t+1} = s_{t+1}|S_t = s_t, A_t = a_t]$, $\mathcal{R}$ is the reward function, and $\gamma \in [0, 1]$ is the discount factor.

State Space

In the multi-domain scheduling environment, the state observation for scheduler $\mathcal{S}_m$ at time step t is defined as a triplet:

$$s_t^m = \{SR_t^m, AT_t^m, QT_t^m\}, \tag{6.23}$$

where:

- $SR_t^m = \{sr_1, sr_2, ..., sr_{|\mathcal{N}_m|}\}$ represents the current status of all servers in domain m, with each sr_i including CPU utilization, CPU frequency, available memory, network load, server type identifier (cloud, edge, or IoT), and bandwidth.
- $AT_t^m = \{at_1, at_2, ..., at_k\}$ describes attributes of the current task to be scheduled, including task ID, application ID, required CPU cycles, memory requirements, and data dependencies (predecessor tasks $PR(\mathcal{A}_i^j)$ and successor tasks $SU(\mathcal{A}_i^j)$).
- $QT_t^m = \{qt_1, qt_2, ..., qt_q\}$ captures the task queue status, including waiting tasks, execution status of predecessor tasks, and scheduling locations of configured tasks.

Action Space

For each scheduler $\mathcal{S}_m$, the action at time step t is defined as:

$$a_t^m = \{n_k | n_k \in \mathcal{N}_m\}. \tag{6.24}$$

This action represents the decision to schedule the current task on server n_k within domain m. The action space is discrete with cardinality equal to the number of available servers in the domain.

Reward Function

The reward function directly implements the optimization objective defined in Section 6.3.2, providing feedback on scheduling decision quality. For scheduler $\mathcal{S}_m$, the reward at time step t is:

$$r_t^m = \begin{cases} -J(\mathcal{C}_i) & \text{if task execution succeeds,} \\ \text{penalty} & \text{if task execution fails.} \end{cases} \tag{6.25}$$

Here, $J(\mathcal{C}_i)$ is the weighted cost model from Eq. 6.15, and the negative sign converts our minimization problem into a reward maximization problem. The penalty for failed executions encourages the agent to avoid decisions that lead to task failures.

Policy and Objective

The agent's policy function defines the probability of selecting action a when observing state s:

$$\pi(a|s) = Pr[A_t = a|S_t = s]. \tag{6.26}$$

The ultimate goal of each scheduler $\mathcal{S}_m$ is to learn a policy π_m that maximizes the expected cumulative discounted reward:

$$\pi_m^* = \arg\max_{\pi_m} \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t^m | \pi_m\right]. \tag{6.27}$$

This DRL formulation transforms our complex optimization problem into an iterative learning process. However, the multi-domain nature introduces several challenges. First, heterogeneity in resource characteristics and workload patterns leads to significantly different state distributions across domains. Second, the non-IID data across domains makes direct policy sharing ineffective. Third, domain-specific constraints and varying computational capabilities necessitate adaptable model architectures. In the following section, we introduce our Knowledge Distillation-empowered Adaptive Federated Reinforcement Learning (KD-AFRL) framework, which addresses these challenges while enabling collaborative learning across domains.

6.4 KD-AFRL Framework

The KD-AFRL framework addresses challenges in multi-domain scheduling by combining federated learning with knowledge distillation techniques.

The KD-AFRL framework operates across M scheduling domains $\{\mathcal{S}_1, \mathcal{S}_2, ..., \mathcal{S}_M\}$, where each domain $\mathcal{S}_m$ possesses different computational resources $\mathcal{N}_m$, and workload characteristics. The framework addresses several key challenges in multi-domain reinforcement learning through three principal mechanisms:

- **Resource-Aware Hybrid Model Architecture Generation** addresses device heterogeneity by adapting DRL model complexity to match the computational capabilities of each domain's devices, enabling participation in federated learning regardless of resource constraints.
- **Privacy-Preserving Environment-Clustered Federated Learning** enhances federated learning by identifying similar domains through differentially-private environmental features, addressing the non-IID challenges in distributed environments.
- **Environment-Oriented Cross-Architecture Knowledge Distillation** complements the federated framework by enabling knowledge transfer between heterogeneous models based on environmental similarities, allowing resource-constrained devices to benefit from complex models without sharing raw data.

It is worth emphasizing that these three mechanisms are not merely integrated but form a synergistic closed loop to resolve the coupled triple heterogeneity. The **Resource-Aware Hybrid Architecture Generation** (Section 6.4.1) solves resource constraints but creates architectural barriers; the **Environment-Oriented Cross-Architecture Knowledge Distillation** (Section 6.4.3) bridges these barriers to enable knowledge transfer; and the **Privacy-Preserving Environment-Clustered Federated Learning** (Section 6.4.2) guides both federated aggregation weights and KD teacher selection based on environmental similarity, preventing negative transfer from incompatible domains. This co-design ensures that heterogeneous domains can collaborate effectively despite their resource, data, and architectural disparities.

6.4.1 Resource-Aware Hybrid Architecture Generation

To address device heterogeneity across domains while enabling collaborative learning, KD-AFRL incorporates a resource-aware model architecture generation mechanism that creates hybrid neural network structures combining standardized shared foundational zones with personalized adaptation zones, as shown in Fig. 6.2.

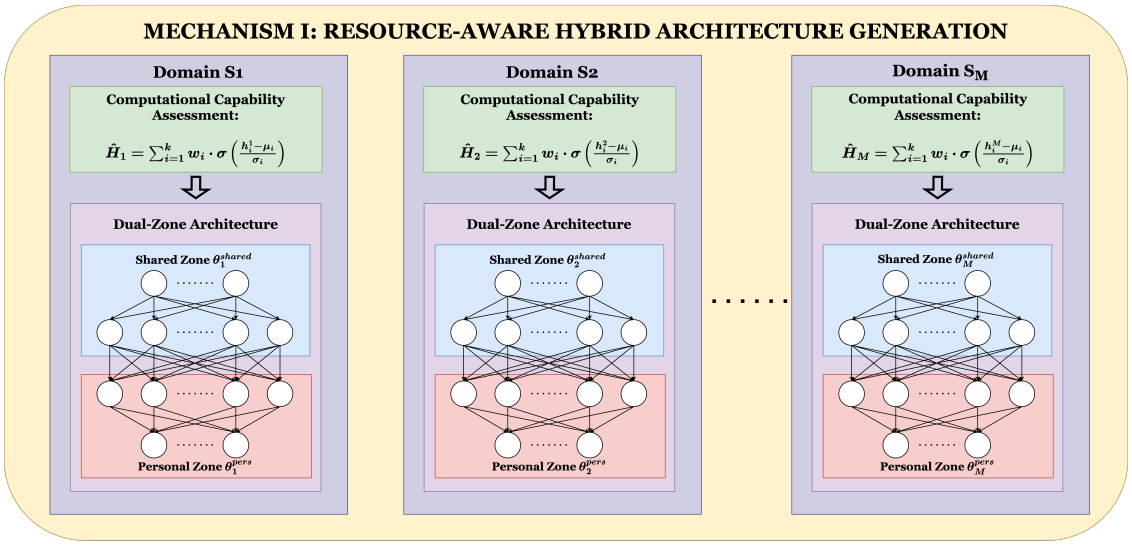


Figure 6.2: Resource-aware hybrid architecture generation mechanism showing computational capability assessment and dual-zone architecture design across heterogeneous domains.

Multi-Dimensional Computational Capability Assessment

To enable precise and adaptive tailoring of model architectures, we quantify each scheduler's computational capability through a comprehensive resource profiling mechanism. For a scheduler in domain $\mathcal{S}_m$, its capability is represented by the vector $H_m \in \mathbb{R}^k$:

$$H_m = [h_1^m, h_2^m, ..., h_k^m]^T, \tag{6.28}$$

where each dimension corresponds to a critical computational resource, such as CPU cores, CPU frequency (GHz), memory capacity (MB), and network bandwidth (Mbps).

To reflect real-time resource availability, we incorporate dynamic utilization effects:

$$h_i^m = \alpha_i \cdot h_i^{m,\text{static}} \cdot (1 - \beta_i \cdot \text{util}_i^m). \tag{6.29}$$

Here, α_i is a weight coefficient for resource i, $h_i^{m,\text{static}}$ denotes the scheduler's inherent capability, β_i is a sensitivity factor capturing the impact of utilization (set to 0 if the resource dimension does not have an associated utilization metric), and util_i^m is the current utilization ratio. This formulation ensures that as resource utilization increases, the effective capacity diminishes proportionally, capturing transient resource constraints in dynamic environments.

We compute a scheduler's comprehensive capability score via a standardized nonlinear aggregation:

$$\hat{H}_m = \sum_{i=1}^{k} w_i \cdot \sigma\left(\frac{h_i^m - \mu_i}{\sigma_i}\right), \tag{6.30}$$

where $\sigma(\cdot)$ is the sigmoid function, μ_i and σ_i are the mean and standard deviation of resource i across all schedulers, and w_i denotes its importance weight. This standardization ensures fair cross-resource comparison, while the sigmoid mapping provides robustness to outliers and maintains sensitivity across diverse resource scales.

Dual-Zone Architecture Design

To enable federated learning while accommodating resource heterogeneity, each domain's model is partitioned into a dual-zone structure with shared foundational zones (participating in federated aggregation) and personalized adaptation zones (retained locally):

$$\theta_m = \{\theta_m^{shared}, \theta_m^{pers}\}. \tag{6.31}$$

The shared foundational zone θ_m^{shared} is selected from K_{arch} predefined architecture types to ensure federated aggregation compatibility, while the personalized adaptation zone θ_m^{pers} is tailored to exploit remaining computational resources for domain-specific optimization. This design bears resemblance to personalized federated learning (pFL) [291, 292], which partitions models into shared and personalized components to address non-

IID data. However, our framework targets a fundamentally different challenge: computational resource heterogeneity. Unlike pFL methods that assume identical shared architectures across clients, we introduce K_{arch} architecture types where only structurally compatible domains aggregate, complemented by cross-architecture knowledge distillation for heterogeneous knowledge transfer.

Shared Foundational Zone Selection: Based on the capability score $\hat{H}_m$, domain m is assigned to one of K_{arch} predefined shared architecture types using a capability-based mapping:

$$\text{Type}_m = \min\left(\lceil \hat{H}_m \cdot K_{arch} \rceil, K_{arch}\right). \tag{6.32}$$

This mapping function divides the normalized capability range [0,1] into K_{arch} equal intervals, where each interval corresponds to a specific shared architecture type. The ceiling function $\lceil \cdot \rceil$ ensures integer type assignment, while the $\min(\cdot, K_{arch})$ operation prevents exceeding the maximum type number. For example, with $K_{arch} = 3$ types and a capability score $\hat{H}_m = 0.7$, the assignment becomes $\text{Type}_m = \min(\lceil 0.7 \times 3 \rceil, 3) = \min(\lceil 2.1 \rceil, 3) = \min(3, 3) = 3$, assigning the domain to the most complex shared architecture type.

Personalized Adaptation Zone Design: The personalized adaptation zone leverages the domain's computational capability to provide domain-specific optimization. The zone's architecture scales proportionally with the capability score:

$$\theta_m^{depth,pers} = \lceil d_{\min} + \alpha_{arch} \cdot \hat{H}_m \cdot (d_{\max} - d_{\min}) \rceil, \tag{6.33}$$

$$\theta_m^{width,pers} = \lceil w_{\min} + \alpha_{arch} \cdot \hat{H}_m \cdot (w_{\max} - w_{\min}) \rceil, \tag{6.34}$$

where $\theta_m^{depth,pers}$ represents the number of layers in the personalized zone, $\theta_m^{width,pers}$ denotes the number of neurons per layer, $d_{\min}$, $d_{\max}$, $w_{\min}$, and $w_{\max}$ define the architectural parameter ranges, and $\alpha_{arch} \in [0, 1]$ is a scaling factor that controls the personalized zone's complexity relative to the domain's total computational capacity.

The detailed procedure for resource-aware hybrid architecture generation is outlined in Algorithm 6.1. The algorithm first assesses each domain's computational capability

through multi-dimensional resource profiling, then assigns appropriate shared architecture types based on capability scores, and finally designs personalized adaptation zones scaled according to available computational resources. This ensures optimal resource utilization while maintaining federated learning compatibility. The computational complexity of this algorithm is $\mathcal{O}(M \cdot k)$, where M is the number of domains and k is the number of resource dimensions. The initialization phase computes resource statistics across all domains in $\mathcal{O}(M \cdot k)$ time, while the architecture generation phase processes each domain independently with constant-time operations per domain. This linear complexity ensures scalability to large-scale multi-domain deployments.

6.4.2 Privacy-Preserving Environment-Clustered Federated Learning

Traditional federated learning assumes that all participating domains share similar data distributions, which rarely holds in practice for multi-domain IoT scheduling environments. Different scheduling domains often exhibit distinct workload patterns, resource characteristics, and environmental dynamics, leading to significant performance degradation when directly applying conventional federated averaging. To address this challenge, KD-AFRL incorporates a privacy-preserving environment-clustered federated learning mechanism that identifies similar scheduling domains while protecting sensitive operational information, and coordinates the training of hybrid dual-zone architectures, as shown in Fig. 6.3.

Privacy-Preserving Environment Feature Modeling

To enable meaningful similarity assessment across scheduling domains, we extract comprehensive environment features that capture the operational characteristics of each domain. For scheduler $\mathcal{S}_m$, the environment feature vector $\mathcal{F}_m \in \mathbb{R}^d$ is constructed from two key dimensions:

Resource Characteristics: These features capture the computational landscape of domain m, including the mean and variance of CPU and memory utilization, inter-server communication bandwidth, and energy consumption (average power per computational unit).

Workload Patterns: These metrics characterize application workload dynamics, including the ratio of average task count to application count, standard deviation of completion times, average task arrival rate, and average dependency ratio in DAG structures.

To protect sensitive operational information while enabling collaborative learning, we implement a differential privacy mechanism by adding calibrated noise to the environment features:

$$\tilde{\mathcal{F}}_m = \mathcal{F}_m + \xi_m, \tag{6.35}$$

where $\xi_m \sim \mathcal{L}(0, 1/\epsilon)$ is zero-mean Laplace noise calibrated to ensure ϵ-differential privacy, with ϵ controlling the privacy level (smaller ϵ provides stronger privacy protection).

Algorithm 6.1: Resource-Aware Hybrid Architecture Generation

Input: Resource capabilities $\{H_m\}_{m=1}^{M}$, architecture parameters K_{arch}, $d_{\min}$, $d_{\max}$, $w_{\min}$, $w_{\max}$, scaling factor α
Output: Final hybrid architectures $\{\theta_m\}_{m=1}^{M}$ with type assignments
/* Initialization */
Initialize predefined shared architectures $\{\theta_k^{shared}\}_{k=1}^{K_{arch}}$ and parameter ranges
Compute resource statistics μ_i and σ_i across all schedulers for each resource dimension
/* Architecture Generation */
for *each scheduler* $m = 1, 2, ..., M$ **do**
  Calculate normalized capability score $\hat{H}_m$
  Assign shared architecture type:
  $\text{Type}_m = \min(\lceil \hat{H}_m \cdot K_{arch} \rceil, K_{arch})$
  Select corresponding shared foundational zone:
  $\theta_m^{shared} = \theta_{\text{Type}_m}^{shared}$
  Design personalized adaptation zone using capability-based scaling:
  $\theta_m^{depth,pers} = \lceil d_{\min} + \alpha \cdot \hat{H}_m \cdot (d_{\max} - d_{\min}) \rceil$
  $\theta_m^{width,pers} = \lceil w_{\min} + \alpha \cdot \hat{H}_m \cdot (w_{\max} - w_{\min}) \rceil$
  Set final architecture:
  $\theta_m = \{\theta_m^{shared}, \theta_m^{pers}\}$
end for
return Final hybrid architectures $\{\theta_m\}_{m=1}^{M}$ with type assignments

Environment Clustering and Similarity Assessment

Given the privacy-preserving environment features $\{\tilde{\mathcal{F}}_1, \tilde{\mathcal{F}}_2, ..., \tilde{\mathcal{F}}_M\}$ from all participating domains, the central server performs K-means clustering to identify groups of sim-

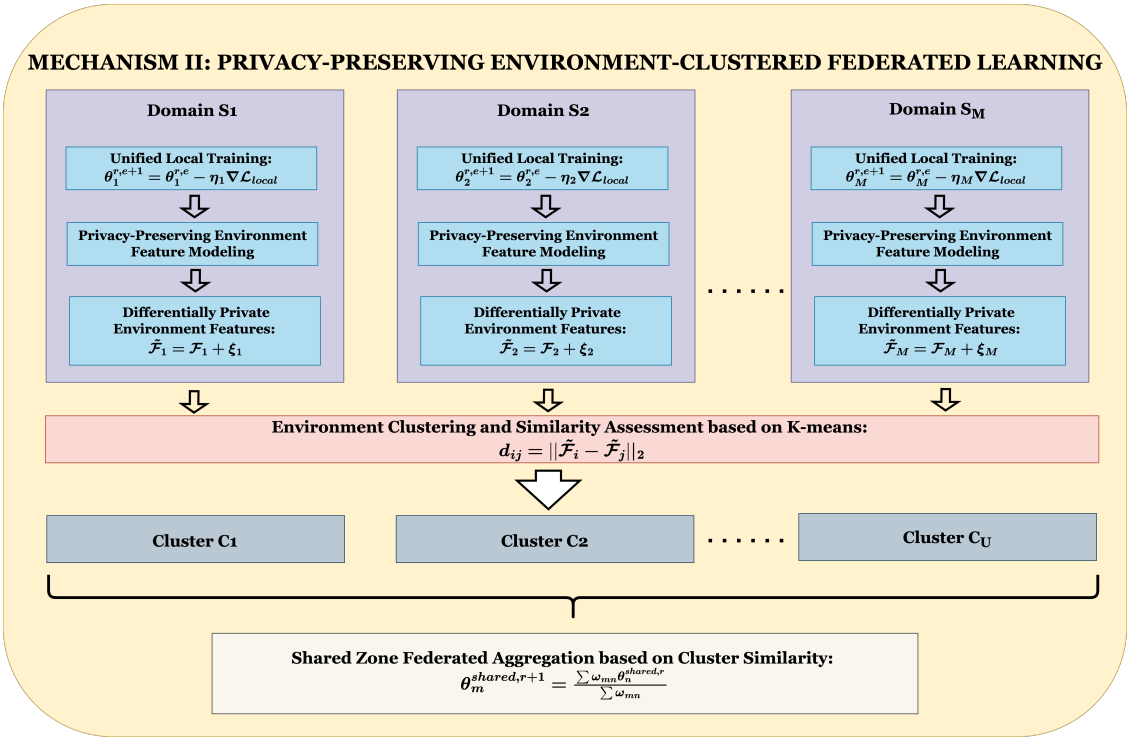


Figure 6.3: Privacy-preserving environment-clustered federated learning mechanism illustrating unified local training, differential privacy protection, environment clustering, and shared zone aggregation.

ilar scheduling environments. The similarity between domains is measured using Euclidean distance:

$$d_{ij} = ||\tilde{\mathcal{F}}_i - \tilde{\mathcal{F}}_j||_2. \tag{6.36}$$

This partitions the domains into U clusters $\{C_1, C_2, ..., C_U\}$. Despite the added Laplace noise for privacy protection, the clustering remains effective as the noise has zero mean and gets averaged out during the iterative process.

To handle the dynamic nature of scheduling environments, we implement a drift-based adaptation mechanism. We periodically compute the drift for each domain:

$$\text{Drift}_m^{(r)} = ||\mathcal{F}_m^{(r)} - \mathcal{F}_m^{(r-\Delta r)}||_2. \tag{6.37}$$

When the maximum drift across all domains exceeds the threshold:

$$\max_m \text{Drift}_m^{(r)} > \tau_{drift}, \tag{6.38}$$

we trigger re-clustering to ensure that domains with substantially changed environments are appropriately reassigned to better-matching clusters.

It is important to clarify that differential privacy and environment clustering serve two independent purposes in our framework. Differential privacy protects sensitive operational information of each domain by adding calibrated Laplace noise to environment features, preventing privacy leakage during the clustering and federated learning process. Environment clustering, operating on these privacy-protected features, addresses the non-IID data challenge by grouping domains with similar workload patterns and resource characteristics, enabling more effective collaboration among compatible domains. These two mechanisms work complementarily but address distinct concerns: privacy protection and heterogeneity mitigation respectively.

Dual-Zone Coordinated Learning Strategy

Based on the clustering results and the hybrid dual-zone architectures, we design a coordinated learning strategy that combines unified local training with shared zone federated aggregation. During local training, both shared and personalized zones are optimized using domain-specific data, while only shared zones participate in cross-domain knowledge sharing through federated aggregation.

Unified Local Training: During local training phases, each domain performs gradient updates on the complete dual-zone model using local data. The training follows standard forward and backward propagation: the input passes through the shared zone then the personalized zone to produce the output. Gradients backpropagate through both zones, updating all parameters $\theta_m = \{\theta_m^{shared}, \theta_m^{pers}\}$ simultaneously. The training process is formalized as:

$$\theta_m^{r,e+1} = \theta_m^{r,e} - \eta_m \nabla_{\theta_m} \mathcal{L}_{local}(\theta_m^{r,e}), \tag{6.39}$$

where $\theta_m^{r,e}$ represents the complete model parameters for domain m at round r and local epoch e, η_m is the learning rate for domain m, and $\mathcal{L}_{local}$ is the local loss function computed on domain m's data using the complete dual-zone model.

Shared Zone Federated Aggregation: After local training, only the shared foundational zones θ_m^{shared} participate in federated aggregation, grouped by their architecture types. Domains with the same shared architecture type (determined by Type_m) can directly aggregate their learned parameters through weighted averaging.

For domain m belonging to environmental cluster C_j, the similarity weight with respect to any other domain n is computed as:

$$\omega_{mn} = \begin{cases} \exp\left(-\frac{d_{mn}^2}{2\sigma_j^2}\right) & \text{if } \text{Type}_n = \text{Type}_m \text{ and } n \in C_j, \\ \beta_{fed} \cdot \exp\left(-\frac{d_{mn}^2}{2\sigma_{global}^2}\right) & \text{if } \text{Type}_n = \text{Type}_m \text{ and } n \notin C_j, \\ 0 & \text{if } \text{Type}_n \neq \text{Type}_m, \end{cases} \tag{6.40}$$

where σ_j^2 is the intra-cluster variance within cluster C_j, σ_{global}^2 is the global variance across all domains, and $\beta_{fed} \in [0, 1]$ is a cross-cluster damping factor that reduces the influence of domains from different environmental clusters. The variance terms σ_j^2 and σ_{global}^2 serve as adaptive scaling factors that normalize distances relative to their respective typical scales (intra-cluster and global), ensuring appropriate weight calibration across different cluster densities and domain distributions.

This weight assignment ensures that: (1) only domains with identical shared architecture types can participate in parameter aggregation; (2) domains within the same environmental cluster have higher influence on each other; and (3) cross-cluster knowledge sharing is maintained but appropriately dampened to prevent interference from dissimilar environments.

The aggregated shared zone parameters $\theta_m^{shared,r+1}$ for domain m at round $r+1$ are computed as:

$$\theta_m^{shared,r+1} = \frac{\sum_{n=1}^{M} \omega_{mn} \cdot \theta_n^{shared,r}}{\sum_{n=1}^{M} \omega_{mn}}. \tag{6.41}$$

The detailed procedure for the privacy-preserving environment-clustered federated learning is outlined in Algorithm 6.2. The algorithm first performs environment clustering with differential privacy protection, then alternates between local training (where

Algorithm 6.2: Privacy-Preserving Environment-Clustered Federated Learning

Input: Environment features $\{\mathcal{F}_m\}_{m=1}^{M}$, hybrid architectures $\{\theta_m\}_{m=1}^{M}$, privacy budget ϵ, number of clusters U, drift threshold τ_{drift}

Output: Cluster assignments $\{C_u\}_{u=1}^{U}$, trained models $\{\theta_m^{shared}, \theta_m^{pers}\}_{m=1}^{M}$

`/* Initial Environment Clustering */`

for *each domain* $m = 1, 2, ..., M$ **do in parallel**
  Extract environment features $\mathcal{F}_m^{(0)}$
  Generate privacy-preserving features:
  $\tilde{\mathcal{F}}_m^{(0)} = \mathcal{F}_m^{(0)} + \xi_m$ where $\xi_m \sim \mathcal{L}(0, 1/\epsilon)$
  Send $\tilde{\mathcal{F}}_m^{(0)}$ to central server
end forpar
Perform K-means clustering on $\{\tilde{\mathcal{F}}_m^{(0)}\}$ to obtain initial clusters $\{C_u\}_{u=1}^{U}$

`/* Dynamic Dual-Zone Federated Learning */`

for *communication round* $r = 1, 2, ..., R$ **do**
  for *each domain* m **do in parallel**
    for *local epoch* $e = 1, 2, ..., E_{local}$ **do**
      Update complete model:
      $\theta_m \leftarrow \theta_m - \eta_m \nabla_{\theta_m} \mathcal{L}_{local}$
    end for
  end forpar
  if $r \bmod T_{fed} = 0$ **then**
    `/* Environment Drift Detection and Re-clustering */`
    for *each domain* m **do in parallel**
      Extract current environment features $\mathcal{F}_m^{(r)}$
      Generate privacy-preserving features:
      $\tilde{\mathcal{F}}_m^{(r)} = \mathcal{F}_m^{(r)} + \xi_m$
      Send $\tilde{\mathcal{F}}_m^{(r)}$ to central server
    end forpar
    for *each domain* m **do**
      Compute drift:
      $\text{Drift}_m^{(r)} = ||\tilde{\mathcal{F}}_m^{(r)} - \tilde{\mathcal{F}}_m^{(r-T_{fed})}||_2$
    end for
    if $\max_m Drift_m^{(r)} > \tau_{drift}$ **then**
      Update clusters:
      Perform K-means clustering on $\{\tilde{\mathcal{F}}_m^{(r)}\}$ to obtain updated $\{C_u\}_{u=1}^{U}$
    end if
    `/* Federated Aggregation */`
    for *each domain* m **do in parallel**
      Send locally-trained θ_m^{shared} to central server
    end forpar
    for *each domain* m **do**
      Compute similarity weights ω_{mn} using current clusters $\{C_u\}$
      Aggregate shared parameters:
      $\theta_m^{shared,r+1} = \frac{\sum_{n=1}^{M} \omega_{mn} \theta_n^{shared,r}}{\sum_{n=1}^{M} \omega_{mn}}$
    end for
    Send aggregated $\theta_m^{shared,r+1}$ back to each domain m
  end if
end for
return Final cluster assignments $\{C_u\}_{u=1}^{U}$, trained models $\{\theta_m^{shared}, \theta_m^{pers}\}_{m=1}^{M}$

both zones learn from local data) and federated aggregation for shared zones (with similarity-weighted parameter averaging based on environmental compatibility). This approach enables collaborative learning among similar domains while preserving domain-specific adaptations and protecting sensitive operational information. The computational complexity comprises four main components. The initial clustering phase requires $\mathcal{O}(M \cdot U \cdot d \cdot I)$ operations, where M is the number of domains, U is the number of clusters, d is the feature dimension, and I is the number of K-means iterations. The local training phase has complexity $\mathcal{O}(R \cdot M \cdot E_{local} \cdot C_{grad})$ across R communication rounds with E_{local} local epochs, where C_{grad} represents the gradient computation cost. The federated aggregation phase, executed $\lfloor R/T_{fed} \rfloor$ times, involves similarity weight computation with $\mathcal{O}(M^2)$ operations (or $\mathcal{O}(M^2 d)$ if based on feature distances) and weighted parameter averaging with $\mathcal{O}(M^2 \cdot |\theta^{shared}|)$ operations, yielding a total of $\mathcal{O}(\lfloor R/T_{fed} \rfloor \cdot M^2 \cdot |\theta^{shared}|)$. The environment drift detection incurs $\mathcal{O}(\lfloor R/T_{fed} \rfloor \cdot M \cdot d)$ cost, with conditional re-clustering adding $\mathcal{O}(M \cdot U \cdot d \cdot I)$ per occurrence when drift exceeds the threshold τ_{drift}. The communication complexity is $\mathcal{O}(\lfloor R/T_{fed} \rfloor \cdot M \cdot (2|\theta^{shared}| + d) + M \cdot d)$ for bidirectional shared parameter transmission and privacy-preserving feature uploads. In summary, the overall computational complexity is $\mathcal{O}(R \cdot M \cdot E_{local} \cdot C_{grad} + \lfloor R/T_{fed} \rfloor \cdot M^2 \cdot |\theta^{shared}| + (1 + N_{re}) \cdot M \cdot U \cdot d \cdot I)$, where N_{re} denotes the number of re-clustering events. In practice, since $C_{grad} \gg |\theta^{shared}| > d$ and M is typically small, the local training term dominates, simplifying the effective complexity to $\mathcal{O}(R \cdot M \cdot E_{local} \cdot C_{grad})$. The additional overhead from environment clustering and similarity-weighted aggregation remains marginal relative to the gradient computation cost. Regarding the correctness of dual-zone architecture training, this partial aggregation approach has theoretical grounding in personalized federated learning literature. Studies such as [291] and [292] have shown that, models with shared and personalized components can achieve convergence guarantees under non-IID data distributions. The key principle is that allowing personalized components to remain local helps mitigate domain-specific biases in the shared component while facilitating knowledge transfer across compatible domains. Our dual-zone training strategy follows this principle: gradients flow through both zones during local optimization, but only the shared zone participates in cross-domain aggregation.

6.4.3 Environment-Oriented Cross-Architecture Knowledge Distillation

While privacy-preserving environmental clustering federated learning enables collaboration among domains with identical shared architecture types, domains with different architecture types cannot directly participate in parameter aggregation due to structural incompatibility, preventing resource-constrained domains from benefiting from knowledge gained by more capable domains. To address this challenge, KD-AFRL introduces an environment-oriented cross-architecture knowledge distillation mechanism that enables knowledge transfer between domains with heterogeneous model architectures while considering environmental similarities, as shown in Fig. 6.4. This mechanism enables effective knowledge transfer between models with heterogeneous architectures, allowing small models on resource-constrained devices to achieve near-comparable performance to larger models.

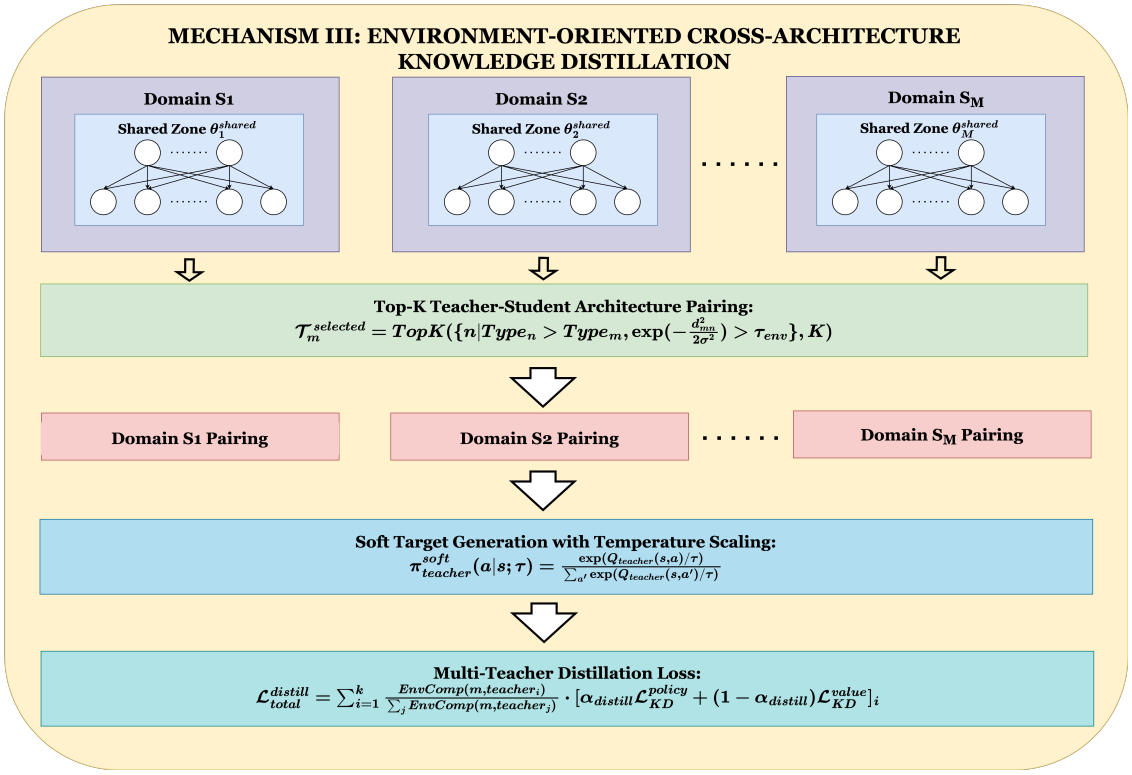


Figure 6.4: Environment-oriented cross-architecture knowledge distillation mechanism demonstrating Top-K teacher-student pairing and multi-teacher distillation process.

Top-K Teacher-Student Architecture Pairing Strategy

The knowledge distillation process operates through teacher-student relationships, where domains with more complex architectures (teachers) transfer knowledge to domains with simpler architectures (students). Given K_{arch} predefined shared architecture types ranked by complexity, we establish a Top-K teacher-student pairing strategy.

For a domain $\mathcal{S}_m$ with architecture type Type_m, its potential teacher set $\mathcal{T}_m$ includes all domains with higher complexity architecture types in the same or similar environmental clusters:

$$\mathcal{T}_m = \{n \mid \text{Type}_n > \text{Type}_m, \text{EnvComp}(m,n) > \tau_{env}\}, \tag{6.42}$$

where $\text{EnvComp}(m,n)$ denotes the environmental compatibility between domains m and n:

$$\text{EnvComp}(m,n) = \exp\left(-\frac{d_{mn}^2}{2\sigma_{global}^2}\right), \tag{6.43}$$

and $\tau_{env} \in [0,1]$ is a threshold parameter controlling the minimum environmental similarity required for teacher-student pairing.

The server automatically establishes distillation pairing relationships based on model architecture complexity. Based on complexity ranking, each student model selects the Top-K models with higher complexity as its teacher set. The teacher selection strategy is defined as:

$$\mathcal{T}_m^{selected} = \text{TopK}\left(\mathcal{T}_m, K\right). \tag{6.44}$$

Soft Target Knowledge Distillation Mechanism

The core challenge of cross-architecture knowledge distillation lies in transferring strategic knowledge between models with different structural configurations. We design a soft target-based dual-level knowledge distillation method that achieves cross-architecture knowledge transfer through the transmission of policy distributions and value estimates.

Temperature-Regulated Soft Target Generation: To capture the complete decision

knowledge of teacher models, we employ temperature regulation techniques to generate soft targets. For a given state s_t, the teacher model's raw Q-value output is $Q_{teacher}(s_t, a)$. By introducing a temperature parameter τ_{temp}, we convert these Q-values into a softened probability distribution:

$$\pi^{soft}_{teacher}(a|s_t;\tau_{temp}) = \frac{\exp(Q_{teacher}(s_t,a)/\tau_{temp})}{\sum_{a'\in\mathcal{A}}\exp(Q_{teacher}(s_t,a')/\tau_{temp})}. \tag{6.45}$$

The temperature parameter τ_{temp} controls the smoothness of the distribution and the granularity of knowledge transfer: when $\tau_{temp} > 1$, the probability distribution becomes more smooth, reducing the dominance of optimal actions and enabling the student model to learn the teacher's relative preferences for suboptimal actions, which helps transfer richer decision knowledge; when $\tau_{temp} = 1$, it degrades to the standard softmax distribution.

Policy Knowledge Distillation: The student model learns the policy distribution by minimizing the Kullback-Leibler divergence with the teacher's soft targets:

$$\begin{aligned}\mathcal{L}^{policy}_{KD} = \mathbb{E}_{s_t\sim\mathcal{D}_{student}}\Big[D_{KL}\big(\pi^{soft}_{teacher}(a|s_t;\tau_{temp}) \\ \| \ \pi_{student}(a|s_t;\tau_{temp})\big)\Big] \times \tau^2_{temp},\end{aligned} \tag{6.46}$$

where the τ^2_{temp} term is a gradient compensation factor to maintain consistent loss scaling across different temperature values.

Value Function Knowledge Transfer: In addition to policy distributions, value functions contain important estimation information about long-term rewards. The value function distillation loss is defined as:

$$\mathcal{L}^{value}_{KD} = \mathbb{E}_{s_t\sim\mathcal{D}_{student}}\left[\left(V_{teacher}(s_t) - V_{student}(s_t)\right)^2\right]. \tag{6.47}$$

Adaptive Distillation Loss Balancing: To balance the contributions of policy distillation and value function distillation, the complete distillation loss function for student domains is:

$$\mathcal{L}_{distill} = \alpha_{distill}\cdot\mathcal{L}^{policy}_{KD} + (1-\alpha_{distill})\cdot\mathcal{L}^{value}_{KD}, \tag{6.48}$$

where $\alpha_{distill} \in [0, 1]$ is a balancing parameter.

Environmental Compatibility-Oriented Multi-Teacher Distillation Strategy

Considering the environmental differences between different domains, we design an environmental compatibility-based multi-teacher distillation strategy.

Teacher Weight Assignment: Considering the environmental differences between different domains, we design an environmental compatibility-based teacher weight assignment strategy. For student model m, the weight of its i-th teacher is based on environmental similarity:

$$w_i = \frac{\text{EnvComp}(m, teacher_i)}{\sum_{j=1}^{k} \text{EnvComp}(m, teacher_j)}. \tag{6.49}$$

This design ensures that teachers with similar environments receive higher weights, as knowledge from teacher models in similar environments is more easily transferred to student models.

Training Episode Allocation: The distillation training episodes for different teachers are adaptively allocated based on their environmental compatibility:

$$E_i = E_{base} \cdot (1 + \beta_{distill} \cdot \text{EnvComp}(m, teacher_i)), \tag{6.50}$$

where E_{base} is the base training episodes and $\beta_{distill}$ is an adjustment parameter. This design ensures that teachers with similar environments receive more training episodes.

Multi-Teacher Loss Aggregation: Based on environmental compatibility weights, the total distillation loss is:

$$\mathcal{L}_{total}^{distill} = \sum_{i=1}^{k} w_i \cdot \mathcal{L}_{distill}^{i}, \tag{6.51}$$

where $\mathcal{L}_{distill}^{i}$ is the distillation loss from the i-th teacher.

The complete cross-architecture knowledge distillation process is outlined in Algorithm 6.3. The algorithm first establishes teacher-student pairings based on architecture complexity and environmental compatibility, then performs multi-teacher distillation where each student learns from multiple teachers with weights and training episodes

allocated according to environmental similarity. This ensures that students prioritize learning from teachers in similar environments while still benefiting from diverse architectural knowledge. The computational complexity consists of two phases. The teacher-student pairing phase requires $\mathcal{O}(M \log M + M^2)$ operations, where $\mathcal{O}(M \log M)$ accounts for sorting domains by architecture complexity and $\mathcal{O}(M^2)$ accounts for computing pairwise environmental compatibility and performing Top-K selection for each student. The multi-teacher distillation phase has complexity $\mathcal{O}(R_{KD} \cdot (M-1) \cdot K \cdot E_{base} \cdot C_{distill})$, where R_{KD} is the number of distillation rounds, $(M-1)$ is the number of student models (excluding the most complex model), K is the maximum number of teachers per student, E_{base} is the base training episodes, and $C_{distill}$ represents the per-episode distillation cost including forward passes through both teacher and student networks and gradient computation. Note that the variable episode count $E_i = E_{base} \cdot (1 + \beta_{distill} \cdot \text{EnvComp}(m_j, m_i))$ introduces a bounded multiplicative factor $(1 + \beta_{distill})$ that is absorbed into the asymptotic notation. The parallel execution of distillation across students (indicated by `ForPar`) reduces the practical wall-clock time to $\mathcal{O}(R_{KD} \cdot K \cdot E_{base} \cdot C_{distill})$ when sufficient computational resources are available, making the framework scalable to large multi-domain deployments.

6.5 Performance Evaluation

This section introduces the experimental setup, hyperparameter tuning configurations, and comprehensive experiments to evaluate KD-AFRL performance.

6.5.1 Experiment Setup

This subsection describes the distributed multi-domain experimental environment, IoT application workloads, and baseline techniques.

Practical Experiment Environment

To evaluate the effectiveness of KD-AFRL in realistic heterogeneous multi-domain environments, we establish a distributed experimental infrastructure consisting of 10 inde-

Algorithm 6.3: Environment-Oriented Cross-Architecture Knowledge Distillation

Input: Domains with architectures $\{\theta_m\}_{m=1}^{M}$, architecture types $\{\text{Type}_m\}_{m=1}^{M}$, environmental compatibility threshold τ_{env}, number of teachers K, base temperature τ_{temp}, adjustment parameter $\beta_{distill}$, balancing parameter $\alpha_{distill}$

Output: Enhanced models $\{\theta_m\}_{m=1}^{M}$ through knowledge distillation

`/* Initialize Top-K Teacher-Student Pairing */`

Sort by model architecture complexity in descending order: $\{m_1, m_2, ..., m_M\}$

for *each student model* m_j, $j = 2$ *to* M **do**

  $\mathcal{T}_{m_j} = \{m_i | i < j, \text{EnvComp}(m_j, m_i) > \tau_{env}\}$

  $\mathcal{T}_{m_j}^{selected} = \text{TopK}(\mathcal{T}_{m_j}, K)$

end for

`/* Multi-Teacher Knowledge Distillation Training */`

for *each distillation round* $r = 1, 2, ..., R_{KD}$ **do**

  for *each student model* m_j **do in parallel**

    for *each teacher* $m_i \in \mathcal{T}_{m_j}^{selected}$, $i = 1$ *to* k **do**

      Set training episodes:

      $E_i = E_{base} \cdot (1 + \beta_{distill} \cdot \text{EnvComp}(m_j, m_i))$

      Compute teacher weight:

      $w_i = \frac{\text{EnvComp}(m_j, m_i)}{\sum_{l=1}^{k} \text{EnvComp}(m_j, m_l)}$

      for *training episode* $e = 1$ *to* E_i **do**

        Obtain teacher targets:

        $\pi_{teacher}^{soft}(a|s; \tau_{temp})$, $V_{teacher}(s)$

        Compute policy distillation loss:

        $\mathcal{L}_{KD}^{policy,i} = D_{KL}(\pi_{teacher}^{soft} \parallel \pi_{student}) \times \tau_{temp}^2$

        Compute value distillation loss:

        $\mathcal{L}_{KD}^{value,i} = ||V_{teacher} - V_{student}||^2$

        Compute single teacher loss:

        $\mathcal{L}_{distill}^{i} = \alpha_{distill} \cdot \mathcal{L}_{KD}^{policy,i} + (1 - \alpha_{distill}) \cdot \mathcal{L}_{KD}^{value,i}$

        Accumulate weighted loss:

        $\mathcal{L}_{total}^{distill} += w_i \cdot \mathcal{L}_{distill}^{i}$

      end for

    end for

    Update student model:

    $\theta_{student} \leftarrow \theta_{student} - \eta_{student} \nabla_{\theta_{student}} \mathcal{L}_{total}^{distill}$

  end forpar

end for

return Enhanced models $\{\theta_m\}_{m=1}^{M}$

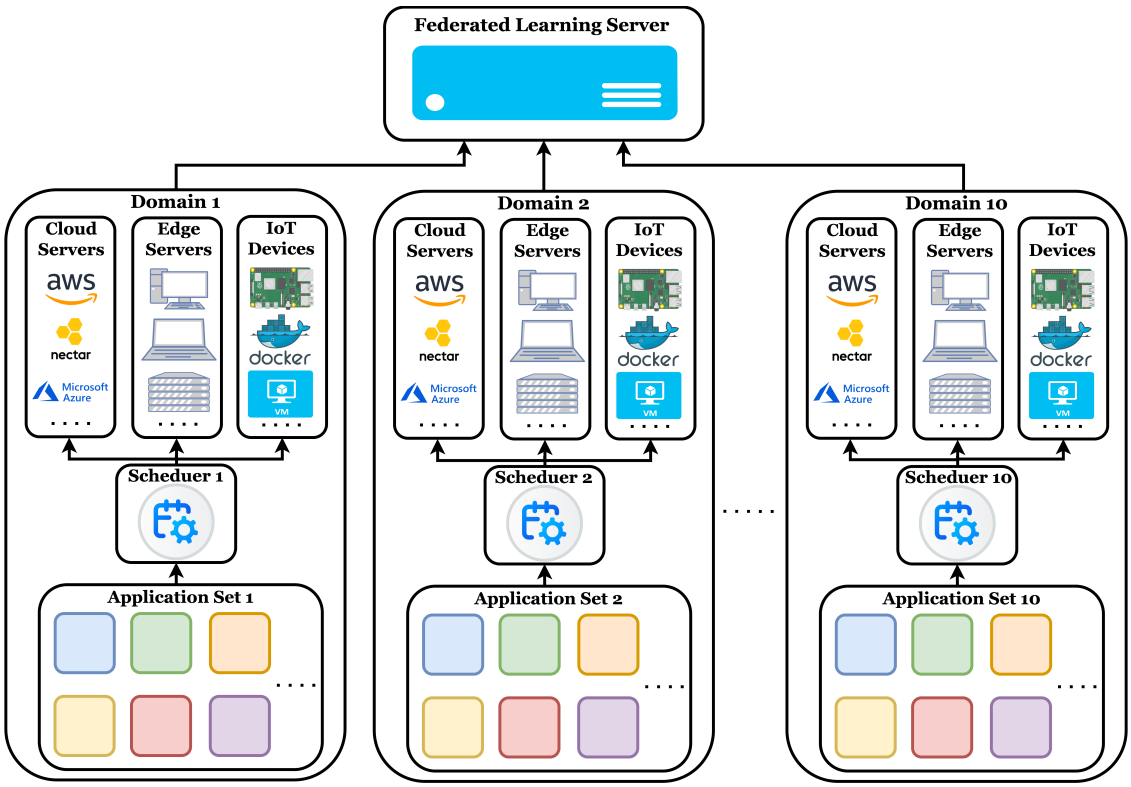


Figure 6.5: KD-AFRL experimental environment

pendent scheduling domains across different geographical locations and infrastructure providers, as shown in Fig. 6.5. Each domain operates as an autonomous entity with an independent scheduler responsible for resource management and workload scheduling, containing various combinations of cloud servers, edge servers, and IoT devices to form a heterogeneous Cloud-Edge-IoT computing environment.

At the Cloud layer, we deploy instances from different cloud service providers, including Nectar Cloud instances (AMD EPYC processors, configurations ranging from 2 cores @2.0GHz 8GB RAM to 32 cores @2.0GHz 128GB RAM), AWS Cloud instances (Intel Xeon processors, configurations ranging from 1 core @2.4GHz 1GB RAM to 16 cores @2.5GHz 64GB RAM), and Microsoft Azure Cloud instances (Intel Xeon processors, configurations ranging from 1 core @2.3GHz 1GB RAM to 24 cores @2.4GHz 96GB RAM). At the Edge layer, we configure various edge computing devices, including devices based on M1 Pro processors (8 cores, 16GB RAM), devices equipped with Intel Core i7 processors (8 cores @2.3GHz, 16GB RAM), Intel Core i9 processors (8 cores @2.5GHz, 32GB

RAM), and Intel Core i5 processors (6 cores @2.8GHz, 8GB RAM) with different configurations. At the IoT layer, we deploy Raspberry Pi devices (Pi OS, Broadcom BCM2837 quad-core @1.2GHz, 1GB RAM), virtual machines and Docker containers equipped with webcams and IP cameras.

The network connections exhibit realistic latency and bandwidth variations, reflecting real-world deployment scenarios. The latency between IoT devices and edge nodes ranges from 1-6ms with bandwidth of 10-25 MB/s. Network characteristics between IoT devices and cloud nodes vary by cloud service provider: latency with Nectar cloud servers ranges from 6-12ms with bandwidth of 14-20MB/s; latency with AWS cloud servers ranges from 15-25ms with bandwidth of 15-22MB/s; latency with Microsoft Azure cloud servers ranges from 7-15ms with bandwidth of 15-21MB/s. Communication latency between edge nodes and cloud nodes ranges from 6-25ms with bandwidth of 15-22MB/s. For energy monitoring, we use the `eco2AI` [246] to implement accurate real-time power measurement. In Equation 6.15, we set the weight parameter α_{cost} to 0.5 to balance response time and energy consumption optimization objectives.

IoT Application Workloads

To rigorously evaluate KD-AFRL under diverse computational conditions, we construct a heterogeneous IoT workload suite that reflects real-world deployments:

- *AudioAmplitudeMonitor*: real-time amplitude tracking implemented with `librosa` [293]; workload scaled by analysis-window length.
- *TextSentimentAnalysis*: sentiment inference on text using `TextBlob` [294] and `NLTK` [295]; workload scaled by text-block size.
- *SpeechRecognition*: speech-to-text inference combining `torchaudio` [296] and a lightweight Transformer decoder; workload scaled by audio-chunk length.
- *DataCompressionService*: lossless compression with `zlib` [297]/`gzip` [298]; workload scaled by file size and compression level.
- *ImageProcessor*: batch image filtering and resizing using `PIL` [299] and `OpenCV` [300]; workload scaled by batch size.

- *HealthTracker*: activity recognition via `TensorFlow Lite` [301, 302] pose estimation; workload scaled by sampling rate and model capacity.
- *FaceDetection*: face detection on streaming video with `OpenCV` [300] and `MediaPipe` [303]; workload scaled by frame resolution and detection frequency.
- *ColorTracking*: HSV-based colour tracking for AR overlays using `OpenCV` [300]; workload scaled by frame rate and tracking-window size.
- *FaceAndEyeDetection*: cascaded face-and-eye detection with landmark refinement using `OpenCV` [300] and `dlib` [304]; workload scaled by input resolution and cascade depth.
- *VideoOCR*: video text spotting (detection + recognition) with `EasyOCR` [305] and `OpenCV` [300]; workload scaled by clip length and model precision.

By systematically sweeping each application's dominant parameters (e.g., window length, batch size, model precision), we generate diverse IoT applications that span the full spectrum of resource footprintscovering I/O-bound, CPU-bound, memory-intensive, GPU-accelerated, and network-constrained characteristicsthereby exercising KD-AFRL across realistic deployment scenarios.

Environmental Heterogeneity Characterization

To characterize the Non-IID nature of our multi-domain deployment, we quantify the environmental diversity using the feature vectors defined in Section 6.4.2. We compute the Euclidean distance between all domain pairs based on their environmental features (including resource characteristics and workload patterns). After standardization, the analysis reveals an average inter-domain distance of 1.45 with a standard deviation of 0.62. The distance distribution ranges from 0.48 (most similar domain pair) to 3.12 (most dissimilar domain pair), demonstrating that our experimental environment naturally encompasses substantial heterogeneity across multiple dimensions including resource capacity, network characteristics, and workload patterns. This quantitative analysis confirms that our practical deployment spans the complete spectrum from high similarity

to significant differences, validating the presence of realistic Non-IID conditions in our evaluation.

Baseline Techniques

We implement KD-AFRL and all baseline techniques on the ReinFog platform [31] to ensure consistent experimental conditions. ReinFog is a modular platform for DRL-based resource management that supports both centralized and distributed techniques. To comprehensively evaluate the effectiveness of KD-AFRL, we adapt four representative state-of-the-art techniques to our multi-domain Cloud-Edge-IoT computing environment:

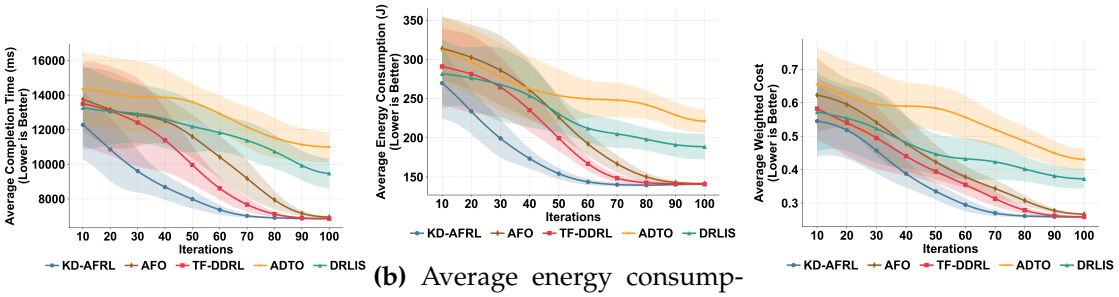
- **AFO**: The adapted version of the technique proposed in [124]. This technique employs federated DRL for task offloading. We adapted it by modifying its architecture and resource allocation models to operate in our heterogeneous Cloud-Edge-IoT computing environments. Additionally, we revised its reward function to align with our optimization objectives.

- **TF-DDRL**: The extended version of the technique proposed in [54]. This technique employs Transformer-enhanced distributed DRL based on IMPALA for IoT application scheduling in heterogeneous edge and cloud computing environments. We extended its architecture to support multi-domain scheduling scenarios.

- **ADTO**: The adapted version of the technique proposed in [288]. This technique employs A3C to solve the task offloading problem. We adapted its architecture to suit our multi-domain Cloud-Edge-IoT task scheduling problem. We also updated its reward function to align with our optimization objectives. Additionally, as A3C is commonly used in current literature ([27, 239, 257, 289]), ADTO provides a representative benchmark for evaluating A3C-based solutions.

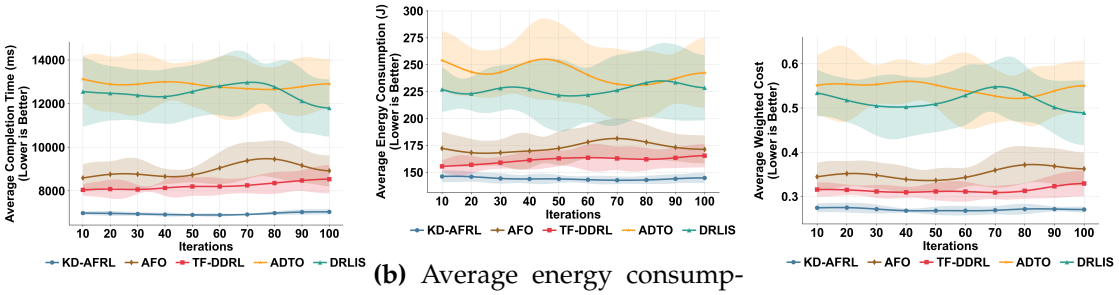
- **DRLIS**: The extended version of the technique proposed in [26]. This technique employs a centralized DRL agent based on PPO for IoT application scheduling. We extended its reward function to align with our optimization objectives. As

PPO is a policy gradient (PG) algorithm with superior training stability and sample efficiency [102], and given that PG algorithms are commonly used in current literature ([233, 234, 284, 285]), DRLIS provides a representative benchmark for evaluating PG-based solutions.



(a) Average completion time **(b)** Average energy consumption **(c)** Average weighted cost

Figure 6.6: Convergence performance analysis during training phase



(a) Average completion time **(b)** Average energy consumption **(c)** Average weighted cost

Figure 6.7: Convergence performance analysis during evaluation phase

6.5.2 Hyperparameter Configuration

We conducted systematic hyperparameter tuning using grid search with cross-validation for each domain. Learning rates and discount factors are optimized per scheduler. Table 6.2 summarizes the key hyperparameter settings. Baseline methods use identical tuning procedures.

Table 6.2: The key hyperparameters setting for KD-AFRL

Parameter	Value	Parameter	Value
Learning Rate η	[0.0001, 0.01]	Privacy Budget ϵ	1.0
Discount Factor γ	[0.8, 0.99]	Drift Threshold τ_{drift}	0.1
Architecture Types K_{arch}	8	Environmental Threshold τ_{env}	0.6
Network Layers	[2, 10]	Temperature τ_{temp}	3.0
Neurons per Layer	[16, 512]	Top-K Teachers K	3
Cost Weight α_{cost}	0.5	Balancing Parameter $\alpha_{distill}$	0.7

6.5.3 Experimental Results and Analysis

The following subsections systematically analyze convergence performance, federated learning and knowledge distillation contributions, adaptive architecture effectiveness, and framework scalability. All experiments reported in this section are conducted with 5 independent runs using different random seeds to ensure statistical rigor. The reported results represent mean values, with shaded areas in all figures indicating ± 1 standard deviation to demonstrate the stability of our approach.

Convergence Performance Analysis

To evaluate the learning efficiency and convergence characteristics of KD-AFRL, we design the experiment comprising training and evaluation phases. The training phase employs a dedicated set of training applications for policy learning and parameter updates, while the evaluation phase freezes the learning process and evaluates the generalization performance of trained policies using a completely different set of evaluation applications. This design ensures genuine assessment of generalization capabilities on unseen applications. To ensure fair comparison, all techniques are evaluated under identical conditions. The reported results represent application-level performance metrics (completion time, energy consumption, and weighted cost as defined in Section 6.3.2) averaged across all 10 scheduling domains.

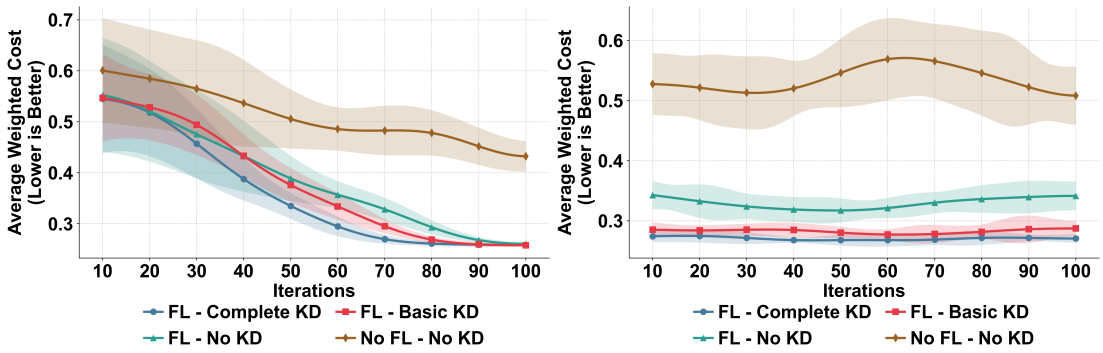
Experimental results demonstrate KD-AFRL's superiority in both training efficiency and evaluation effectiveness. During the training phase (Fig. 6.6), KD-AFRL achieves optimal convergence within 70-80 iterations, approximately 21% faster than TF-DDRL and AFO (90-100 iterations), while DRLIS and ADTO fail to converge within 100 it-

erations. In the evaluation phase (Fig. 6.7), KD-AFRL not only maintains stable performance on unseen applications but also outperforms the best baseline (TF-DDRL) by 15.7%, 10.8%, and 13.9% in completion time, energy consumption, and weighted cost respectively, while all baseline techniques exhibit significant performance fluctuations and degradation. This superior performance stems from the synergistic effects of our three core mechanisms: adaptive architectures that prevent computational mismatches, environment clustering that ensures compatible domains share knowledge, and cross-architecture distillation that enables resource-constrained devices to learn from more capable ones.

Federated Learning and Knowledge Distillation Analysis

Federated learning and knowledge distillation are critical mechanisms for improving scheduling performance in heterogeneous IoT environments, directly affecting application completion time and energy consumption. To evaluate individual contributions of federated learning and knowledge distillation, we compare four learning strategies: No FL - No KD (local-training only), FL - No KD (federated only), FL - Basic KD (federated with basic distillation), and FL - Complete KD (federated with environment-oriented distillation). As shown in Fig. 6.8, FL - Complete KD achieves fastest convergence (80 iterations), outperforming FL - Basic KD (90 iterations), FL - No KD (100 iterations), and No FL - No KD (non-convergent). During evaluation, transitioning from local-only to federated learning reduces cost from 0.5 to 0.33 (34% improvement), basic distillation further reduces it to 0.28, while complete distillation achieves optimal performance at 0.27. These results demonstrate that combining federated learning with environment-oriented knowledge distillation yields a 46% overall improvement (from 0.5 to 0.27), where federated learning enables collaboration among compatible domains while knowledge distillation extends this benefit to heterogeneous models.

To further demonstrate the effectiveness of cross-architecture knowledge distillation in bridging the performance gap between heterogeneous models, we analyze the performance differences between small models (2-4 layers with up to 32 neurons per layer) and large models (8-10 layers with up to 512 neurons per layer) under the FL - Com-



(a) Training phase **(b)** Evaluation phase

Figure 6.8: Performance comparison of different learning strategies during training and evaluation phases

plete KD strategy. As illustrated in Fig. 6.9, despite significant architectural disparities, the knowledge distillation mechanism successfully enables small models on resource-constrained devices to achieve an average weighted cost of 0.283, closely approaching the 0.261 achieved by large models on high-end devices. This represents 92.2% relative efficiency, demonstrating that small models can effectively learn from and approximate the decision-making capabilities of their more complex counterparts.

Sensitivity Analysis of Top-K Teacher Selection

To systematically evaluate the impact of the Top-K teacher selection parameter on knowledge distillation effectiveness, we conduct sensitivity analysis by varying K from 1 to 5. As shown in Fig. 6.10, K=3 achieves the fastest convergence during training phase, while other configurations exhibit slower convergence to varying degrees. During evaluation, K=3 still maintains the lowest average weighted cost and exhibits the most stable performance with minimal fluctuation, while other configurations show higher average costs and greater performance variability across iterations. These results demonstrate that K=3 provides the optimal balance between knowledge diversity and guidance consistency, achieving superior convergence speed and generalization capability while avoiding negative interference from excessive heterogeneous teaching signals.

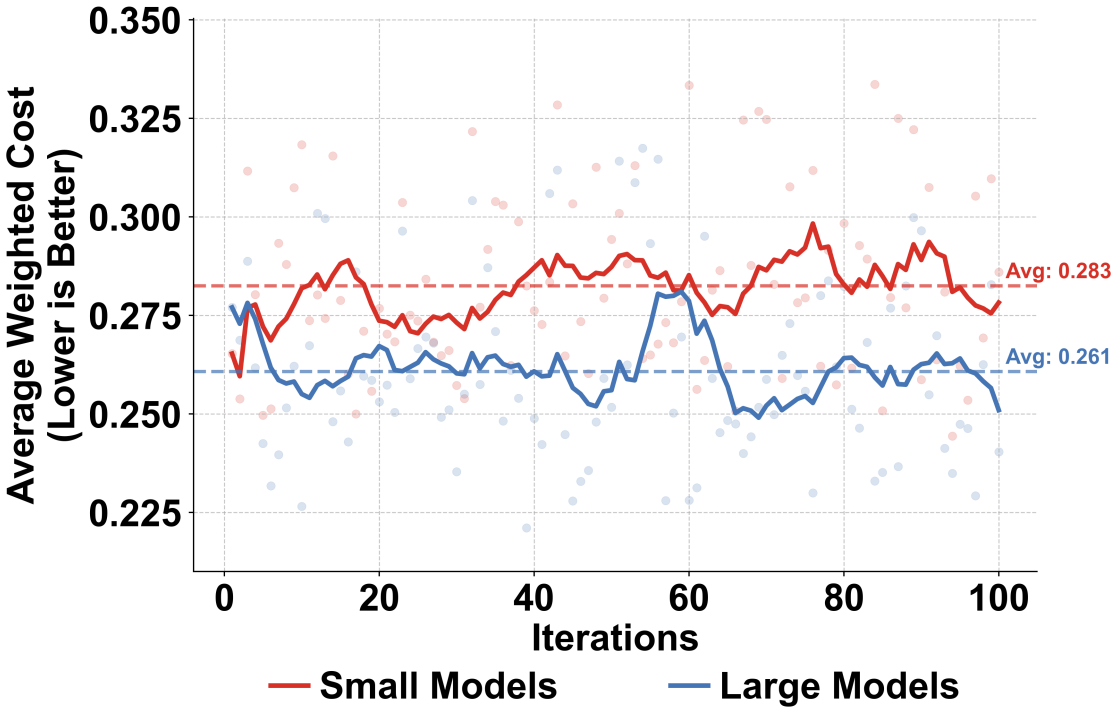
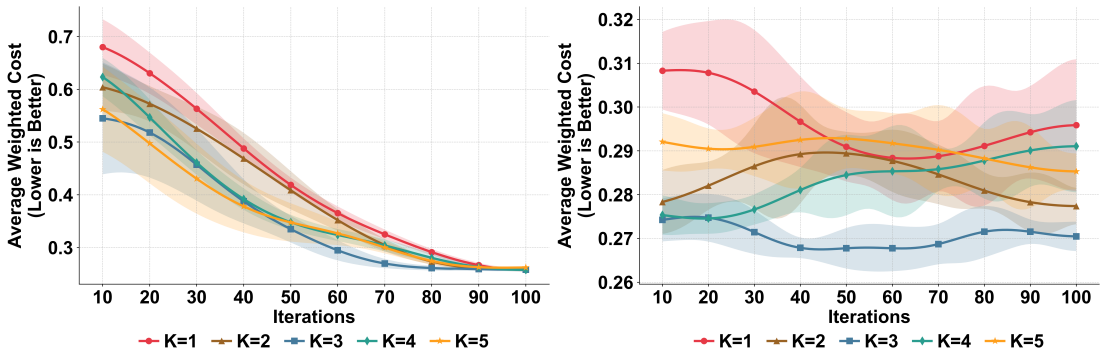


Figure 6.9: Cross-architecture knowledge distillation performance between small and large models

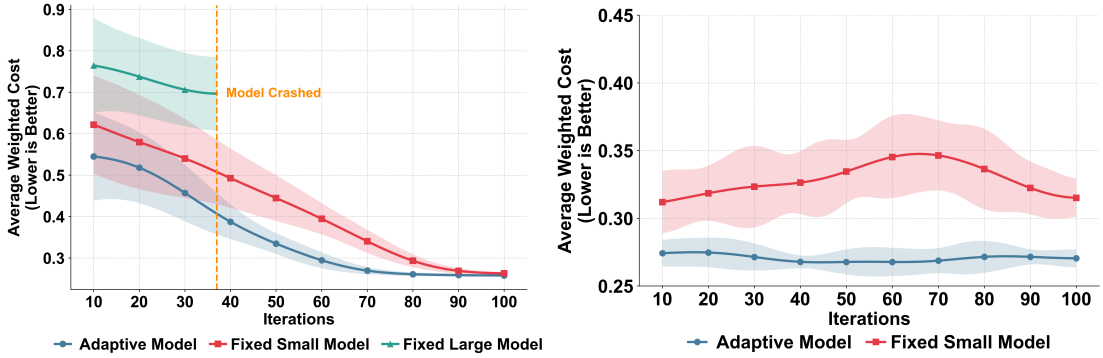
Adaptive Architecture Analysis

To evaluate the effectiveness of the adaptive architecture generation mechanism, we conduct experiments in a heterogeneous distributed environment with multiple device types. Each scheduler is required to handle various types of applications with different computational demands. We deploy schedulers across three categories of heterogeneous devices, with multiple instances configured for each category: low-end devices (1-2 cores and 1-2 GB RAM), medium devices (4-8 cores and 8-16GB RAM), and high-end devices (8-32 cores and 32-128GB RAM). For comparison, we evaluate three approaches: (1) Fixed Small Model - a lightweight architecture with 2 hidden layers of 32 neurons deployed uniformly across all devices, (2) Fixed Large Model - a complex architecture with 8 hidden layers of 512 neurons deployed uniformly across all devices, and (3) Adaptive Model - our proposed approach that automatically adjusts architecture complexity based on device capabilities.



(a) Training phase **(b)** Evaluation phase

Figure 6.10: Performance comparison across different Top-K teacher selection values

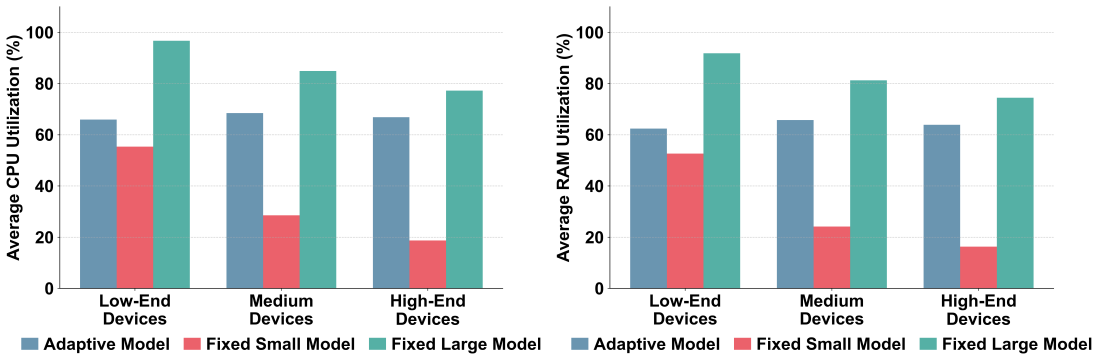


(a) Training phase **(b)** Evaluation phase

Figure 6.11: Performance comparison of adaptive, fixed small, and fixed large models during training and evaluation phases

The training and evaluation phase results demonstrate the superior stability and convergence characteristics of the adaptive model. As shown in Fig. 6.11, during training, the adaptive model achieves rapid convergence within approximately 80 iterations, while the fixed small model requires around 100 iterations, and the fixed large model crashes at around iteration 37 due to resource constraints. During the evaluation phase, the adaptive model maintains stable optimal performance levels at around 0.27. In contrast, the fixed small model exhibits significant performance oscillations, with its weighted cost fluctuating between around 0.31 and 0.35.

We also examine the resource utilization patterns to further evaluate the effectiveness of the adaptive model. Fig. 6.12 shows the adaptive model achieves balanced resource



(a) CPU resource utilization **(b)** RAM resource utilization

Figure 6.12: Resource utilization comparison across heterogeneous devices

utilization across all device types (CPU: 66-68%, RAM: 62-66%), while the fixed small model severely underutilizes resources on medium and high-end devices (CPU: 19-29%, RAM: 16-24%), and the fixed large model causes excessive consumption on low-end devices (CPU: 97%, RAM: 92%), leading to system failure. These results demonstrate that our adaptive architecture generation mechanism effectively balances computational demands and available resources in heterogeneous environments.

Framework Scalability Analysis

To evaluate the scalability of KD-AFRL, we design a dynamic expansion experiment that incrementally increases the number of domains from 10 to 30. As shown in Fig. 6.13, KD-AFRL demonstrates superior scalability compared to all baseline methods across different domain sizes. When the number of domains increases from 10 to 30, KD-AFRL maintains the lowest average weighted cost, increasing from approximately 0.25 to 0.27, representing only an 8% performance degradation. In contrast, the baseline techniques show significantly steeper performance decline: AFO increases from 0.27 to 0.38 (41% degradation), TF-DDRL from 0.27 to 0.36 (33% degradation), ADTO from 0.45 to 0.55 (22% degradation), and DRLIS from 0.44 to 0.59 (34% degradation). This demonstrates that KD-AFRL exhibits 3-5 times better performance retention compared to existing methods as the system scales.

The superior scalability of KD-AFRL stems from the complementary effects of feder-

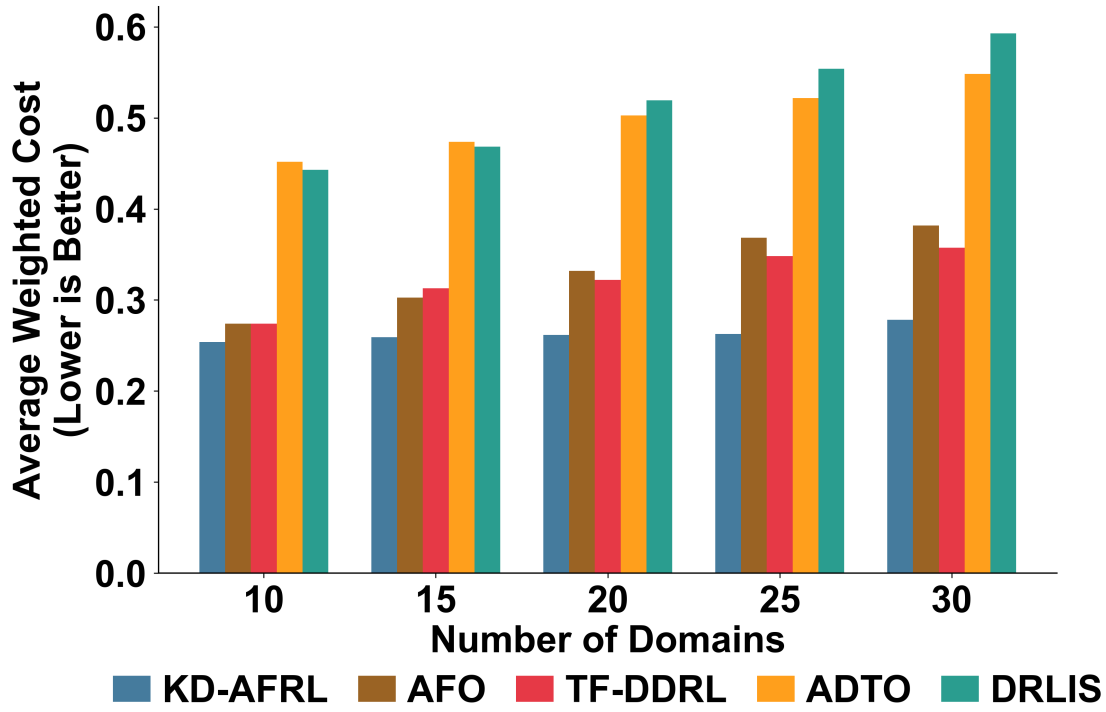


Figure 6.13: Scalability performance comparison across varying number of domains

ated learning and environment-oriented knowledge distillation, enabling new domains to quickly learn from existing experienced domains without starting from scratch. This scalability advantage is crucial for practical deployments where IoT systems continuously expand with new domains joining.

6.6 Summary

In this chapter, we propose KD-AFRL, a Knowledge Distillation-empowered Adaptive Federated Reinforcement Learning framework for multi-domain IoT application scheduling. The framework addresses the fundamental challenge of enabling effective collaborative learning across heterogeneous domains through three synergistic mechanisms: resource-aware hybrid architecture generation, privacy-preserving environment-clustered federated learning, and environment-oriented cross-architecture knowledge distillation. A central insight from our work is that these mechanisms must be co-

designed rather than treated independently. Resource-aware architecture generation addresses computational heterogeneity by adapting model complexity, but simultaneously creates architectural barriers. Cross-architecture knowledge distillation breaks through these barriers via soft-target transfer. Environment clustering guides both federated aggregation and knowledge distillation based on domain compatibility, preventing negative transfer from incompatible domains. Through practical deployment evaluation across real Cloud-Edge-IoT infrastructure, we demonstrate that federated reinforcement learning can effectively operate in truly heterogeneous multi-domain environments where device capabilities, data distributions, and operational contexts vary significantly. This chapter establishes the feasibility of multi-domain collaborative learning for IoT scheduling and provides a foundation for extending federated learning paradigms from traditional homogeneous settings to realistic heterogeneous deployments.

Chapter 7

DeFRiS: Silo-Cooperative IoT Applications Scheduling via Decentralized Federated Reinforcement Learning

Next-generation IoT applications increasingly span autonomous administrative entities, necessitating silo-cooperative scheduling that leverages diverse computational resources while preserving data privacy. However, efficient cooperation faces significant challenges from infrastructure heterogeneity, Non-IID workload shifts, and adversarial environments. To address these challenges, we propose DeFRiS, a Decentralized Federated Reinforcement Learning framework for robust silo-cooperative IoT scheduling. DeFRiS integrates three synergistic innovations: (i) an action-space-agnostic policy using candidate resource scoring to enable knowledge transfer across heterogeneous silos; (ii) a silo-optimized local learning mechanism combining Generalized Advantage Estimation with clipped policy updates to address sparse delayed rewards; and (iii) a Dual-Track Non-IID robust decentralized aggregation protocol leveraging gradient fingerprints for similarity-aware knowledge transfer and anomaly detection, coupled with gradient tracking for optimization momentum. Experiments on a distributed testbed with 20 heterogeneous silos and realistic IoT workloads demonstrate significant improvements over state-of-the-art baselines: 6.4% reduction in average response time, 7.2% in energy consumption, 10.4% lower tail latency risk ($CVaR_{0.95}$), and near-zero deadline violations. Furthermore, DeFRiS achieves over 3× better performance retention during scaling and over 8× better stability under adversarial conditions compared to the best-performing baseline.

This chapter is derived from:

- **Zhiyu Wang**, Mohammad Goudarzi, Mingming Gong, and Rajkumar Buyya, "DeFRiS: Silo-Cooperative IoT Applications Scheduling via Decentralized Federated Reinforcement Learning", *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 2026 [Under Review].

7.1 Introduction

The Internet of Things (IoT) has become ubiquitous in smart manufacturing, healthcare, and urban infrastructure [306]. Traditional deployments strictly adhere to single-authority boundaries, where a solitary entity retains exclusive control over resource provisioning and scheduling [52]. However, next-generation applications are breaking these boundaries, demanding cross-organizational cooperation [307]. In smart cities, distinct departments manage traffic, energy, and emergency services utilizing separate infrastructures that must nevertheless coordinate during critical events. Similarly, industrial supply chains link manufacturers, logistics providers, and retailers, necessitating computational synergy among independent systems without exposing proprietary data. These scenarios share a fundamental requirement: enabling resource cooperation across autonomous administrative boundaries while maintaining strict data privacy. We term each such independent administrative unit a computing silo [308].

Realizing multi-silo cooperation, however, presents distinct technical challenges. First, significant infrastructure heterogeneity exists across silos tasked with scheduling applications modeled as Directed Acyclic Graphs (DAGs) [309]. Mapping these complex task dependencies onto diverse hardware configurations creates severe compatibility issues. A silo equipped with sparse edge resources and abundant cloud capacity presents a fundamentally different state-action space compared to an edge-dense counterpart. This dimensional and semantic mismatch makes transferring scheduling policies between silos non-trivial. Second, workload variations exacerbate this complexity [310]. The divergence between real-time stream processing and batch analytics creates a Non-Independent and Identically Distributed (Non-IID) shift in underlying data distributions [34, 311]. Indiscriminate aggregation of knowledge across such diverse environments can induce negative transfer, where incompatible patterns degrade rather than enhance performance. Finally, the scheduling environment involves inherent uncertainties and adversarial risks [312]. The optimization process is plagued by sparse, delayed rewards, where feedback spans hundreds of decision steps [313], and is further threatened by potential hardware failures or malicious model poisoning [314].

Deep Reinforcement Learning (DRL) provides a powerful solution for addressing

these dynamic scheduling complexities [155], yet its application in multi-silo environments faces a fundamental dilemma between autonomy and cooperation. Existing approaches generally adopt either independent learning or cooperative learning paradigms. Independent learning preserves data sovereignty by restricting training to local resources, but this isolation precludes the sharing of optimization experience, resulting in severe sample inefficiency and suboptimal convergence [201]. Conversely, cooperative learning seeks to synthesize collective intelligence but faces distinct structural and algorithmic hurdles [315]. Conventional cooperative frameworks typically rely on centralized coordination mechanisms that introduce communication bottlenecks and single points of failure, effectively compromising scalability and robustness [316]. More critically, these approaches predominantly operate under idealized assumptions of model homogeneity and truthful participation [14]. Consequently, they lack the architectural flexibility to support heterogeneous state-action spaces, the mathematical robustness to mitigate Non-IID distribution shifts, and the security mechanisms to withstand adversarial environments (e.g., malicious nodes, hardware failures, network disruptions) [14, 119, 316].

To bridge this gap, we propose a Decentralized Federated Reinforcement Learning framework for Silo-Cooperative IoT Application Scheduling (DeFRiS). DeFRiS fundamentally dismantles the centralized bottleneck by adopting a fully decentralized peer-to-peer architecture, where silos exchange knowledge via dynamically selected neighbors without a central coordinator. The framework integrates three synergistic innovations to resolve the aforementioned technical hurdles. First, to overcome infrastructure heterogeneity, we propose a novel action-space-agnostic policy. By employing candidate resource scoring that decouples policy parameters from specific physical dimensions, it enables seamless knowledge transfer across silos with varying configurations. Second, to handle sparse delayed rewards, we design a silo-optimized local learning mechanism. This combines an Actor-Critic architecture with Generalized Advantage Estimation (GAE) to transform sparse terminal signals into dense credits, utilizing clipped policy updates to guarantee training stability. Finally, to ensure robustness against Non-IID shifts and adversarial threats, we introduce a Dual-Track Non-IID robust decentralized aggregation protocol. This employs a dual-track strategy: using gradient fingerprints to detect anomalies and selectively transfer Actor knowledge based on similarity, while

leveraging gradient tracking for Critic networks to harmonize global optimization momentum with local value estimation.

The main contributions of this chapter are:

- We formulate silo-cooperative IoT scheduling as a risk-sensitive joint optimization problem. By incorporating Conditional Value at Risk (CVaR) to manage tail risks alongside energy objectives, we strictly enforce hard constraints while optimizing soft goals, formalizing the system as a distributed Markov Decision Process (MDP).

- We propose a novel action-space-agnostic policy that decouples decision-making from physical resource dimensions. Utilizing a candidate resource scoring mechanism instead of fixed-dimensional output layers, this approach enables effective policy knowledge sharing across silos with fundamentally heterogeneous resource configurations.

- We design a silo-optimized local learning mechanism to address the sparse reward challenge in long-horizon scheduling. By integrating GAE for precise credit assignment and clipped policy updates, we ensure training stability and high-quality local parameter generation.

- We introduce a dual-track Non-IID robust aggregation protocol that enables cooperation and defense. For Actor networks, we employ gradient fingerprint-based similarity measurement to detect anomalies and selectively transfer knowledge; for Critic networks, we utilize gradient tracking to share global optimization momentum while preserving accurate local value estimation.

- We implement and evaluate DeFRiS on a distributed testbed comprising heterogeneous silos running realistic IoT applications. Extensive experiments demonstrate that DeFRiS significantly outperforms state-of-the-art baselines in terms of response time, energy efficiency, QoS guarantee, scalability, and adversarial robustness.

The rest of the chapter is organized as follows. Section 7.2 reviews related work and identifies research gaps through systematic comparison. Section 7.3 establishes the

system model, optimization objectives, and MDP formalization. Section 7.4 details the DeFRiS framework. Section 7.5 presents comprehensive experimental evaluation in real heterogeneous environments. Section 7.6 concludes the chapter.

7.2 Related Work

This section systematically reviews existing IoT application scheduling approaches, categorizing them into independent and cooperative learning paradigms, and identifies four critical research gaps through a comprehensive qualitative comparison that motivate the design of DeFRiS.

7.2.1 Independent Learning-based Approaches

Chen et al. [235] proposed a dependency-aware task offloading approach with Deep Q-Network (DQN) for real-time task offloading in cloud-edge environments. They modeled mobile applications as DAGs and customized DQN to train the decision-making model that considers task parallelism without presetting task priority. The objective is to minimize the response time of mobile applications. Tang et al. [283] proposed a redundancy-aware adaptive search offloading approach based on DQN for task offloading in mobile edge computing networks. They designed a fine-grained task recombination scheme to eliminate redundant subtask data and codes, and organized devices into a spatial index MP-tree for efficient search. The objective is to minimize offloading delay and energy consumption. Deng et al. [317] proposed a DRL approach integrating Transformer models for DAG task scheduling in the Internet of Vehicles. They employ Transformers to learn vehicle trajectory features, combined with Proximal Policy Optimization (PPO) for decision-making. The objective is to minimize task completion time. Wang et al. [26] proposed a DRL-based IoT application scheduling algorithm using PPO for DAG-based IoT applications in fog computing environments. They designed a weighted cost model considering task dependencies. The objective is to optimize load balancing across servers while minimizing application response time. Zhang et al. [318] proposed a Dueling Double Deep Q-Network enabled SEIDEL (D3QN-SEIDEL) algo-

rithm for diversified-task co-offloading in Industrial IoT. They constructed a synthetic-expense function considering task correlations, task preferences, and industrial channel characteristics, and categorized production line tasks into three types. The objective is to minimize wireless transmission costs and task execution delay.

7.2.2 Cooperative Learning-based Approaches

Li et al. [319] proposed a Multi-Agent Proximal Policy Optimization (MAPPO) algorithm for multi-task scheduling in 6G-enabled intelligent autonomous transport systems. They designed a vehicle-infrastructure network where vehicles can assign multiple computation tasks to edge servers and other idle vehicles, considering vehicle mobility and current workload. The objective is to jointly minimize service request completion time and energy consumption. Chen et al. [320] proposed a DRL-based algorithm for vehicular edge computing networks with edge-to-edge collaboration. They employ MAPPO for sub-channel allocation and power control, and PPO for load balancing. The objective is to minimize total task delay. Liu et al. [321] proposed the VCPN framework integrating IoTs, vehicles, and edge servers as adaptive computing nodes. They employ the Multi-Agent Deep Deterministic Policy Gradient (MADDPG) algorithm with centralized training and decentralized execution paradigm. The objective is to minimize end-to-end task completion time. Wu et al. [287] proposed PG-DDQN, combining Prioritized Experience Replay and Distributed DQN, for proactive caching in cloud-edge computing. They use the multi-agent architecture for centralized training and distributed inference. The objective is to maximize edge hit ratio while minimizing content access latency and traffic cost. Zhang et al. [27] proposed LsiA3CS, an Asynchronous Advantage Actor-Critic (A3C) based framework for cross-edge cloud collaborative task scheduling in large-scale Industrial IoT. They designed a Markov game model with heuristic policy annealing and action masking to achieve distributed, asynchronous task scheduling across heterogeneous computational resources. The objective is to minimize task completion time and energy consumption. Wang et al. [54] proposed TF-DDRL, a Transformer-enhanced Distributed DRL technique based on IMPALA for IoT application scheduling. They integrate Transformers for capturing task dependen-

cies and Prioritized Experience Replay for reducing exploration costs. The objective is to minimize response time, energy consumption, and monetary cost. Raju et al. [322] proposed DMITS, using federated DRL with Soft Actor-Critic (SAC) for vehicular fog computing. They model social relationships among vehicles based on bridge centrality and Jaccard similarity. The objective is to maximize task success rate and minimize energy consumption. Chen et al. [323] proposed MCM-FDRL, using Federated DRL for vehicular task offloading. Each vehicle acts as an independent agent making offloading decisions using DQN, with model parameters aggregated through federated learning for global optimization. The objective is to minimize task response time while achieving load balancing across edge servers.

Table 7.1: Qualitative Comparison of IoT Application Scheduling Approaches

Work	System & Application Properties			Problem Formulation				Approach Properties								Evaluation			
	Task Number	Task Dependency	Multi-Silo	Time	Energy	Multi-Objective	QoS	Learning Paradigm	Algorithm	Coordination Architecture	Action Space	Sparse Reward Handling	Non-IID Handling	Aggregation	Robustness	Edge	Cloud	IoT Apps	Hetero-geneity
[235]	Multiple	✓	×	✓	×	×	×	Independent	DQN	N/A	Fixed	×	N/A	N/A	×	○	○	○	✓
[283]	Multiple	✓	×	✓	✓	✓	✓		DQN	N/A	Fixed	×	N/A	N/A	×	○	○	◐	×
[317]	Multiple	✓	×	✓	×	×	×		PPO	N/A	Fixed	✓	N/A	N/A	×	○	○	◐	✓
[26]	Multiple	✓	×	✓	×	✓	×		PPO	N/A	Fixed	✓	N/A	N/A	×	●	●	●	✓
[318]	Multiple	✓	×	✓	×	✓	✓		D3QN	N/A	Fixed	×	N/A	N/A	×	○	○	●	✓
[319]	Single	×	×	✓	✓	✓	✓	Cooperative	MAPPO	Centralized	Fixed	✓	N/A	N/A	×	○	N/A	○	×
[320]	Single	×	×	✓	×	×	✓		MAPPO	Centralized	Fixed	✓	N/A	N/A	×	○	N/A	◐	×
[321]	Single	×	×	✓	×	×	✓		MADDPG	Centralized	Fixed	×	N/A	N/A	×	○	○	○	✓
[287]	Single	×	×	✓	×	✓	×		Distributed DQN	Centralized	Fixed	✓	N/A	N/A	×	○	○	◐	×
[27]	Single	×	×	✓	×	✓	✓		A3C	Centralized	Fixed	✓	N/A	N/A	×	○	○	◐	✓
[54]	Multiple	✓	×	✓	✓	✓	×		IMPALA	Centralized	Fixed	✓	N/A	N/A	×	●	●	●	✓
[322]	Multiple	✓	×	✓	✓	✓	✓		Federated SAC	Centralized	Fixed	✓	N/A	Weighted Aggregation	×	◐	○	◐	✓
[323]	Single	×	×	✓	×	×	×		Federated DQN	Centralized	Fixed	×	N/A	FedAvg	×	○	N/A	◐	×
DeFRiS	Multiple	✓	✓	✓	✓	✓	✓		Federated Actor-Critic	Decentralized	Agnostic	✓	Gradient Fingerprint	Gradient Fingerprint + Gradient Tracking	Gradient Fingerprint + Median Absolute Deviation	●	●	●	✓

Note: For Edge/Cloud columns: ○ indicates simulation, ● indicates real deployment. For IoT Apps column: ○ indicates fully simulated applications, ◐ indicates applications simulated using real-world datasets, and ● indicates real IoT application deployment.

7.2.3 A Qualitative Comparison

To systematically evaluate existing IoT application scheduling approaches and identify research gaps, we construct a qualitative comparison encompassing multiple indicators across four major categories, as presented in Table 7.1.

Comparative Analysis Dimensions

In the system and application properties dimension, we examine task number (single-task or multi-task applications), inter-task dependencies (independent or with precedence-successor constraints), and multi-silo architecture support. The problem formulation di-

mension covers optimization objective selection, including time, energy consumption, multi-objective trade-offs, and QoS constraints (e.g., deadlines). The approach properties dimension characterizes technical differences from multiple perspectives: learning paradigm (independent or cooperative learning), specific algorithm choice, coordination architecture (centralized or decentralized), action space type (fixed or adaptive), sparse reward handling (whether approaches employ mechanisms to address delayed feedback), while particularly focusing on how federated learning approaches handle Non-IID data, model aggregation strategies, and robustness in adversarial environments. The evaluation dimension assesses deployment realism across edge, cloud, and IoT applications, while heterogeneity indicates whether the evaluation considers environmental heterogeneity (e.g., diverse device capabilities, network conditions).

Research Gap Identification

Based on the comprehensive comparison presented in Table 7.1, we identify four critical research gaps that are intrinsically interconnected and constitute systematic challenges in IoT application scheduling.

Gap 1: Lack of Multi-Silo Architectural Support. Among all 14 studied works, only DeFRiS considers the multi-silo scenario, indicating that existing works universally assume scheduling occurs within a single edge cloud environment, which contradicts real-world cross-organizational cooperation scenarios. More critically, among the 9 cooperative learning approaches, all 8 except DeFRiS adopt centralized coordination architectures, which suffer from data privacy concerns, a single point of failure, and limited scalability in geographically distributed scenarios.

Gap 2: Dual Absence of Non-IID Handling and Robustness Mechanisms. The Non-IID Handling and Robustness columns reveal that 13 works neither address Non-IID data nor consider robustness mechanisms. This dual absence is particularly problematic as multi-silo scenarios inevitably introduce data distribution divergence across different geographical locations, while distributed learning environments face threats from malicious or faulty nodes uploading poisoned gradients. Even federated learning works (e.g., weighted aggregation in [322], FedAvg in [323]) merely employ vanilla

aggregation schemes without specialized mechanisms for these critical challenges.

Gap 3: Limited Action Space Adaptability. The Action Space column shows that 13 works employ fixed action spaces, with only DeFRiS supporting action space agnosticism. Fixed action spaces assume that scheduling decision options remain static at runtime, failing to exploit task divisibility for fine-grained parallel optimization, adapt to dynamic resource availability changes, or adjust allocation granularity based on heterogeneous device capabilities.

Gap 4: Insufficient Real-World Validation. Statistics from the Evaluation dimension reveal pervasive validation insufficiency: except DeFRiS, only 2 works ([26, 54]) conducted real edge/cloud deployment, and only 3 works ([26, 54, 318]) employed real IoT applications. This validation gap leaves the performance transferability from simulation to real deployment unverified and masks practical challenges (e.g., network latency variance, straggler effects, device failures) in heterogeneous environments.

In summary, these four research gaps are deeply interdependent and form a cascading chain of challenges. Multi-silo scenarios (Gap 1) inherently introduce data heterogeneity and security vulnerabilities across distributed sites, necessitating sophisticated Non-IID handling and robustness mechanisms (Gap 2). These distributed heterogeneous environments exhibit dynamic resource availability and varying device capabilities, demanding adaptive action space strategies (Gap 3). Ultimately, addressing these challenges requires rigorous validation through comprehensive real-world deployment (Gap 4). While existing works have advanced individual aspects, none provide an integrated framework to tackle these interconnected challenges holistically, which constitutes the fundamental motivation for DeFRiS.

7.3 Problem Formulation

This section presents the mathematical formulation of the silo-cooperative IoT scheduling problem. We establish the system model, define the objectives of response time and energy consumption, and formulate the joint optimization problem subject to operational constraints. Finally, we transform the problem into a distributed Markov Decision Process (MDP) to enable federated DRL-based solutions.

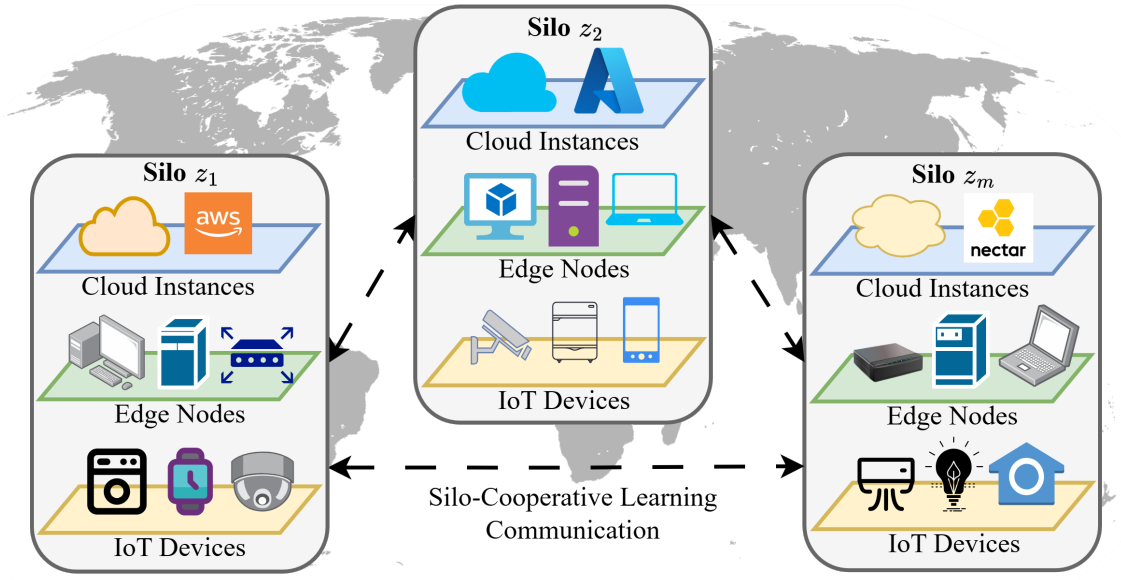


Figure 7.1: The distributed IoT system model illustrating autonomous silos with heterogeneous resources connected for cooperative learning.

7.3.1 Silo-Cooperative System Model

As illustrated in Fig. 7.1, we consider a large-scale distributed IoT system comprising multiple dispersed autonomous computing silos $\mathcal{Z} = \{z_1, z_2, \ldots, z_M\}$, each deploying heterogeneous computational resources to support diverse IoT applications. The silos can communicate with each other for cooperative learning purposes while maintaining operational independence. Each autonomous silo $z_i \in \mathcal{Z}$ represents an independent management entity containing a heterogeneous set of computing resources $\mathcal{N}_i$ spanning IoT devices, edge nodes, and cloud instances.

Each computing entity $n \in \mathcal{N}_i$ is characterized by its computing capability spectrum $\boldsymbol{\rho}_n = (f_n, m_n, d_n, e_n)$, where:

- f_n represents the processor frequency (GHz),
- m_n represents the available memory capacity (GB),
- d_n represents the storage space size (GB),
- e_n represents the energy capacity constraint (J), primarily for battery-powered IoT devices; for edge and cloud resources with a stable power supply, $e_n = \infty$.

Each silo z_i receives its own set of IoT applications denoted as $\mathcal{T}_i \subseteq \mathcal{T}$, where $\mathcal{T}$ represents the global set of all applications in the system. Considering the complexity and componentized characteristics of modern IoT applications, we model each IoT application $\tau \in \mathcal{T}_i$ as a weighted DAG $\mathcal{G}_\tau = (\mathcal{V}_\tau, \mathcal{E}_\tau, \mathcal{W}_\tau)$, where the vertex set $\mathcal{V}_\tau = \{v_1, v_2, \ldots, v_{|\tau|}\}$ represents atomic tasks, the edge set $\mathcal{E}_\tau \subseteq \mathcal{V}_\tau \times \mathcal{V}_\tau$ encodes data dependency relationships among tasks, and the weight function $\mathcal{W}_\tau : \mathcal{E}_\tau \to \mathbb{R}_+$ quantifies the data transfer volume between tasks.

We assume that IoT applications provide predefined information upon submission, which is reasonable in practice since IoT applications typically have well-defined computational patterns and explicit QoS requirements. Based on this assumption, each application τ has an overall deadline constraint $T_\tau^{deadline} \in \mathbb{R}_+$ (in ms), and each task $v_j \in \mathcal{V}_\tau$ is described by the following attribute tuple:

$$v_j = \langle f_j^{req}, m_j^{req}, d_j^{req} \rangle, \tag{7.1}$$

where:

- f_j^{req} is the required CPU cycles (Million cycles),
- m_j^{req} is the required memory size (MB),
- d_j^{req} is the required storage space (MB).

The edge set $\mathcal{E}_\tau$ represents precedence constraints between tasks, where $(v_i, v_j) \in \mathcal{E}_\tau$ indicates that task v_i must complete before task v_j can begin execution. The weight function $\mathcal{W}_\tau : \mathcal{E}_\tau \to \mathbb{R}_+$ quantifies the data volume (in MB) that must be transferred from task v_i to task v_j, formally defined as:

$$\mathcal{W}_\tau(v_i, v_j) = \text{DataSize}(v_i \to v_j), \quad \forall (v_i, v_j) \in \mathcal{E}_\tau. \tag{7.2}$$

Each application $\tau \in \mathcal{T}_i$ requires resource allocation decisions for its constituent tasks within silo z_i. The scheduling decision for silo z_i is represented by allocation function $\boldsymbol{x}_i : \bigcup_{\tau \in \mathcal{T}_i} \mathcal{V}_\tau \to \mathcal{N}_i$, where $\boldsymbol{x}_i(v_j)$ denotes the computing entity assigned to task v_j within silo z_i.

7.3.2 Response Time Objective Function

IoT applications typically manifest as task graphs with complex dependency relationships, where the total response time is determined by the longest processing path rather than the sum of individual task times. For any processing path $p \in \mathcal{P}(\tau)$, where $\mathcal{P}(\tau)$ denotes the set of all possible execution paths from source to sink tasks in application τ, its total response time consists of three time overheads: computational overhead (actual processing time of tasks on assigned resources), communication overhead (data transmission when dependent tasks are distributed across different computing nodes), and queuing delay caused by resource contention:

$$\mathcal{T}_{\text{path}}(p, \boldsymbol{x}_i) = \sum_{v_j \in p} T_{\text{proc}}(v_j, \boldsymbol{x}_i) + \sum_{(v_j, v_k) \in p} T_{\text{comm}}(v_j, v_k, \boldsymbol{x}_i) + \sum_{v_j \in p} T_{\text{queue}}(v_j, \boldsymbol{x}_i). \tag{7.3}$$

The processing time directly depends on the task's computational requirements and the processing capacity of the assigned resource. For task v_j, its processing time on the assigned resource is:

$$T_{\text{proc}}(v_j, \boldsymbol{x}_i) = \frac{f_j^{\text{req}}}{f_{\boldsymbol{x}_i(v_j)}}. \tag{7.4}$$

When dependent tasks are assigned to different resources, data transmission becomes inevitable. Communication delay comprises two components: transmission time proportional to data volume and fixed network round-trip overhead:

$$T_{\text{comm}}(v_j, v_k, \boldsymbol{x}_i) = \frac{\mathcal{W}_\tau(v_j, v_k)}{\text{BW}_{\boldsymbol{x}_i(v_j), \boldsymbol{x}_i(v_k)}} + \text{RTT}_{\boldsymbol{x}_i(v_j), \boldsymbol{x}_i(v_k)}, \tag{7.5}$$

where BW_{n_1, n_2} represents the available bandwidth between computing entities n_1 and n_2, and RTT_{n_1, n_2} is the round-trip time.

In resource-constrained IoT environments, multiple tasks competing for the same computational resource is very common. Queuing delay reflects the impact of such

resource contention, calculated as the remaining processing time of all tasks that arrived earlier at the same resource and have not yet completed:

$$T_{\text{queue}}(v_j, \boldsymbol{x}_i) = \sum_{\substack{v_l : \boldsymbol{x}_i(v_l) = \boldsymbol{x}_i(v_j) \\ \text{arrive}(v_l) < \text{arrive}(v_j)}} T_{\text{proc}}(v_l, \boldsymbol{x}_i). \tag{7.6}$$

With the single-path latency model, we can determine the overall response time of the application. Since different paths in the DAG can be processed in parallel, the application's response time is determined by the longest path (critical path):

$$\mathcal{T}_{\text{critical}}(\tau, \boldsymbol{x}_i) = \max_{p \in \mathcal{P}(\tau)} \mathcal{T}_{\text{path}}(p, \boldsymbol{x}_i). \tag{7.7}$$

However, optimizing expected response time alone is insufficient. IoT environments exhibit high uncertainty: network condition fluctuations, device performance variations, and randomness in task arrival patterns can all lead to dramatic performance fluctuations. For critical applications, occasional extremely long delays may be more unacceptable than sustained minor delay increases. To simultaneously guarantee average performance and QoS stability, we introduce Conditional Value at Risk (CVaR) to control tail risk. CVaR_α is defined as the conditional expectation beyond the α-quantile:

$$\text{CVaR}_\alpha[X] = \mathbb{E}[X | X \geq \text{VaR}_\alpha(X)]. \tag{7.8}$$

For example, $\text{CVaR}_{0.95}$ represents the average response time of the slowest 5% of applications, effectively quantifying system performance under extreme conditions. In practice, we employ the Rockafellar-Uryasev formulation [324] for CVaR computation:

$$\text{CVaR}_\alpha[X] = \min_\eta \left\{ \eta + \frac{1}{1-\alpha} \mathbb{E}\left[(X - \eta)_+\right] \right\}, \tag{7.9}$$

where η is the threshold parameter and $(X - \eta)_+ = \max(0, X - \eta)$ captures excess beyond the threshold. The optimal η^* equals the α-quantile $\text{VaR}_\alpha(X)$. For empirical estima-

tion with batch size B, we approximate η^* using the sample α-quantile $\hat{\eta}$ and compute:

$$\widehat{\mathrm{CVaR}}_\alpha = \hat{\eta} + \frac{1}{(1-\alpha)B} \sum_{k=1}^{B} (\mathcal{T}_{\mathrm{critical},k} - \hat{\eta})_+ . \tag{7.10}$$

For silo z_i processing its application set $\mathcal{T}_i$, the response time objective function is:

$$\mathcal{L}_{\mathrm{RT}}^{(i)}(\boldsymbol{x}_i) = \sum_{\tau \in \mathcal{T}_i} [\mathbb{E}\left[\mathcal{T}_{\mathrm{critical}}(\tau, \boldsymbol{x}_i)\right] + \beta \cdot \mathrm{CVaR}_\alpha \left[\mathcal{T}_{\mathrm{critical}}(\tau, \boldsymbol{x}_i)\right]], \tag{7.11}$$

where the first term optimizes average response time and the second term controls worst-case performance across all applications in silo z_i. Parameter β balances the two objectives: when $\beta = 0$, focus is on average performance, and as β increases, more emphasis is placed on performance stability. This design ensures that scheduling strategies can achieve excellent average performance while providing reliable QoS guarantees for delay-sensitive IoT applications.

7.3.3 Energy Consumption Objective Function

We decompose the total energy consumption into three main components: computation energy, communication energy, and idle maintenance energy. Computation energy is the primary energy consumption source during task execution, directly depending on task computational intensity and resource processing capacity:

$$E_{comp}(v_j, \boldsymbol{x}_i) = P_{comp}^{(\boldsymbol{x}_i(v_j))} \cdot T_{proc}(v_j, \boldsymbol{x}_i), \tag{7.12}$$

where $P_{comp}^{(n)}$ is the computation power of resource n when executing tasks.

When dependent tasks are assigned to different resources, data transmission generates additional communication energy consumption for both sending and receiving endpoints:

$$E_{comm}(v_j, v_k, \boldsymbol{x}_i) = \left[P_{send}^{(\boldsymbol{x}_i(v_j))} + P_{recv}^{(\boldsymbol{x}_i(v_k))}\right] \cdot T_{comm}(v_j, v_k, \boldsymbol{x}_i), \tag{7.13}$$

where $P_{send}^{(n)}$ represents the transmission power of resource n when sending data, and

$P_{recv}^{(n)}$ represents the reception power of resource n when receiving data. Both endpoints consume energy for the entire communication duration $T_{comm}(v_j, v_k, \boldsymbol{x}_i)$ as they are simultaneously engaged in the data transfer process.

Even when no tasks are executing, computing resources still consume energy to maintain their available state:

$$E_{idle}(n) = P_{standby}^{(n)} \cdot T_{idle}^{(n)}, \tag{7.14}$$

where $P_{standby}^{(n)}$ is the standby power of resource n, and $T_{idle}^{(n)}$ is the idle time of resource n.

Integrating the three components, the total energy consumption of silo z_i under scheduling decision $\boldsymbol{x}_i$ is:

$$\mathcal{L}_{Energy}^{(i)}(\boldsymbol{x}_i) = \sum_{\tau \in \mathcal{T}_i} \left[\sum_{v_j \in \mathcal{V}_\tau} E_{comp}(v_j, \boldsymbol{x}_i) + \sum_{(v_j, v_k) \in \mathcal{E}_\tau} E_{comm}(v_j, v_k, \boldsymbol{x}_i) \right] + \sum_{n \in \mathcal{N}_i} E_{idle}(n). \tag{7.15}$$

7.3.4 Joint Optimization Problem

Considering that response time and energy consumption have different dimensions and numerical ranges, we first normalize the two objective functions. For a single silo z_i, the normalized objective functions are:

$$\hat{\mathcal{L}}_{RT}^{(i)}(\boldsymbol{x}_i) = \frac{\mathcal{L}_{RT}^{(i)}(\boldsymbol{x}_i) - \mathcal{L}_{RT,min}^{(i)}}{\mathcal{L}_{RT,max}^{(i)} - \mathcal{L}_{RT,min}^{(i)}}, \tag{7.16}$$

$$\hat{\mathcal{L}}_{Energy}^{(i)}(\boldsymbol{x}_i) = \frac{\mathcal{L}_{Energy}^{(i)}(\boldsymbol{x}_i) - \mathcal{L}_{Energy,min}^{(i)}}{\mathcal{L}_{Energy,max}^{(i)} - \mathcal{L}_{Energy,min}^{(i)}}, \tag{7.17}$$

where the maximum and minimum values are obtained through historical scheduling data or theoretical analysis. The single-silo optimization problem is formulated as:

$$\min_{\boldsymbol{x}_i} \mathcal{L}^{(i)}_{Total}(\boldsymbol{x}_i) = \min_{\boldsymbol{x}_i} \left[\lambda^{(i)}_{RT} \cdot \hat{\mathcal{L}}^{(i)}_{RT}(\boldsymbol{x}_i) + \lambda^{(i)}_{Energy} \cdot \hat{\mathcal{L}}^{(i)}_{Energy}(\boldsymbol{x}_i) \right], \tag{7.18}$$

where the weight coefficients satisfy $\lambda^{(i)}_{RT} + \lambda^{(i)}_{Energy} = 1$ and $\lambda^{(i)}_{RT}, \lambda^{(i)}_{Energy} \geq 0$, reflecting silo i's preference between performance and energy efficiency.

From a system-wide perspective, the optimization objective is to minimize the weighted total cost of all silos:

$$\min_{\{\boldsymbol{x}_i\}_{i=1}^{M}} \sum_{i=1}^{M} \omega_i \cdot \mathcal{L}^{(i)}_{Total}(\boldsymbol{x}_i), \tag{7.19}$$

where ω_i is the importance weight of silo i, satisfying $\sum_{i=1}^{M} \omega_i = 1$. The above optimization problem is subject to the following constraints:

Resource capacity constraints: At any given time, the total resource demands of all concurrently executing tasks on a resource cannot exceed the resource's capacity:

$$\sum_{\substack{v_j : \boldsymbol{x}_i(v_j) = n \\ t \in [t_{start}(v_j), t_{finish}(v_j)]}} (f^{req}_j, m^{req}_j, d^{req}_j) \preceq (f_n, m_n, d_n), \quad \forall t \in \mathbb{R}_+, \forall n \in \mathcal{N}_i, \tag{7.20}$$

where $\preceq$ denotes element-wise inequality (i.e., $(a_1, a_2) \preceq (b_1, b_2)$ means $a_1 \leq b_1$ and $a_2 \leq b_2$), and $t_{start}(v_j, \boldsymbol{x}_i)$ and $t_{finish}(v_j, \boldsymbol{x}_i)$ represent the start and finish times of task v_j under allocation $\boldsymbol{x}_i$.

Energy budget constraints: For resources with limited energy capacity (i.e., battery-powered IoT devices), the total energy consumption cannot exceed the available energy

budget:

$$\sum_{v_j:\boldsymbol{x}_i(v_j)=n} E_{comp}(v_j,\boldsymbol{x}_i) + \sum_{(v_j,v_k):\boldsymbol{x}_i(v_j)=n} E^{send}_{comm}(v_j,v_k,\boldsymbol{x}_i) + \sum_{(v_j,v_k):\boldsymbol{x}_i(v_k)=n} E^{recv}_{comm}(v_j,v_k,\boldsymbol{x}_i) + E_{idle}(n) \leq e_n, \quad \forall n \in \mathcal{N}_i \text{ with } e_n < \infty. \tag{7.21}$$

Deadline constraints: Each application's critical path execution time must not exceed its specified deadline:

$$\mathcal{T}_{critical}(\tau,\boldsymbol{x}_i) \leq T^{deadline}_{\tau}, \quad \forall \tau \in \mathcal{T}_i. \tag{7.22}$$

Precedence constraints: Tasks with dependencies must execute in the correct order, with predecessor tasks completing and data transmission finishing before their successors can begin:

$$t_{finish}(v_j,\boldsymbol{x}_i) + T_{comm}(v_j,v_k,\boldsymbol{x}_i) \leq t_{start}(v_k,\boldsymbol{x}_i), \quad \forall (v_j,v_k) \in \mathcal{E}_\tau, \tau \in \mathcal{T}_i. \tag{7.23}$$

Assignment constraints: Each task must be assigned to exactly one resource within its silo:

$$\boldsymbol{x}_i(v_j) \in \mathcal{N}_i, \quad \forall v_j \in \bigcup_{\tau \in \mathcal{T}_i} \mathcal{V}_\tau. \tag{7.24}$$

This optimization problem exhibits the following characteristics: First is the large-scale state space, where the system state includes resource states, task queues, and network conditions of all silos, with the state space growing exponentially with the number of silos and resource scales. Second is the delayed reward property, where the effects of scheduling decisions are often observed only after task execution completion, forming a typical delayed feedback problem. Third is the distributed coordination challenge, where silos need to cooperate effectively while maintaining autonomy, making traditional centralized optimization approaches difficult to apply.

These characteristics make the problem naturally suitable for MDP modeling, learning optimal scheduling policies through DRL via interaction with the environment. Meanwhile, the coordination requirements in distributed environments provide motivation for applying federated learning approaches.

7.3.5 MDP Formulation

We model the silo-cooperative IoT scheduling problem as a distributed MDP. The entire system can be represented as a tuple:

$$\langle \mathcal{S}, \mathcal{A}, P, \mathcal{R}, \gamma \rangle, \tag{7.25}$$

where:

- $\mathcal{S} = \mathcal{S}_1 \times \mathcal{S}_2 \times \cdots \times \mathcal{S}_M$ is the global state space.
- $\mathcal{A} = \mathcal{A}_1 \times \mathcal{A}_2 \times \cdots \times \mathcal{A}_M$ is the joint action space.
- $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the state transition probability function.
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function.
- $\gamma \in [0, 1)$ is the discount factor.

State Space Design

The local state $s_i^{(t)} \in \mathcal{S}_i$ of each silo z_i contains four key components:

$$s_i^{(t)} = \langle \mathbf{U}_i^{(t)}, \mathbf{Q}_i^{(t)}, \mathbf{W}_i^{(t)}, \mathbf{T}_i^{(t)} \rangle, \tag{7.26}$$

where $\mathbf{U}_i^{(t)}$ records the resource utilization of computing nodes within the silo, including current occupancy rates of CPU, memory, storage, and energy, providing standardized resource information. $\mathbf{Q}_i^{(t)}$ represents the current task queue lengths of computing nodes, reflecting system load distribution and potential scheduling bottlenecks. $\mathbf{W}_i^{(t)}$ describes the network topology and connectivity within the silo, with diagonal elements

recording network bandwidth of nodes and off-diagonal elements recording inter-node communication delays. $\mathbf{T}_i^{(t)}$ contains feature information of pending applications, such as the number of tasks, resource requirements, and deadline constraints.

Action Space Design

The action space $\mathcal{A}_i$ of silo z_i corresponds to task-to-resource allocation decisions. For the current task $v_{current}$ to be scheduled, the action is defined as:

$$a_i^{(t)} = \boldsymbol{x}_i(v_{current}) \in \mathcal{N}_i, \tag{7.27}$$

selecting a computing node within the silo for executing the current task.

We distinguish between hard constraints that must be physically satisfied and soft constraints that can be violated with performance penalties. Hard constraints include resource capacity (Eq. 7.20), energy budget (Eq. 7.21), precedence (Eq. 7.23), and assignment constraints (Eq. 7.24), as violations would result in physically infeasible or logically incorrect executions. Soft constraints include deadline requirements (Eq. 7.22), where violations lead to QoS degradation but tasks can still be completed.

To ensure physically feasible scheduling decisions, we define the constrained action set that satisfies all hard constraints:

$$\mathcal{A}_i^{feasible}(s_i^{(t)}) = \{a_i \in \mathcal{A}_i : \text{satisfying Eqs. 7.20, 7.21, 7.23, and 7.24}\}.$$

State Transition Function

The state transition probability function $P(s^{(t+1)}|s^{(t)}, a^{(t)})$ characterizes the system's state evolution after executing joint actions. The transition process includes deterministic factors (such as resource utilization updates and task queue changes) and stochastic factors (such as new application arrivals, task completion time fluctuations, and network condition changes).

Reward Function Design

The reward function is directly based on Eq. 7.18, using a negative cost formulation:

$$r_i^{(t)} = -\mathcal{L}_{Total}^{(i)}(\boldsymbol{x}_i^{(t)}) - C \cdot \mathbb{I}_d, \tag{7.28}$$

where $\mathbb{I}_d$ is an indicator function that equals 1 when soft constraints (deadline requirements in Eq. 7.22) are violated and 0 otherwise. C is a large positive constant that severely penalizes deadline violations to guide policy learning while maintaining decision flexibility.

Policy and Objective Function

Each silo z_i maintains a parameterized stochastic policy $\pi_{\theta_i} : \mathcal{S}_i \to \Delta(\mathcal{A}_i)$, making independent decisions based on local observations:

$$\pi_{\theta_i}(a_i|s_i) = P(a_i|s_i;\theta_i). \tag{7.29}$$

The global joint policy is defined as the product of local policies of all silos:

$$\pi(a_1,\ldots,a_M|s_1,\ldots,s_M) = \prod_{i=1}^{M} \pi_{\theta_i}(a_i|s_i). \tag{7.30}$$

The distributed optimization objective is to maximize the expected cumulative discounted reward across all silos:

$$\max_{\{\theta_i\}_{i=1}^M} \sum_{i=1}^{M} \omega_i \mathbb{E}_{\pi_{\theta_i}} \left[\sum_{t=0}^{\infty} \gamma^t r_i^{(t)} \right], \tag{7.31}$$

where ω_i is the importance weight of silo i from Eq. 7.19, satisfying $\sum_{i=1}^{M} \omega_i = 1$.

7.4 DeFRiS Framework

Silo-cooperative IoT scheduling poses three fundamental challenges for standard DRL: **(i) heterogeneous action spaces**: silos expose resource sets with different cardinalities

and semantics, preventing straightforward parameter sharing and transfer across policies; **(ii) stable and sample-efficient local optimization**: each silo must learn from limited, privacy-constrained experience under sparse and delayed rewards; updates must remain stable and keep models ready for subsequent federation; **(iii) Non-IID aggregation bias and attacks**: workload distributions differ significantly across silos, and malicious participants may inject corrupted parameters, making robust aggregation essential for effective cooperation.

As depicted in Fig. 7.2, DeFRiS addresses these challenges with three complementary components: **Action-Space-Agnostic Policy** that models resource selection via candidate scoring with masking, enabling cross-silo parameter sharing; **Silo-Optimized Local Learning** that uses advantage estimation and clipped update steps to provide stable on-policy improvement; and **Dual-Track Non-IID Robust Decentralized Aggregation**, employing gradient fingerprint-based similarity assessment and gradient tracking mechanism to mitigate aggregation bias while detecting and filtering anomalous behavior.

7.4.1 Action-Space-Agnostic Policy

Different silos possess computing resource collections of varying scales and configurations, leading to significant differences in action space dimensions. For example, an edge-computing-dominant silo might contain 20 edge nodes and 5 cloud instances, while a cloud-computing-dominant silo might contain 8 edge nodes and 15 cloud instances. Traditional policy networks require output layer weight matrices that must match the action space size, preventing parameter sharing when different silos have different action space dimensions.

Candidate Scoring Design Philosophy

To solve the heterogeneous action space problem, we redesign the policy function $\pi_{\theta_i}(a_i|s_i)$ defined in the MDP into a candidate scoring-based form. The core idea is to design a scoring network that computes suitability scores for each candidate resource node, then obtains policy probability distributions through normalization.

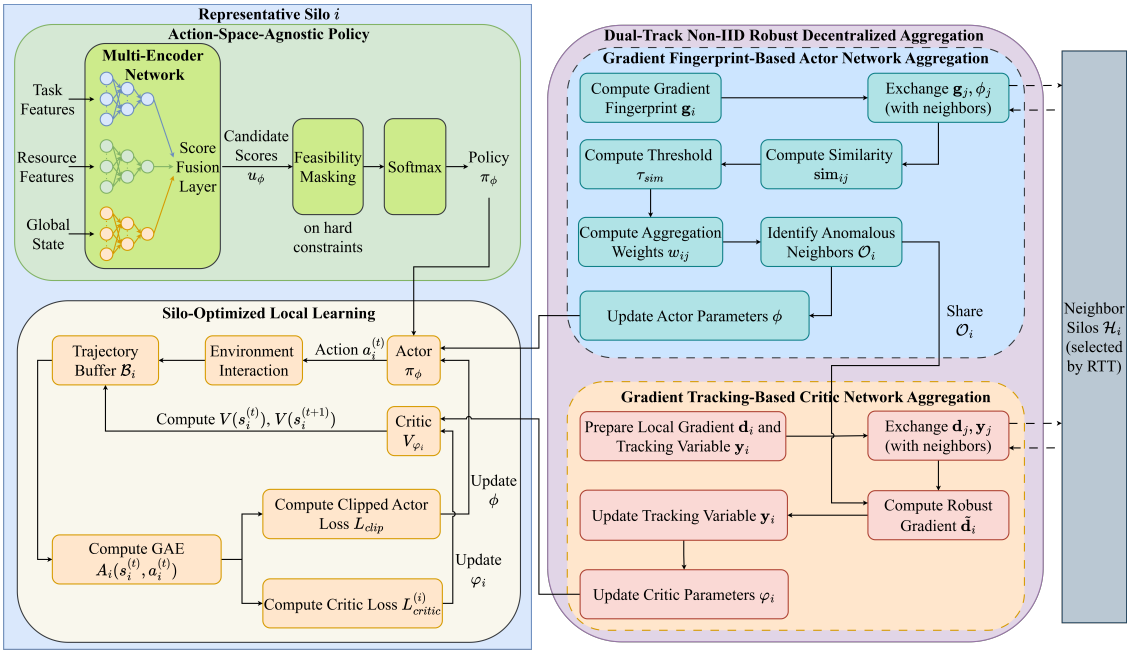


Figure 7.2: The overview of the DeFRiS framework. It consists of three synergistic components: (1) Action-Space-Agnostic Policy enables parameter sharing across heterogeneous silos via candidate scoring; (2) Silo-Optimized Local Learning ensures stable convergence under sparse rewards using GAE and clipped updates; (3) Dual-Track Non-IID Robust Decentralized Aggregation facilitates robust and similarity-aware knowledge transfer through gradient fingerprints and tracking.

Specifically, we redefine the policy function as:

$$\pi_\phi(n|s_i) = \frac{\exp(u_\phi(s_i, n))}{\sum_{n' \in \mathcal{A}_i^{feasible}(s_i)} \exp(u_\phi(s_i, n'))}, \tag{7.32}$$

where $u_\phi(s_i, n)$ is the state-resource scoring function, $n \in \mathcal{N}_i$ represents candidate resource nodes, and $\mathcal{A}_i^{feasible}(s_i)$ is the feasible action set satisfying hard constraints. The key innovation is the independence of the parameters ϕ from the resource count, enabling the same scoring function u_ϕ to handle candidate sets of arbitrary scale.

This remodeling brings three important advantages. First, parameter shareability allows all silos to use the same ϕ parameters because the scoring function only needs to process individual state-resource pairs. Second, scalability ensures that when a silo's resource configuration changes, no model retraining is needed. Third, semantic consistency enables the scoring function to learn general matching patterns between states and

resource features, and this knowledge has good transferability across different silos.

Multi-Encoder Network Architecture Design

The scoring function $u_\phi(s_i, n)$ adopts a multi-encoder architecture with multiple fully connected layers that separately encode task features, resource features, and global state:

- **Task Encoder** processes features of the current task to be scheduled:

$$\mathbf{h}_{task} = \text{ReLU}(\mathbf{W}_{task} \cdot \psi_{task}(s_i) + \mathbf{b}_{task}), \tag{7.33}$$

  where $\psi_{task}(s_i)$ extracts features of the current task such as requirements, deadline, etc.

- **Resource Encoder** processes features of candidate resource nodes:

$$\mathbf{h}_{resource} = \text{ReLU}(\mathbf{W}_{resource} \cdot \psi_{resource}(n) + \mathbf{b}_{resource}), \tag{7.34}$$

  where $\psi_{resource}(n)$ includes features of resource n such as index, capacity, current load, queue length, network conditions, etc.

- **Global State Encoder** processes system-wide state information:

$$\mathbf{h}_{global} = \text{ReLU}(\mathbf{W}_{global} \cdot \psi_{global}(s_i) + \mathbf{b}_{global}), \tag{7.35}$$

  where $\psi_{global}(s_i)$ contains global information such as overall load distribution across all resources, network topology, etc.

- **Score Fusion Layer** computes final task-resource matching scores:

$$\begin{aligned} u_\phi(s_i, n) = \mathbf{w}^T \cdot \text{ReLU}([&\mathbf{h}_{task}; \mathbf{h}_{resource}; \mathbf{h}_{global}; \\ &\mathbf{h}_{task} \odot \mathbf{h}_{resource}]), \end{aligned} \tag{7.36}$$

  where the concatenation operation $[\mathbf{h}_{task}; \mathbf{h}_{resource}; \mathbf{h}_{global}]$ captures independent features of tasks, resources, and global state, while the element-wise product $\mathbf{h}_{task} \odot$

$\mathbf{h}_{resource}$ models interactive matching effects between tasks and resources.

Feasibility Masking

We handle hard constraints through a feasibility masking mechanism. The policy function can be equivalently expressed as:

$$\begin{aligned}\pi_\phi(n|s_i) &= \frac{\exp(u_\phi(s_i,n))}{\sum_{n'\in\mathcal{A}_i^{feasible}(s_i)}\exp(u_\phi(s_i,n'))} \\ &= \frac{\exp(u_\phi(s_i,n))\cdot\mathbb{I}_i(s_i,n)}{\sum_{n'\in\mathcal{N}_i}\exp(u_\phi(s_i,n'))\cdot\mathbb{I}_i(s_i,n')},\end{aligned} \tag{7.37}$$

where $\mathbb{I}_i(s_i,n)\in\{0,1\}$ is the feasibility mask for silo i. When resource n satisfies all hard constraints for the current task in silo i, $\mathbb{I}_i(s_i,n)=1$, otherwise it equals 0. This design ensures that the policy only computes probability distributions over physically feasible action spaces while maintaining the mathematical properties of softmax normalization.

7.4.2 Silo-Optimized Local Learning

Silo-cooperative IoT scheduling faces the challenge of learning effective policies from sparse, delayed rewards. To address this, we develop a specialized local learning mechanism that combines Actor-Critic architecture with advanced training techniques to ensure both sample efficiency and stability for subsequent federated aggregation.

Actor-Critic Architecture

We adopt an Actor-Critic architecture that leverages the action-space-agnostic policy while adding a dedicated value network for stable learning. In the following discussion, we adopt time-indexed notation $(\cdot)^{(t)}$ to support the temporal learning dynamics.

The Actor network directly uses the policy $\pi_\phi(n|s_i^{(t)})$ from the Section 7.4.1 with its multi-encoder scoring function, enabling cross-silo parameter sharing:

$$\pi_\phi(n|s_i^{(t)}) = \frac{\exp(u_\phi(s_i^{(t)},n))\cdot\mathbb{I}_i(s_i^{(t)},n)}{\sum_{n'\in\mathcal{N}_i}\exp(u_\phi(s_i^{(t)},n'))\cdot\mathbb{I}_i(s_i^{(t)},n')}. \tag{7.38}$$

The Critic network employs independent parameters φ_i, specifically learning state value functions for silo i's local environment:

$$V_{\varphi_i}(s_i^{(t)}) = \mathbf{w}_{val}^T \cdot \text{ReLU}(\mathbf{W}_{val} \cdot \psi_{state}(s_i^{(t)}) + \mathbf{b}_{val}) + b_{val}, \tag{7.39}$$

where $\psi_{state}(s_i^{(t)})$ converts MDP states into unified feature representations, $\mathbf{b}_{val}$ is the hidden layer bias vector, and b_{val} is the output layer bias scalar.

Stable Local Training

In IoT scheduling environments, reward signals exhibit typical sparsity and delay characteristics, where the effects of a scheduling decision may only be observed upon completion of the entire application execution. This creates a significant credit assignment problem [325], as it becomes difficult to determine which specific scheduling decisions among a long sequence of actions contribute to the final application performance. To address these challenges and ensure training stability, we employ two specialized training techniques:

- **Generalized Advantage Estimation (GAE) Mechanism**: To solve the sparse reward problem, we employ GAE to convert sparse terminal rewards into dense intermediate learning signals. GAE combines multi-step temporal difference errors through exponential weighting:

$$A_i(s_i^{(t)}, a_i^{(t)}) = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_i^{(t+l)}, \tag{7.40}$$

  where $\delta_i^{(t)} = r_i^{(t)} + \gamma V_{\varphi_i}(s_i^{(t+1)}) - V_{\varphi_i}(s_i^{(t)})$ is the Temporal Difference (TD) error for silo i at time t. The parameter $\lambda \in [0, 1]$ controls the estimation time span: when λ approaches 0, it focuses on immediate rewards with low bias but high variance; when λ approaches 1, it considers long-term returns with unbiased but high variance estimates. This design enables each intermediate decision to receive gradient signals from final performance, effectively addressing the credit assignment challenge in sparse reward environments.

- **Clipped Update Stability Guarantee**: Policy network updates may lead to significant differences between new and old policies, easily triggering training instability in complex scheduling environments. We monitor policy change magnitude through importance sampling ratios:

$$\kappa_i(s_i^{(t)}, a_i^{(t)}) = \frac{\pi_\phi(a_i^{(t)}|s_i^{(t)})}{\pi_\phi^{old}(a_i^{(t)}|s_i^{(t)})}. \tag{7.41}$$

When $\kappa_i(s_i^{(t)}, a_i^{(t)})$ deviates far from 1, it indicates large policy differences. Such dramatic policy changes can cause the agent to take actions that are poorly understood by the current value function, leading to unstable learning or even performance collapse. To prevent this, we employ a clipped objective function that constrains the magnitude of policy updates:

$$\begin{aligned} L_{clip}^{(i)}(\phi) = \min \Big(& \kappa_i(s_i^{(t)}, a_i^{(t)}) \cdot A_i(s_i^{(t)}, a_i^{(t)}), \\ & \text{clip}(\kappa_i(s_i^{(t)}, a_i^{(t)}), 1-\epsilon, 1+\epsilon) \cdot A_i(s_i^{(t)}, a_i^{(t)}) \Big). \end{aligned} \tag{7.42}$$

The clipping function restricts ratios to the interval $[1-\epsilon, 1+\epsilon]$, ensuring policy update step sizes are controlled within safe ranges and avoiding performance degradation due to single large updates.

Algorithm 7.1 presents the complete silo-optimized local learning procedure. During the experience collection phase (Lines 4-9), each silo executes its current policy $\pi_\phi(n|s_i^{(t)})$ with feasibility masking $\mathbb{I}_i(s_i^{(t)}, n)$ to gather trajectory data while ensuring all sampled actions satisfy hard constraints, forming the training batch $\mathcal{B}_i$. In the GAE advantage estimation phase (Lines 10-16), the algorithm implements Eq. 7.40 through backward recursive computation. The Actor update phase (Lines 17-22) employs the clipped objective as designed in Eq. 7.42, where importance sampling ratios are first computed

following Eq. 7.41, then incorporated into the clipped gradient computation:

$$\begin{aligned}\nabla_\phi L_{clip}^{(i)} = \frac{1}{|\mathcal{B}_i|} \sum_{(s_i^{(t)}, a_i^{(t)}) \in \mathcal{B}_i} \\ \nabla_\phi \min \Big(\kappa_i(s_i^{(t)}, a_i^{(t)}) \cdot A_i(s_i^{(t)}, a_i^{(t)}), \\ \text{clip}(\kappa_i(s_i^{(t)}, a_i^{(t)}), 1-\epsilon, 1+\epsilon) \cdot A_i(s_i^{(t)}, a_i^{(t)}) \Big).\end{aligned} \tag{7.43}$$

Actor parameter updates follow gradient ascent: $\phi \leftarrow \phi + \alpha_\phi \nabla_\phi L_{clip}^{(i)}$. The Critic update phase (Lines 23-26) optimizes value prediction accuracy by minimizing the temporal difference loss function:

$$L_{critic}^{(i)} = \frac{1}{|\mathcal{B}_i|} \sum_{s_i^{(t)} \in \mathcal{B}_i} \frac{1}{2} (V_{\varphi_i}(s_i^{(t)}) - V_{target}(s_i^{(t)}))^2, \tag{7.44}$$

where target values $V_{target}(s_i^{(t)}) = r_i^{(t)} + \gamma V_{\varphi_i}(s_i^{(t+1)})$ incorporate both immediate scheduling rewards and discounted future value estimates. Critic parameter updates follow gradient descent: $\varphi_i \leftarrow \varphi_i - \alpha_\varphi \nabla_{\varphi_i} L_{critic}^{(i)}$.

7.4.3 Dual-Track Non-IID Robust Decentralized Aggregation

In distributed IoT scheduling environments, silos face significantly different task workloads and resource configurations, creating severe Non-IID challenges. Additionally, distributed environments may contain faulty nodes, network attacks, or malicious behavior, requiring robust aggregation mechanisms. This section proposes a decentralized robust aggregation mechanism that achieves effective cross-silo knowledge transfer through similarity-aware, anomaly detection, and gradient tracking techniques, while resisting malicious nodes.

Decentralized Communication and Aggregation Protocol

To achieve privacy protection and fault tolerance, we adopt a fully decentralized peer-to-peer aggregation architecture. The system constructs a time-varying overlay network $\mathcal{G}_{comm}(t) = (\mathcal{Z}, \mathcal{E}_{comm}(t))$, where the edge set $\mathcal{E}_{comm}(t)$ adjusts dynamically based on

Algorithm 7.1: Silo-Optimized Local Learning

Input: Silo z_i, initial parameters $\phi^{(0)}, \varphi_i^{(0)}$, learning rates $\alpha_\phi, \alpha_\varphi$, clipping parameter ϵ, GAE parameter λ, discount factor γ

Output: Trained parameters ϕ, φ_i

for *training episode* $e = 1, 2, \ldots$ **do**
  Initialize trajectory buffer $\mathcal{B}_i = \{\}$
  `/* Experience Collection Phase */`
  for *step* $t = 1, 2, \ldots, T$ **do**
    Observe current state $s_i^{(t)}$ from local environment
    Sample action:
    $a_i^{(t)} \sim \pi_\phi(n|s_i^{(t)})$ using feasibility masking $\mathbb{I}_i(s_i^{(t)}, n)$
    Execute action $a_i^{(t)}$, observe reward $r_i^{(t)}$ and next state $s_i^{(t+1)}$
    Store transition $(s_i^{(t)}, a_i^{(t)}, r_i^{(t)}, s_i^{(t+1)})$ in $\mathcal{B}_i$
  end for
  `/* GAE Advantage Estimation Phase */`
  Initialize advantage accumulator $A = 0$
  for $(s_i^{(t)}, a_i^{(t)}, r_i^{(t)}, s_i^{(t+1)}) \in \mathcal{B}_i$ *in reverse chronological order* **do**
    Compute TD error:
    $\delta_i^{(t)} = r_i^{(t)} + \gamma V_{\varphi_i}(s_i^{(t+1)}) - V_{\varphi_i}(s_i^{(t)})$
    Update accumulator:
    $A = \delta_i^{(t)} + \gamma\lambda A$
    Set advantage:
    $A_i(s_i^{(t)}, a_i^{(t)}) = A$
  end for
  `/* Actor Update Phase with Stability Control */`
  Store current policy parameters: $\phi^{old} = \phi$
  Compute importance sampling ratios:
  $\kappa_i(s_i^{(t)}, a_i^{(t)}) = \frac{\pi_\phi(a_i^{(t)}|s_i^{(t)})}{\pi_{\phi^{old}}(a_i^{(t)}|s_i^{(t)})}, \forall (s_i^{(t)}, a_i^{(t)}) \in \mathcal{B}_i$
  Compute clipped policy objective:
  $L_{clip}^{(i)} =$
  $\frac{1}{|\mathcal{B}_i|} \sum_{(s_i^{(t)}, a_i^{(t)}) \in \mathcal{B}_i} \min(\kappa_i(s_i^{(t)}, a_i^{(t)}) \cdot A_i(s_i^{(t)}, a_i^{(t)}), \text{clip}(\kappa_i(s_i^{(t)}, a_i^{(t)}), 1-\epsilon, 1+\epsilon) \cdot A_i(s_i^{(t)}, a_i^{(t)}))$
  Update Actor parameters:
  $\phi \leftarrow \phi + \alpha_\phi \nabla_\phi L_{clip}^{(i)}$
  `/* Critic Update Phase */`
  Compute value targets:
  $V_{target}(s_i^{(t)}) = r_i^{(t)} + \gamma V_{\varphi_i}(s_i^{(t+1)}), \forall (s_i^{(t)}, a_i^{(t)}, r_i^{(t)}, s_i^{(t+1)}) \in \mathcal{B}_i$
  Compute critic loss:
  $L_{critic}^{(i)} = \frac{1}{|\mathcal{B}_i|} \sum_{s_i^{(t)} \in \mathcal{B}_i} \frac{1}{2} (V_{\varphi_i}(s_i^{(t)}) - V_{target}(s_i^{(t)}))^2$
  Update Critic parameters:
  $\varphi_i \leftarrow \varphi_i - \alpha_\varphi \nabla_{\varphi_i} L_{critic}^{(i)}$
end for

network conditions.

Each silo z_i randomly samples k_{sample} candidate silos for Round-Trip Time (RTT) measurement during initialization, where $k_{sample} < M$. Based on the delay measurement results, each silo z_i selects the d_{max} silos with the lowest latency as neighbors, forming the neighbor set $\mathcal{H}_i(t)$. Therefore, each silo maintains a neighbor set $\mathcal{H}_i(t) \subseteq \mathcal{Z} \setminus \{z_i\}$ satisfying the degree constraint $|\mathcal{H}_i(t)| \leq d_{max}$, ensuring communication efficiency and fault tolerance.

Parameter propagation follows a gossip protocol. In each communication round, silo z_i randomly selects partial neighbors from $\mathcal{H}_i(t)$ for parameter exchange. During parameter exchange, neighbor z_j attaches summary information of its current neighbor list, including neighbor identifiers and corresponding RTT values. Silo z_i identifies potential better neighbor candidates based on this recommendation information, selecting those candidates whose RTT is significantly lower than the current worst neighbor for connectivity testing. If the actual RTT of a new candidate is indeed better, it replaces the neighbor with the highest RTT in the current neighbor set, achieving gradual network topology optimization.

Recognizing the different characteristics of the Actor-Critic architecture, we design a dual-track aggregation strategy with periodic aggregation (every Ω local training rounds). The Actor network generates generalizable scheduling patterns and leverages cross-silo policy knowledge; the Critic network evaluates local state values and needs to maintain accurate modeling of the local environment, thus adopting gradient tracking to preserve personalization.

Gradient Fingerprint-Based Actor Network Aggregation

The Actor network needs to learn generalizable scheduling policies, but different silos have varying contribution values. Directly averaging all neighbor parameters may introduce knowledge unsuitable for the local environment, leading to performance degradation. Meanwhile, in open distributed environments, some silos may send abnormal parameters due to hardware failures, network attacks, or malicious behavior, further threatening aggregation effectiveness. Therefore, we design a robust aggregation mech-

anism based on similarity awareness and anomaly detection that can prioritize learning policy knowledge from silos with similar environments while resisting attacks.

We propose a gradient fingerprint-based similarity measurement to capture the strategic preference patterns of different silos. The fundamental insight is that silos learning effective resource selection policies will exhibit consistent gradient patterns with respect to resource features, reflecting their learned preferences for specific resource attributes. Silos demonstrating similar preference patterns in their optimization dynamics are more likely to benefit from mutual policy knowledge transfer.

During each backpropagation step, the gradient of the candidate scoring function $u_\phi(s_t, n_t)$ with respect to the resource feature encoding $\psi_{\text{resource}}(n_t)$ reveals the model's sensitivity and preference intensity toward different resource attributes. We construct a gradient fingerprint by averaging the resource feature gradients over the most recent K training samples:

$$\mathbf{g}_i = \frac{1}{K} \sum_{t=1}^{K} \frac{\partial u_\phi(s_t, n_t)}{\partial \psi_{\text{resource}}(n_t)}. \tag{7.45}$$

Each component of the gradient fingerprint vector $\mathbf{g}_i \in \mathbb{R}^{d_{\text{resource}}}$ corresponds to a resource feature dimension, with its magnitude indicating the strategic importance and optimization direction that silo i's policy assigns to that particular feature. This representation exhibits three properties desirable for heterogeneous federated learning: *(i) cross-silo comparability*, since $\mathbf{g}_i \in \mathbb{R}^{d_{\text{resource}}}$ is defined in the shared resource-feature space $\psi_{\text{resource}}(\cdot)$ and is thus comparable across silos regardless of candidate-set size or identity; *(ii) negligible marginal cost*, because the required gradients are produced by standard backpropagation and averaged locally, with no extra forward passes or cross-silo data exchange; and *(iii) temporal adaptivity*, as maintaining $\mathbf{g}_i$ as a sliding-window average over recent updates makes it responsive to policy evolution and workload shifts, providing a low-latency summary of emerging preference patterns.

Inter-silo similarity is computed using cosine similarity to measure gradient direction alignment:

$$\text{sim}_{ij} = \frac{\mathbf{g}_i^\top \mathbf{g}_j}{\|\mathbf{g}_i\|\|\mathbf{g}_j\|}. \tag{7.46}$$

Cosine similarity captures gradient direction consistency while maintaining invariance to magnitude differences, ensuring that similarity measurement is unaffected by variations in training progress or data scale across silos. A high similarity score indicates that both silos have developed consistent preferences regarding resource attribute importance, suggesting compatible optimization objectives.

To identify suspicious neighbors, we also employ the gradient fingerprint for anomaly detection. In distributed environments, normally cooperating silos should exhibit consistent resource preference patterns, while abnormal silos (whether due to failures or malicious behavior) may demonstrate dramatically different optimization objectives that manifest as divergent gradient fingerprints. We identify anomalous neighbors based on similarity thresholds τ_{sim}:

$$\mathcal{O}_i = \{j \in \mathcal{H}_i : \text{sim}_{ij} < \tau_{\text{sim}}\}, \tag{7.47}$$

where τ_{sim} is set using robust statistics to account for the non-Gaussian distribution of cosine similarity scores:

$$\begin{aligned}\tau_{\text{sim}} = \text{median}(\{\text{sim}_{ij} : j \in \mathcal{H}_i\}) - \\ \xi \cdot \text{MAD}(\{\text{sim}_{ij} : j \in \mathcal{H}_i\}),\end{aligned} \tag{7.48}$$

where MAD is the Median Absolute Deviation:

$$\begin{aligned}\text{MAD}(\{\text{sim}_{ij} : j \in \mathcal{H}_i\}) = \text{median}(\{|\text{sim}_{ij} \\ - \text{median}(\{\text{sim}_{ik} : k \in \mathcal{H}_i\})| : j \in \mathcal{H}_i\}),\end{aligned} \tag{7.49}$$

and ξ is a robustness parameter. This robust approach provides reliable anomaly detection for bounded similarity measures without distributional assumptions.

Aggregation weights are computed through softmax normalization of similarity scores

for non-anomalous neighbors:

$$w_{ij} = \frac{\exp(\text{sim}_{ij}/\nu)}{\sum_{j' \in \mathcal{H}_i \setminus \mathcal{O}_i} \exp(\text{sim}_{ij'}/\nu)}, \tag{7.50}$$

where $\nu > 0$ is a temperature parameter controlling the concentration of the similarity distribution.

Actor parameter updates employ robust weighted aggregation. First, anomalous neighbors are excluded, then weighted aggregation is performed on the remaining neighbors' parameters:

$$\phi^{(k+1)} = (1 - \alpha_{\text{agg}}) \cdot \phi^{(k)} + \alpha_{\text{agg}} \cdot \sum_{j \in \mathcal{H}_i \setminus \mathcal{O}_i} w_{ij} \cdot \phi_j^{(k)}, \tag{7.51}$$

where $\alpha_{\text{agg}} \in (0, 1)$ is the aggregation rate for the Actor network.

Gradient Tracking-Based Critic Network Aggregation

The core function of the Critic network $V_{\varphi_i}(s_i)$ is to accurately evaluate the local state values of silo z_i. Direct parameter averaging would destroy the accuracy of value functions because different silos have different state value distributions. To address this challenge, we propose a gradient tracking mechanism that enables silos to maintain personalized value functions while benefiting from neighbors' optimization directions.

Each silo z_i maintains two key variables: the local gradient $\mathbf{d}_i^{(k)} = \nabla_{\varphi_i} L_{critic}^{(i)}(\varphi_i^{(k)})$ reflecting the local optimization direction under current parameters, and the tracking variable $\mathbf{y}_i^{(k)}$ accumulating gradient information from distributed cooperation. To resist abnormal gradients, we use the same anomalous neighbor set $\mathcal{O}_i$. Each silo computes its own robust gradient by averaging filtered neighbor gradients:

$$\tilde{\mathbf{d}}_i^{(k)} = \frac{1}{|\mathcal{H}_i \setminus \mathcal{O}_i| + 1} \sum_{j \in (\mathcal{H}_i \setminus \mathcal{O}_i) \cup \{i\}} \mathbf{d}_j^{(k)}. \tag{7.52}$$

The update of tracking variables $\mathbf{y}_i^{(k)}$ follows:

$$\mathbf{y}_i^{(k+1)} = \sum_{j \in \mathcal{H}_i \cup \{i\}} c_{ij} \left(\mathbf{y}_j^{(k)} + \tilde{\mathbf{d}}_j^{(k+1)} - \tilde{\mathbf{d}}_j^{(k)} \right). \tag{7.53}$$

This formula design has three key components. First are the uniform mixing weights c_{ij}, which are chosen to satisfy the row stochastic condition $\sum_{j=1}^{M} c_{ij} = 1$. Specifically, the weights are defined based on the network topology as $c_{ij} = \frac{1}{|\mathcal{H}_i|+1}$ for $j \in \mathcal{H}_i \cup \{i\}$ and $c_{ij} = 0$ otherwise. This choice ensures balanced information propagation and convergence to a network-wide consensus while maintaining computational simplicity. Second, the tracking variable $\mathbf{y}_j^{(k)}$ carries historical cooperative gradient information from neighbor silos, enabling cross-silo optimization experience transfer. Most critically, the gradient difference term $\tilde{\mathbf{d}}_j^{(k+1)} - \tilde{\mathbf{d}}_j^{(k)}$ captures the robust gradient change of neighbors from round k to $k+1$, reflecting new optimization directions discovered by neighbors or adaptive adjustments to environmental changes. This design enables each silo to perceive neighbors' optimization dynamics: when neighbors discover better optimization directions, the gradient difference term transmits this improvement information; when environmental changes cause gradient adjustments, other silos can promptly perceive and adjust accordingly. Through mixing and accumulation of tracking variables, the optimization wisdom of the entire network is shared while each silo maintains the ability to adapt to local environments.

Critic parameters are updated based on robust tracking variables:

$$\varphi_i^{(k+1)} = \varphi_i^{(k)} - \alpha_{cag} \cdot \mathbf{y}_i^{(k+1)}, \tag{7.54}$$

where α_{cag} is the aggregation learning rate for the Critic network.

Algorithm 7.2 presents the complete dual-track Non-IID robust decentralized aggregation. It first constructs the communication topology in the initialization phase (Lines 2-7) by selecting d_{max} lowest-latency neighbors through RTT measurements for each silo, and initializes gradient fingerprint buffers and tracking variables. In the local training phase (Lines 10-18), each silo performs Ω rounds of Actor-Critic training in parallel (invoking Algorithm 7.1), while collecting resource feature gradients and maintaining a

sliding window of size K, then computes the gradient fingerprint as a compact representation of policy preferences upon training completion. In the aggregation phase (Lines 20-42), each silo first exchanges gradient fingerprints, model parameters, and tracking variables with neighbors (Lines 22-24), then computes similarity based on gradient fingerprints and performs robust anomaly detection through MAD to identify suspicious neighbors (Lines 26-32). For the Actor network, similarity-weighted aggregation is employed, where aggregation weights are assigned based on policy similarity after excluding anomalous neighbors (Lines 34-35); for the Critic network, the gradient tracking mechanism is adopted, achieving optimization momentum sharing through robust gradient averaging and tracking variable updates (Lines 37-38). Finally, the communication topology is dynamically optimized based on neighbor recommendations and RTT improvements (Line 41).

7.5 Performance Evaluation

This section presents a comprehensive experimental evaluation of DeFRiS in realistic silo-cooperative IoT environments. We first describe the experimental setup and hyperparameter configuration, then evaluate DeFRiS across five dimensions: convergence performance, ablation study, QoS guarantee, scalability, and robustness in adversarial environments.

7.5.1 Experiment Setup

This subsection describes the distributed silo-cooperative testbed, IoT application workloads, and baseline approaches.

Practical Experiment Environment

To validate the effectiveness of DeFRiS in realistic silo-cooperative IoT environments, we establish a distributed experimental testbed comprising 20 autonomous computing silos, each with different resource configurations and workload characteristics. Each silo operates as an independent management entity with complete autonomy over its

Algorithm 7.2: Dual-Track Non-IID Robust Decentralized Aggregation

Input: Silos $\mathcal{Z} = \{z_1, z_2, \ldots, z_M\}$, aggregation period Ω, neighbor size d_{max}, similarity window K, temperature ν, robustness parameter ξ, aggregation rates $\alpha_{\text{agg}}, \alpha_{\text{cag}}$
Output: Updated Actor parameters $\{\phi_i\}$ and Critic parameters $\{\varphi_i\}$

`/* Initialization: Network Topology Construction */`
for *each silo* $z_i \in \mathcal{Z}$ **do**
  Sample k_{sample} candidates, measure RTT
  Select $d_{\max}$ lowest-latency neighbors $\rightarrow \mathcal{H}_i(0)$
  Initialize gradient fingerprint buffer $\mathcal{B}_i^{\text{grad}} = \{\}$
  Initialize tracking variable $\mathbf{y}_i^{(0)} = \mathbf{0}$
end for
for *aggregation round* $r = 1, 2, \ldots$ **do**
  `/* Local Training Phase (Ω rounds) */`
  for *each silo* z_i **do in parallel**
    for *local step* $k = 1, 2, \ldots, \Omega$ **do**
      Perform local Actor-Critic training (Algorithm 7.1)
      Collect resource feature gradient:
      $\nabla_{\psi_{\text{resource}}(n_k)} u_\phi(s_k, n_k)$
      Add to buffer:
      $\mathcal{B}_i^{\text{grad}} \leftarrow \mathcal{B}_i^{\text{grad}} \cup \{\nabla_{\psi_{\text{resource}}(n_k)} u_\phi(s_k, n_k)\}$
      Keep only most recent K gradients in $\mathcal{B}_i^{\text{grad}}$
    end for
    Compute gradient fingerprint:
    $\mathbf{g}_i = \frac{1}{K} \sum_{\mathbf{g} \in \mathcal{B}_i^{\text{grad}}} \mathbf{g}$
  end forpar
  `/* Aggregation Phase */`
  for *each silo* z_i **do**
    `/* Information Exchange with All Neighbors */`
    for *each neighbor* $z_j \in \mathcal{H}_i$ **do**
      Exchange gradient fingerprints $\mathbf{g}_j$, Actor parameters $\phi_j^{(r)}$, Critic gradients $\mathbf{d}_j^{(r)}$, tracking variables $\mathbf{y}_j^{(r)}$, and neighbor recommendations
    end for
    `/* Similarity Assessment and Anomaly Detection */`
    for *each neighbor* $z_j \in \mathcal{H}_i$ **do**
      Compute similarity:
      $\text{sim}_{ij} = \frac{\mathbf{g}_i^\top \mathbf{g}_j}{\|\mathbf{g}_i\| \|\mathbf{g}_j\|}$
    end for
    Compute median similarity:
    $\text{med}_i = \text{median}(\{\text{sim}_{ij} : j \in \mathcal{H}_i\})$
    Compute MAD:
    $\text{MAD}_i = \text{median}(\{|\text{sim}_{ij} - \text{med}_i| : j \in \mathcal{H}_i\})$
    Set anomaly threshold:
    $\tau_{\text{sim}} = \text{med}_i - \xi \cdot \text{MAD}_i$
    Identify anomalous neighbors:
    $\mathcal{O}_i = \{j \in \mathcal{H}_i : \text{sim}_{ij} < \tau_{\text{sim}}\}$
    `/* Actor Network Aggregation */`
    Compute aggregation weights:
    $w_{ij} = \frac{\exp(\text{sim}_{ij}/\nu)}{\sum_{j' \in \mathcal{H}_i \setminus \mathcal{O}_i} \exp(\text{sim}_{ij'}/\nu)}$
    Update Actor parameters:
    $\phi_i^{(r+1)} = (1 - \alpha_{\text{agg}}) \cdot \phi_i^{(r)} + \alpha_{\text{agg}} \cdot \sum_{j \in \mathcal{H}_i \setminus \mathcal{O}_i} w_{ij} \cdot \phi_j^{(r)}$
    `/* Critic Network Aggregation */`
    Compute robust gradients:
    $\tilde{\mathbf{d}}_i^{(r)} = \frac{1}{|\mathcal{H}_i \setminus \mathcal{O}_i| + 1} \sum_{j \in (\mathcal{H}_i \setminus \mathcal{O}_i) \cup \{i\}} \mathbf{d}_j^{(r)}$
    Update tracking variables:
    $\mathbf{y}_i^{(r+1)} = \sum_{j \in \mathcal{H}_i \cup \{i\}} c_{ij} (\mathbf{y}_j^{(r)} + \tilde{\mathbf{d}}_j^{(r+1)} - \tilde{\mathbf{d}}_j^{(r)})$
    Update Critic parameters:
    $\varphi_i^{(r+1)} = \varphi_i^{(r)} - \alpha_{\text{cag}} \cdot \mathbf{y}_i^{(r+1)}$
    `/* Dynamic Neighbor Optimization */`
    Optimize neighbor set using received recommendations and RTT improvements
  end for
end for

local resource scheduling decisions while participating in the decentralized federated learning protocol for cooperative optimization.

Each computing silo contains a heterogeneous mix of IoT devices, edge servers, and cloud instances, forming a realistic three-tier computing hierarchy:

Cloud Layer: We deploy cloud instances from diverse providers to ensure environmental heterogeneity, including Nectar cloud instances (AMD EPYC processors, configurations ranging from 2 cores @2.0GHz 8GB RAM to 32 cores @2.0GHz 128GB RAM), AWS cloud instances (Intel Xeon processors, configurations ranging from 1 core @2.4GHz 1GB RAM to 16 cores @2.5GHz 64GB RAM), and Microsoft Azure cloud instances (Intel Xeon processors, configurations ranging from 1 core @2.3GHz 1GB RAM to 24 cores @2.4GHz 96GB RAM).

Edge Layer: We configure various edge computing devices, including M1 Pro processor-based devices (8 cores, 16GB RAM), Intel Core i7 processor-equipped devices (8 cores @2.3GHz, 16GB RAM), Intel Core i9 processor devices (8 cores @2.5GHz, 32GB RAM), and Intel Core i5 processor devices (6 cores @2.8GHz, 8GB RAM) with different configurations.

IoT Device Layer: We deploy Raspberry Pi devices (Pi OS, Broadcom BCM2837 quad-core @1.2GHz, 1GB RAM), virtual machines, and Docker containers equipped with cameras and IP cameras. Some IoT devices are equipped with limited battery capacity (simulated as 10,000-50,000 Joules), while others are connected to stable power supplies.

To reflect inter-silo heterogeneity, we configure different resource combinations for different silos: Silos 1-5 primarily contain cloud computing resources (70% cloud instances), Silos 6-10 are edge-computing dominant (60% edge nodes), Silos 11-15 adopt balanced configurations (evenly distributed resources across all tiers), and Silos 16-20 are IoT-device dense (over 50% IoT devices).

Network connections exhibit realistic latency and bandwidth variations, reflecting real-world deployment scenarios. The latency between IoT devices and edge devices ranges from 1-6ms with a bandwidth of 10-25 MB/s. Network characteristics between IoT devices and cloud instances vary by cloud service provider, with latency ranging from 6-25ms and bandwidth from 14-22MB/s. Communication latency between edge

devices and cloud instances ranges from 6-25ms with a bandwidth of 15-22MB/s.

For silo-cooperative learning communication, each silo maintains up to 5 neighbor connections ($d_{max} = 5$), dynamically selecting optimal neighbors based on RTT measurements. For energy monitoring, we use the eco2AI [246] toolkit to implement accurate real-time power measurement. In the response time objective function Eq. 7.11, CVaR risk control parameters are set to $\alpha = 0.95$ and $\beta = 0.3$, ensuring tail risk control while optimizing average performance. In the joint optimization objective function Eq. 7.18, we set weight parameters $\lambda_{RT}^{(i)} = 0.5$ and $\lambda_{Energy}^{(i)} = 0.5$ to balance response time and energy consumption optimization objectives.

IoT Application Workloads

To comprehensively evaluate DeFRiS across realistic silo-cooperative scenarios, we deploy a diverse suite of IoT applications adapted from production deployment requirements:

- *VoiceHomeController*: speech-to-text and intent recognition for smart home automation, combining `torchaudio` [296] and a lightweight Transformer decoder; workload scaled by audio-chunk length.
- *IndustrialNoiseMonitor*: acoustic amplitude tracking for factory equipment health monitoring, implemented with `librosa` [293]; workload scaled by analysis-window length.
- *DriverFatigueMonitor*: cascaded face-and-eye detection with fatigue state classification for automotive safety systems, using `OpenCV` [300] and `dlib` [304]; workload scaled by input resolution and cascade depth.
- *MedicalImagePreprocessor*: batch medical image filtering, enhancement, and standardization for telehealth diagnostics, utilizing `PIL` [299] and `OpenCV` [300]; workload scaled by batch size and processing pipeline depth.
- *GestureInteractionService*: HSV-based hand-gesture tracking for AR/VR interaction systems using `OpenCV` [300]; workload scaled by frame rate and tracking-window size.

- *CustomerFeedbackAnalyzer*: real-time sentiment analysis on customer reviews for retail analytics, using `TextBlob` [294] and `NLTK` [295]; workload scaled by text-block size.
- *ElderlyFallDetection*: pose-estimation-based fall detection for elderly care monitoring, implemented with `TensorFlow Lite` [301, 302]; workload scaled by sampling rate and model capacity.
- *TrafficSignRecognition*: video-based traffic sign detection and text recognition for ADAS systems, combining `EasyOCR` [305] and `OpenCV` [300]; workload scaled by clip length and model precision.
- *EdgeDataTransmitter*: adaptive data compression for bandwidth-constrained IoT gateways, using `zlib` [297]/`gzip` [298]; workload scaled by file size and compression level.
- *SecuritySurveillance*: real-time face detection for access control and intrusion monitoring, implemented with `OpenCV` [300] and `MediaPipe` [303]; workload scaled by frame resolution and detection frequency.

To reflect workload heterogeneity in real-world deployments, different silo types tend to receive applications matching their resource profiles: cloud-dominant silos (1-5) more frequently receive compute-intensive tasks (e.g., MedicalImagePreprocessor processing high-resolution imagery, CustomerFeedbackAnalyzer analyzing large-scale text datasets); edge-dominant silos (6-10) more often execute latency-sensitive applications (e.g., DriverFatigueMonitor processing real-time video streams, VoiceHomeController providing instant voice responses); balanced silos (11-15) receive a mixture of application types; IoT-dense silos (16-20) primarily deploy energy-optimized variants (e.g., ElderlyFallDetection using pruned lightweight pose models, IndustrialNoiseMonitor employing downsampled audio processing to conserve energy). By systematically varying workload parameters (e.g., video resolution from 480p to 4K, audio sampling rate from 8kHz to 48kHz, batch size from 10 to 500, model complexity from tiny to large), we generate a comprehensive set of application instances that stress-test DeFRiS across diverse IoT scheduling scenarios.

Baseline Approaches

We implement DeFRiS using ReinFog[31], a modular framework for DRL-based resource management that supports both centralized and distributed learning techniques development and integration in practical edge-fog-cloud environments. To comprehensively evaluate DeFRiS, we compare it against four representative state-of-the-art approaches:

- **MCM-FDRL** [323]: A federated DRL framework where independent clients locally train DQN models and update a global model via periodic parameter averaging. We adapt it to multi-silo scenarios by treating each silo as a client and modifying the action design and reward functions.

- **TF-DDRL** [54]: A distributed DRL approach employing Transformer and IMPALA framework for asynchronous training. We extend its architecture to multi-silo scenarios by deploying distributed actors across heterogeneous silos to asynchronously feed the central learner.

- **VCPN** [321]: A multi-agent DRL approach based on MADDPG using centralized training with shared replay buffers and decentralized execution. We adapt it to multi-silo scenarios by treating each silo as an agent and modifying the state representation and reward functions.

- **D3QN-SEIDEL** [318]: A single-agent DRL approach based on D3QN optimizing task offloading through separate value and advantage streams. We deploy independent agents per silo and adapt the state-action space and reward design for each local environment.

These baselines are strategically selected to cover the full spectrum of coordination paradigms in IoT scheduling: MCM-FDRL represents federated learning with global parameter aggregation [322, 323]; TF-DDRL exemplifies parallelized centralized learning where distributed actors asynchronously collect experience for a shared policy [27, 54, 287]; VCPN adopts the multi-agent paradigm with Centralized Training and Decentralized Execution (CTDE) to handle inter-agent interactions [319–321]; and D3QN-SEIDEL

serves as the baseline for independent single-agent learning without external coordination [26, 235, 283, 317, 318]. This diverse selection enables a comprehensive evaluation of DeFRiS across distinct methodological approaches.

7.5.2 Hyperparameter Configuration

To ensure optimal performance, we conduct systematic hyperparameter tuning for DeFRiS and all baseline approaches using grid search. All approaches share identical network architectures where applicable to ensure fair comparison. Table 7.2 summarizes the key hyperparameters for DeFRiS.

Table 7.2: Hyperparameter Configuration

Parameter	Value
Objective Function	
CVaR risk level (α)	0.95
CVaR weight (β)	0.3
Response time weight ($\lambda_{RT}^{(i)}$)	0.5
Energy consumption weight ($\lambda_{Energy}^{(i)}$)	0.5
Local Training	
Discount factor (γ)	0.99
Actor learning rate (α_ϕ)	3e-4
Critic learning rate (α_φ)	1e-3
GAE parameter (λ)	0.95
Clipping parameter (ϵ)	0.2
Network Architecture	
Hidden layers	2
Hidden units	128-512
Activation function	ReLU
Federated Aggregation	
Maximum neighbors (d_{max})	5
Candidate sampling size (k_{sample})	10
Gradient fingerprint window (K)	100
Similarity temperature (ν)	0.1
Robustness parameter (ξ)	3.0
Actor aggregation rate (α_{agg})	0.3
Critic aggregation rate (α_{cag})	0.1

7.5.3 Experimental Results and Analysis

We evaluate DeFRiS through convergence performance, ablation study, QoS guarantee analysis, scalability evaluation, and robustness in adversarial environments. All exper-

iments are repeated 10 times with different random seeds, and the reported results represent the mean performance averaged over all runs.

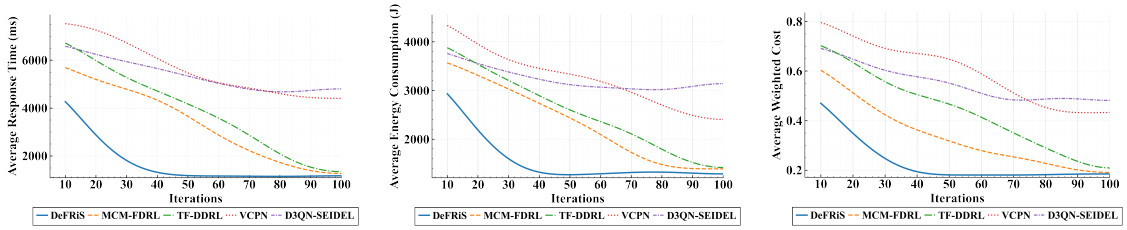
Convergence Performance

Figure 7.3 presents the convergence behavior of all approaches across 100 training iterations, evaluated in terms of response time, energy consumption, and weighted cost. All metrics represent the average performance across 20 silos, reflecting system-level overall performance. DeFRiS consistently demonstrates superior convergence speed and final performance across all three metrics.

Response time optimization. As shown in Figure 7.3a, DeFRiS achieves a final average response time of around 1170 ms, representing 6.4% and 13.3% improvements over MCM-FDRL (around 1250 ms) and TF-DDRL (around 1350 ms), respectively, and substantial gains of 73.4% and 75.6% compared to VCPN (around 4400 ms) and D3QN-SEIDEL (around 4800 ms). DeFRiS exhibits a rapid descent during the first 40 iterations and stabilizes near the optimal solution by iteration 50. In contrast, MCM-FDRL and TF-DDRL require around 100 iterations to achieve comparable stability, while VCPN and D3QN-SEIDEL remain trapped at substantially higher response times throughout training, reflecting optimization difficulties in multi-silo Non-IID environments.

Energy efficiency. Figure 7.3b demonstrates that DeFRiS achieves a final energy consumption of around 1290 J, saving 7.2% and 9.2% compared to MCM-FDRL (around 1390 J) and TF-DDRL (around 1420 J), respectively, and achieving substantial reductions of 46.3% and 58.4% over VCPN (around 2400 J) and D3QN-SEIDEL (around 3100 J). DeFRiS exhibits rapid energy optimization during the first 40 iterations and stabilizes by iteration 50. In contrast, MCM-FDRL and TF-DDRL require nearly 100 iterations to converge, while VCPN cannot converge within 100 iterations and maintains consistently high energy consumption throughout training. Notably, D3QN-SEIDEL exhibits an energy rebound phenomenon after iteration 80, demonstrating the instability of independent learning without cross-silo knowledge transfer.

Weighted cost performance. The joint optimization metric in Figure 7.3c shows that DeFRiS achieves the lowest final cost of around 0.185, outperforming MCM-FDRL



(a) Response Time **(b)** Energy Consumption **(c)** Weighted Cost

Figure 7.3: Convergence performance comparison across 100 training iterations.

(around 0.191), TF-DDRL (around 0.209), VCPN (around 0.433), and D3QN-SEIDEL (around 0.482) by 3.0%, 11.5%, 57.3%, and 61.6%, respectively. Similarly, DeFRiS demonstrates rapid optimization during the first 40 iterations and stabilizes by iteration 50. In contrast, MCM-FDRL and TF-DDRL require the full 100 iterations to approach convergence, while VCPN and D3QN-SEIDEL stabilize at substantially higher cost levels, highlighting the critical importance of effective knowledge sharing and action space design in heterogeneous multi-silo environments.

DeFRiS's superior performance stems from the synergistic effect of three core components. First, the action-space-agnostic policy enables parameter sharing across heterogeneous silos through candidate scoring mechanisms, allowing silos with different resource configurations to effectively transfer scheduling knowledge. Second, silo-optimized local learning combines GAE with clipped policy updates, addressing both the credit assignment problem under sparse rewards and ensuring training stability, thereby achieving rapid and smooth convergence. Finally, Non-IID robust decentralized aggregation employs a dual-track aggregation strategy that selectively transfers policy knowledge through gradient fingerprints while preserving local accuracy of value functions through gradient tracking.

Ablation Study

To validate the individual contribution of each core component in DeFRiS, we conduct systematic ablation experiments. Specifically, we construct four variants that respectively remove the action-space-agnostic policy (w/o ASAP adopts fixed-dimension output layers with masking to handle heterogeneous action spaces), GAE+CLIP mecha-

nism (w/o GAE+CLIP adopts vanilla policy gradient), gradient fingerprint-based Actor aggregation (w/o GF adopts simple parameter averaging instead), and gradient tracking-based Critic aggregation (w/o GT adopts simple parameter averaging instead). As shown in Figure 7.4, the full DeFRiS achieves a final cost of 0.185 and significantly outperforms all variants at every stage of training.

Removing the GAE+CLIP mechanism causes the most severe performance loss, with the final cost degrading to 0.252 (36.2% degradation). This variant also exhibits the slowest convergence throughout training, reaching 0.674 at iteration 10 and remaining at 0.516 at iteration 30 while full DeFRiS has descended to 0.248. This validates that both GAE's credit assignment and clipped updates' stability guarantees are indispensable in sparse reward environments, as their absence severely impairs learning efficiency and produces low-quality parameters that contaminate federated aggregation.

Removing gradient fingerprint leads to the second largest performance loss at 0.242 (30.8% degradation). Starting at 0.663 at iteration 10, this variant remains at 0.394 at iteration 30 and plateaus around 0.24 after iteration 50, reflecting the harm of blind aggregation in Non-IID environments. Simple parameter averaging forcibly fuses incompatible policy knowledge from heterogeneous silos, while gradient fingerprint's similarity-aware selective aggregation enables effective knowledge transfer matching each silo's environment.

Removing action-space-agnostic policy results in 17.3% degradation to 0.217. At iteration 10, this variant reaches 0.579, and at iteration 30 remains at 0.405 versus the full version's 0.248, demonstrating significantly slower convergence. ASAP's candidate scoring mechanism enables more efficient representation learning through its unified resource feature space, achieving faster policy optimization and more effective knowledge aggregation across heterogeneous silos throughout training.

Removing gradient tracking degrades performance to 0.205 (10.8% degradation). This variant reaches 0.555 at iteration 10 and 0.322 at iteration 30, consistently falling behind full DeFRiS throughout training. This confirms that simple Critic parameter averaging destroys accurate modeling of local state distributions, while gradient tracking successfully balances cooperative learning and local adaptation by sharing optimization momentum while maintaining personalized value functions.

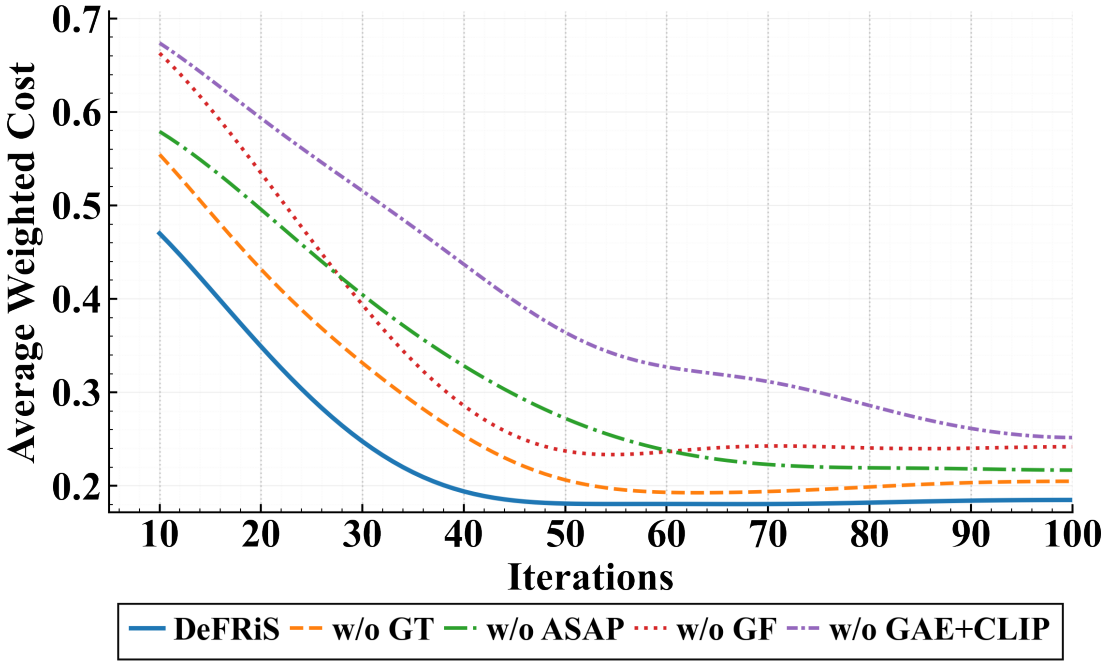
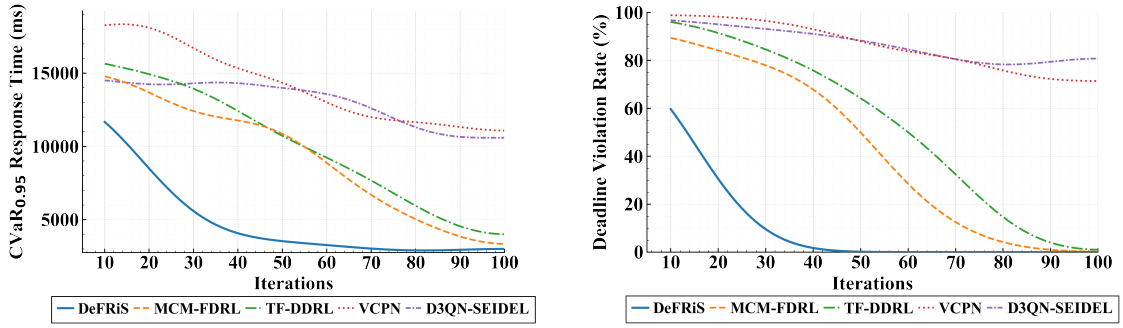


Figure 7.4: Ablation study results showing performance degradation when removing individual components of DeFRiS.

QoS Guarantee Analysis

To evaluate the performance of each approach in service quality assurance, we analyze the convergence behavior of two critical metrics. Figure 7.5a shows $\text{CVaR}_{0.95}$ response time, measuring tail latency risk at 95% confidence level, while Figure 7.5b shows deadline violation rate, measuring the proportion of tasks failing to meet latency requirements. DeFRiS achieves fast convergence and optimal performance on both metrics, validating its effectiveness in guaranteeing QoS requirements for mission-critical tasks.

Tail latency optimization. As shown in Figure 7.5a, DeFRiS's $\text{CVaR}_{0.95}$ response time rapidly decreases from around 11700 ms at iteration 10 to around 5800 ms at iteration 30 (50.4% reduction), ultimately converging to around 3000 ms. In contrast, while MCM-FDRL and TF-DDRL also achieve reductions, their convergence speeds are significantly slower than DeFRiS, finally stabilizing at around 3350 ms and 4000 ms, respectively. More notably, VCPN and D3QN-SEIDEL maintain extremely high tail latencies throughout training, ultimately reaching around 11000 ms. DeFRiS reduces tail latency by 10.4%



(a) $\text{CVaR}_{0.95}$ Response Time **(b)** Deadline Violation Rate

Figure 7.5: QoS guarantee performance comparison.

compared to the best baseline MCM-FDRL, an improvement crucial for latency-sensitive critical tasks, as tail latency often determines worst-case performance guarantees.

Deadline violation rate optimization. Figure 7.5b demonstrates even more significant performance differences. DeFRiS achieves a near 0% violation rate by iteration 50. MCM-FDRL and TF-DDRL exhibit slower improvement rates, requiring over 90 iterations to decrease to the same level. VCPN and D3QN-SEIDEL perform particularly poorly, with violation rates approaching 100% at iteration 10, and despite limited improvement throughout training, ultimately maintaining high levels of around 70% and 80%.

These results validate DeFRiS's ability to satisfy strict QoS constraints in production environments. By achieving near 0% deadline violations and significantly lower tail latencies, DeFRiS demonstrates the critical importance of effective cross-silo coordination in meeting real-time service requirements that baseline approaches consistently fail to guarantee.

Scalability Evaluation

To evaluate the scalability of DeFRiS in large-scale distributed deployments, we systematically vary the number of independent silos from 5 to 30, testing the final performance of each approach at different scales. Figure 7.6 presents the scalability performance of all approaches.

Within the range of 5 to 20 silos, DeFRiS maintains a stable average weighted cost

(approximately 0.182), 3.7% and 12.1% lower than MCM-FDRL (approximately 0.189) and TF-DDRL (approximately 0.207), respectively, and 56.7% and 59.7% lower than VCPN (approximately 0.420) and D3QN-SEIDEL (approximately 0.452), respectively. When the scale expands to 30 silos, performance divergence becomes significant: DeFRiS increases moderately to 0.194 (a 6.6% increase), while MCM-FDRL, TF-DDRL, and VCPN surge dramatically to 0.227 (20.1% increase), 0.259 (25.1% increase), and 0.468 (11.4% increase), respectively, and D3QN-SEIDEL remains at 0.458 (1.3% increase). This demonstrates that DeFRiS achieves over 3 times better performance retention compared to the best-performing baseline (MCM-FDRL) as the system scales. DeFRiS's superior scalability stems from its decentralized architecture that avoids centralized bottlenecks, gradient fingerprint and gradient tracking-based aggregation that enables robust policy transfer while preserving local adaptation, and dynamic neighbor optimization that controls communication overhead. In contrast, the centralized architectures of MCM-FDRL, TF-DDRL, and VCPN all face centralized coordination bottlenecks and Non-IID noise accumulation as scale increases, leading to sharp performance degradation. While D3QN-SEIDEL's independent learning avoids scale overhead, it cannot benefit from cooperation, maintaining a high cost level.

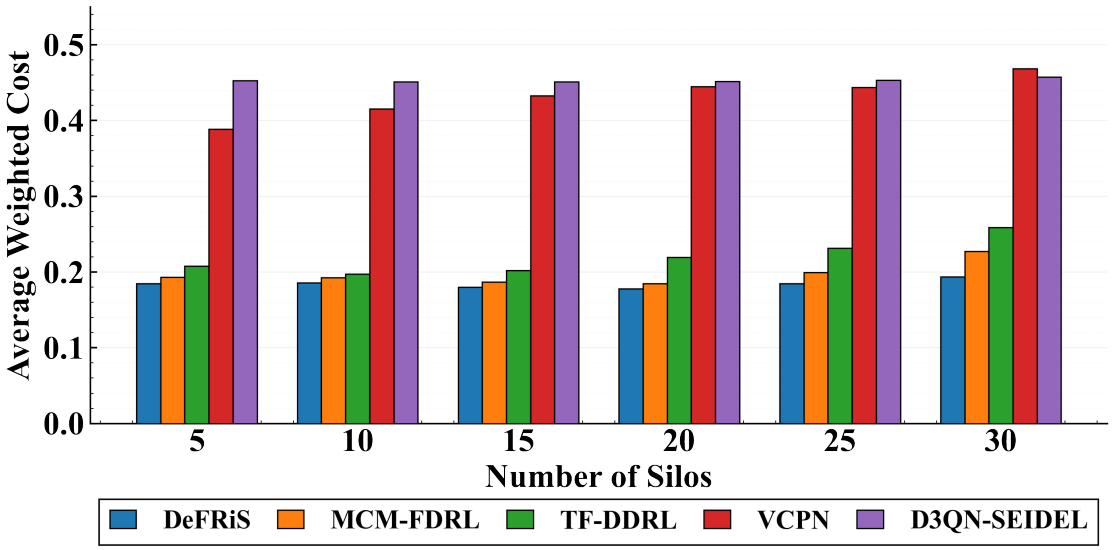


Figure 7.6: Scalability evaluation showing average weighted cost across different numbers of silos.

Robustness in Adversarial Environments

To evaluate the robustness of DeFRiS in adversarial environments, we introduce 30% malicious nodes into the 20-silo system, implementing three realistic threat scenarios: random noise injection (representing hardware failures) [326], gradient reversal attacks (representing malicious behavior) [35], and intermittent network disruptions (representing communication faults) [327]. We compare the performance of full DeFRiS, DeFRiS without anomaly detection (DeFRiS w/o Defense), and three baseline approaches involving cross-silo cooperation (MCM-FDRL, TF-DDRL, and VCPN) under attack. Since D3QN-SEIDEL trains each silo independently without cooperation mechanisms, it is immune to distributed attacks and thus excluded from this experiment. Figure 7.7 presents the average weighted cost across 100 training iterations for all approaches.

DeFRiS demonstrates superior robustness under attack. Full DeFRiS converges from approximately 0.554 to approximately 0.218, with final performance degrading only 17.8% compared to the benign scenario (0.185). In contrast, DeFRiS w/o Defense converges to approximately 0.435, suffering 135.1% performance degradation compared to the benign scenario, validating the critical role of the anomaly detection mechanism. Baseline approaches exhibit severe vulnerability: MCM-FDRL achieves approximately 0.501, representing 162.3% degradation compared to its benign performance (0.191); TF-DDRL reaches approximately 0.527, degrading 152.2% from its benign performance (0.209); while VCPN barely improves during training (from around 0.774 to around 0.757, only 2.2% reduction) with 74.8% degradation from its benign performance (0.433), nearly losing all learning capability. This highlights that DeFRiS achieves over 8 times better performance stability compared to the best-performing baseline. These results demonstrate that DeFRiS's anomaly detection mechanism effectively identifies and filters corrupted parameters through gradient fingerprint-based similarity assessment and robust statistical thresholding (median and MAD), while baseline approaches lack robustness mechanisms, leading to training collapse or severe performance degradation in adversarial environments.

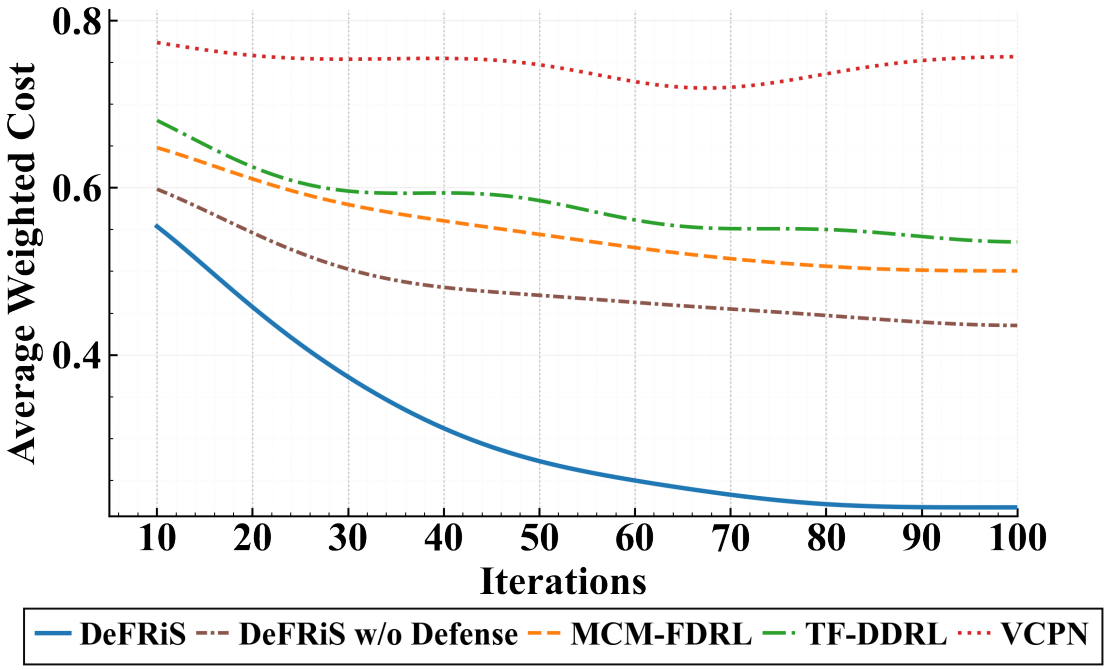


Figure 7.7: Robustness evaluation in adversarial environments.

7.6 Summary

In this chapter, we propose DeFRiS, a decentralized federated reinforcement learning framework designed to tackle the challenges of infrastructure heterogeneity, Non-IID workload shifts, and adversarial risks in silo-cooperative IoT scheduling. By integrating an action-space-agnostic policy for seamless cross-silo knowledge transfer, a GAE-enhanced local learning mechanism for stable convergence under sparse rewards, and a gradient fingerprint-based robust aggregation protocol, DeFRiS effectively synthesizes collective intelligence without a central coordinator. Extensive experiments on a real-world distributed testbed demonstrate that DeFRiS significantly outperforms state-of-the-art baselines, reducing average response time by 6.4% and energy consumption by 7.2% compared to the best-performing method while ensuring strict QoS guarantees. Notably, the framework exhibits exceptional resilience and scalability, achieving over 3 times better performance retention as the system scales and over 8 times better stability in adversarial environments.

Chapter 8

AirFed: A Federated Graph-Enhanced Multi-Agent Reinforcement Learning Framework for Multi-UAV Cooperative Mobile Edge Computing

Multiple UAV cooperative Mobile Edge Computing systems face critical challenges in coordinating trajectory planning, task offloading, and resource allocation while ensuring QoS under dynamic and uncertain environments. Existing approaches suffer from limited scalability, slow convergence, and inefficient knowledge sharing among UAVs, particularly when handling large-scale IoT deployments with stringent deadline constraints. To address these challenges, we propose AirFed, a federated graph-enhanced multi-agent reinforcement learning framework with three key innovations. First, we design dual-layer dynamic Graph Attention Networks that model spatial-temporal dependencies among UAVs and IoT devices, capturing both service relationships and collaborative interactions. Second, we develop a dual-Actor single-Critic architecture that jointly optimizes continuous trajectory control and discrete task offloading decisions. Third, we propose a reputation-based decentralized federated learning mechanism with gradient-sensitive adaptive quantization for efficient and robust knowledge sharing across heterogeneous UAVs. Extensive experiments demonstrate that AirFed achieves 42.9% reduction in weighted cost compared to state-of-the-art baselines, over 99% deadline satisfaction, 94.2% IoT device coverage, and 54.5% lower communication overhead. Scalability analysis confirms robust performance across varying UAV numbers, IoT device densities, and system scales, validating AirFed's practical applicability for large-scale UAV-MEC deployments.

This chapter is derived from:

- **Zhiyu Wang**, Suman Raj, and Rajkumar Buyya, "AirFed: Federated Graph-Enhanced Multi-Agent Reinforcement Learning for Multi-UAV Cooperative Mobile Edge Computing", *IEEE Transactions on Mobile Computing (TMC)*, 2026 [Under Review].

8.1 Introduction

The Internet-of-Things (IoT) paradigm [1] has fundamentally transformed data acquisition and decision-making across diverse spatio-temporal domains, including disaster management [328], surveillance [329], urban monitoring [330] and smart agriculture [331]. Despite this proliferation, contemporary IoT deployments deploy lightweight ground sensors and end-user devices with constrained computation, memory, and energy resources, thereby limiting their scalability to perform real-time analytics. The advent of 5G/6G communication technologies has facilitated large-scale data offloading from these devices to cloud or remote high-performance computing infrastructures for advanced processing [332]. Nevertheless, such centralized offloading becomes impractical for latency-sensitive applications due to variable end-to-end communication delays, and economically unsustainable as the monetary cost of cloud service usage escalates sharply with the increasing number of devices continuously generating data streams [54]. Furthermore, in remote or disaster-affected regions where network connectivity is intermittent or bandwidth-constrained, cloud-based processing may not even be feasible.

An emerging paradigm mitigating these challenges is Drones-as-a-Service (DaaS) [333], which extends the principles of Mobile Edge Computing (MEC) into the aerial domain. Recent advances in embedded hardware have made it feasible to equip drones, or more formally Unmanned Aerial Vehicles (UAVs), with onboard accelerators such as the NVIDIA Jetson Orin Nano, featuring 1024 CUDA cores, a 6-core Arm Cortex-A78AE CPU, and 8 GB of unified memory, all within a compact 100 79 mm form factor, a power envelope of 715 W and costing just around US$400 [334]. Within this paradigm, UAVs function as mobile sensing and compute nodes, dynamically repositioned to provide localized edge computing services to ground IoT devices. They can operate either autonomously or as part of a cooperative aerial fleet, forming a distributed edge computing layer serving ground IoT devices. This enables efficient execution of Deep Neural Network (DNN) inference tasks on-board, facilitating real-time analytics without dependence on continuous backhaul connectivity [335].

When integrated into a DaaS orchestration framework, UAVs can form a collabora-

tive service fabric in which incoming requests from ground IoT devices are dynamically mapped to aerial nodes based on proximity, residual compute capacity, and deadline constraints [336]. A UAV that receives a service request may either process it locally or offload it to another UAV within the fleet, similar to a distributed microservice invocation, thus ensuring Quality-of-Service (QoS) guarantees and efficient resource utilization.

8.1.1 Challenges

Realizing effective UAV-enabled MEC in IoT environments entails addressing several fundamental challenges arising from the distributed, dynamic, and resource-constrained nature of aerial edge computing. UAV fleets typically comprise heterogeneous platforms with varying computational capabilities, where high-end UAVs equipped with accelerators can execute inference tasks within milliseconds, while lower-end UAVs may require several seconds for the same workload [337]. This heterogeneity necessitates intelligent task scheduling that efficiently exploits resource diversity while managing the critical energy trade-offs among onboard computation, wireless communication, and flight operations under stringent battery constraints.

Further, the inherent mobility of both UAVs and ground IoT devices, coupled with stochastic task arrivals and time-varying wireless channel conditions, introduces significant uncertainty into the system. Traditional approaches that rely on accurate system models and deterministic parameters struggle to maintain performance under such dynamic conditions, as environmental changes necessitate frequent re-optimization with high computational overhead [26]. Real-time decision-making in this context requires adaptive policies capable of responding to evolving network states without complete system re-design.

Moreover, multi-UAV systems require distributed coordination for collaborative task allocation and resource sharing, yet centralized control architectures introduce single-point failure risks and communication bottlenecks for state aggregation [338]. Achieving efficient decentralized coordination across heterogeneous UAVs while avoiding resource overload remains a fundamental challenge, particularly when UAVs must cooperatively

serve overlapping areas and relay tasks among themselves.

Finally, real-world IoT applications impose stringent QoS requirements, demanding both timely task completion within application-specific deadlines and adequate coverage for distributed IoT devices [339]. Jointly optimizing task scheduling and UAV trajectory planning to satisfy these QoS requirements under dynamic conditions is non-trivial, especially when considering the cascading effects of UAV mobility on network topology and task execution latency.

8.1.2 Contributions

This chapter proposes AirFed, a federated graph-enhanced multi-agent reinforcement learning framework for multi-UAV cooperative mobile edge computing. AirFed achieves joint optimization of UAV trajectory planning and task offloading through spatial-temporal graph modeling, Constrained Multi-Agent Reinforcement Learning (CMARL), and communication efficient decentralized federated learning, while guaranteeing coverage and deadline QoS requirements. The main contributions of our chapter are as follows:

1. We establish a comprehensive system model encompassing multi-hop task offloading, UAV heterogeneity, coverage guarantees, and deadline constraints, and formulate the joint optimization as a multi-objective Mixed-integer non-linear programming (MINLP) problem.

2. We design dual-layer dynamic Graph Attention Network (GATs) with Gated Recurrent Unit (GRU) to model spatial-temporal dependencies, and develop a dual-Actor single-Critic architecture that unifies continuous trajectory control and discrete task offloading within a hybrid action space.

3. We propose reputation-based decentralized federated learning with gradient-sensitive adaptive quantization, enabling efficient knowledge sharing across UAVs while reducing communication overhead.

4. We conduct comprehensive experiments demonstrating AirFed's superior performance in convergence, QoS guarantees, and communication efficiency, with ablation and scalability analysis validating effectiveness across different system scales.

The rest of the chapter is organized as follows: Section 8.2 reviews related work and identifies research gaps; Section 8.3 establishes the system model and formulates the optimization problem; Section 8.4 presents the detailed design of the AirFed framework; Section 8.5 evaluates the performance of the framework through extensive experiments; Section 8.6 concludes the chapter.

8.2 Related Work

Recent research on UAV-assisted MEC has focused primarily on joint optimization of trajectory planning and resource allocation to improve system performance. Existing approaches can be broadly categorized into optimization-based methods and learning-based methods.

8.2.1 Optimization-based Approaches

Pervez et al. [340] proposed a Block Coordinate Descent (BCD) method combined with game theory and Successive Convex Approximation (SCA) for multi-UAV assisted MEC networks. The approach jointly optimizes task offloading decisions and UAV trajectory to minimize energy and latency-based cost function. Qi et al. [341] proposed an SCA-based joint optimization algorithm for UAV-relaying-assisted MEC networks with moving users. The approach minimizes average task completion time by optimizing communication bandwidth, CPU frequency, task division ratio, and UAV three-dimensional location deployment. Dai et al. [342] proposed a Lyapunov optimization-based method combined with Markov chain approximation for online UAV-assisted task offloading in vehicular edge computing networks. The approach minimizes vehicular task delay under long-term UAV energy constraints without requiring future information. He et al. [343] proposed an online joint optimization approach based on Lyapunov method for QoE maximization in UAV-enabled MEC. The method employs a two-stage optimization combining game theory and convex optimization to solve per-slot task offloading, resource allocation, and UAV trajectory planning problems. Xu et al. [344] proposed a surrogate Lagrangian relaxation method with hybrid numerical techniques for ser-

vice selection in MEC-based UAV last-mile delivery systems. The approach addresses heterogeneous service requirements by jointly optimizing delivery and computational service selection to minimize UAV energy consumption and service response time. Tun et al. [336] proposed a BCD approach for joint UAV deployment and resource allocation in MEC-enabled integrated space-air-ground networks. The method decomposes the problem into four subproblems solved by matching game, concave-convex procedure, SCA, and block successive upper-bound minimization approaches to minimize device and UAV energy consumption. Du et al. [345] proposed an online optimization framework based on Lyapunov optimization and surrogate Lagrangian relaxation for hierarchical collaborative MEC systems. The approach decouples decisions across time slots and employs hybrid numerical techniques for service placement, task scheduling, and resource allocation to minimize energy consumption while ensuring service placement stability. Gao et al. [346] proposed a BCD-based method with SCA technique for multi-UAV assisted MEC to minimize task completion time. The approach iteratively optimizes UAV-user association, UAV trajectory planning, and transmit power allocation to improve quality of experience.

8.2.2 Learning-based Approaches

Ning et al. [37] proposed MUTO, a Multi-Agent Deep Reinforcement Learning (MADRL) algorithm with prioritized experience replay for UAV trajectory optimization in differentiated services scenarios. The approach formulates a Markov game model and employs Actor-Critic architecture to achieve distributed trajectory control of multiple UAVs while minimizing energy consumption based on local observations. Song et al. [347] proposed a multi-objective learning approach based on Proximal Policy Optimization (PPO) combined with genetic operators for aerial-ground collaborative MEC. The method addresses latency and energy tradeoff by optimizing UAV flight paths and task offloading ratios through crossover and mutation operations at policy network parameter level. Li et al. [348] proposed an improved federated Deep Deterministic Policy Gradient (IF-DDPG) algorithm for computation offloading in multi-UAV assisted MEC. The method enhances traditional federated DDPG through dual experience replay and

mixed noise exploration to minimize task response delay and user energy consumption. Chen et al. [349] proposed the JTORA algorithm integrating Soft Actor-Critic (SAC) and Lyapunov optimization techniques for joint trajectory optimization and resource allocation in UAV-MEC systems. The approach employs Lyapunov techniques for problem transformation and combines DRL with convex optimization to minimize mobile user energy consumption.

Table 8.1: A qualitative comparison of AirFed with existing related works

Work	Problem Formulation				System Modeling			Solution Approach				Performance Evaluation		
	Optimization Objective			Decision	Multi-hop	UAV	Spatial-Temporal	Solution	Cooperation	Knowledge	Communication	QoS Guarantee		Scalability
	Time	Energy	Multi-Obj	Variables	Offloading	Heterogeneity	Modeling	Method	Mechanism	Transfer	Efficiency	Coverage	Deadline	
Pervez et al. [340]	✓	✓	✓	Joint	×	✓	None	Optimization	Centralized	None	None	×	×	✓
Qi et al. [341]	✓	×	×	Joint	×	✓	None	Optimization	Centralized	None	None	×	×	✓
Dai et al. [342]	✓	×	×	Offloading	×	✓	None	Optimization	Centralized	None	None	×	×	✓
He et al. [343]	✓	✓	✓	Joint	×	✓	None	Optimization	Centralized	None	None	×	✓	×
Xu et al. [344]	✓	✓	✓	Joint	×	✓	None	Optimization	Centralized	None	None	×	×	✓
Tun et al. [336]	×	✓	×	Joint	✓	✓	None	Optimization	Centralized	None	None	×	✓	✓
Du et al. [345]	×	✓	×	Offloading	×	✓	None	Optimization	Centralized	None	None	×	✓	✓
Gao et al. [346]	✓	×	×	Joint	×	✓	None	Optimization	Centralized	None	None	×	×	✓
Ning et al. [37]	×	✓	×	Trajectory	×	✓	None	MARL	Decentralized	Experience	None	×	×	✓
Song et al. [347]	✓	✓	✓	Joint	×	✓	None	Single DRL	None	None	None	×	×	✓
Li et al. [348]	✓	✓	✓	Joint	×	✓	None	Single DRL	Centralized	Federated	None	×	✓	✓
Chen et al. [349]	×	✓	×	Joint	×	×	None	Single DRL	None	None	None	×	×	✓
AirFed (Ours)	✓	✓	✓	Joint	✓	✓	Dual-layer GATs	MARL	Decentralized	Federated	Adaptive Quantization	✓	✓	✓

8.2.3 A Qualitative Comparison

To systematically position our work and identify research gaps, we conduct a comprehensive qualitative comparison of related works presented in Table 8.1.

Comparative Analysis Dimensions

We evaluate related works across four dimensions: problem formulation, system properties, solution approach, and performance guarantees.

Problem Formulation dimensions assess the optimization objectives and decision variables. **Optimization Objective** includes three sub-dimensions: Time (whether latency is optimized), Energy (whether energy consumption is optimized), and Multi-Objective (whether multiple objectives are jointly optimized). **Decision Variables** categorizes control variables: trajectory planning only, task offloading only, or joint optimization of both.

System Modeling dimensions characterize how the UAV-MEC network is represented. **Multi-hop Offloading** indicates whether the system supports task forwarding among UAVs. **UAV Heterogeneity** specifies whether UAVs are heterogeneous or homogeneous in terms of computational and communication capabilities. **Spatial-Temporal Modeling** indicates whether the approach explicitly models the spatio-temporal relationships in the UAV-MEC network.

Solution Approach dimensions analyze the algorithmic paradigm and technical architecture. **Solution Method** reflects the solution paradigm: traditional optimization techniques, single-agent DRL, or MADRL. **Cooperation Mechanism** describes how multiple UAVs coordinate: no cooperation, centralized coordination (requiring a central controller), or decentralized peer-to-peer collaboration. **Knowledge Transfer** evaluates how learning knowledge is shared among agents: no sharing, experience sharing, or model sharing (e.g., federated learning). **Communication Efficiency** examines whether communication overhead optimization techniques are employed.

Performance Evaluation dimensions assess the system's performance characteristics and capabilities. **QoS Guarantee** includes two sub-dimensions: **Coverage Guarantee** indicates whether IoT device coverage requirements are ensured, and **Deadline Guarantee** indicates whether task deadline constraints are ensured. **Scalability** assesses whether the approach evaluates its capability to handle system scale growth (increasing number of UAVs and devices).

Research Gap Identification

Based on the systematic analysis presented in Table 8.1, we identify five critical research gaps in existing UAV-MEC solutions:

Gap 1 - Inadequate Multi-hop Offloading Support and Spatio-Temporal Modeling. First, multi-hop task offloading remains largely unexplored, with only Tun et al. [336] addressing this scenario through traditional optimization. However, in practical deployments, direct communication between users and optimal serving UAVs may be infeasible due to coverage limitations, severe path loss, or resource contention [36]. Multi-hop relay through intermediate UAVs is essential for extending service coverage

and enabling flexible load balancing. Second, the table reveals that all existing works report *None* for spatial-temporal modeling. UAV-MEC networks inherently form dynamic graphs where nodes (UAVs) and edges (wireless links) evolve over time with complex spatial dependencies [350]. However, current approaches treat UAVs as independent entities or employ flat vector representations, failing to capture network topology and spatio-temporal evolution patterns.

Gap 2 - Dominance of Centralized Optimization with Limited Adaptability. Table 8.1 shows that 67% of existing works (8 out of 12) employ traditional optimization methods, and all optimization-based works adopt centralized cooperation mechanisms, requiring a central controller to collect global state and solve complex MINLP problems. This centralized architecture introduces single-point failure risks and communication bottlenecks for state aggregation. More fundamentally, optimization methods rely on accurate system models and deterministic parameter assumptions, while real systems face stochastic task arrivals, time-varying channel conditions, and unpredictable interference patterns [39]. When environments deviate from modeling assumptions, these methods require complete re-optimization with high computational complexity, making online adaptation infeasible and unable to cope with the dynamics and uncertainties of UAV-MEC networks.

Gap 3 - Limited Decentralized Cooperation with Efficient Knowledge Sharing. A critical gap exists in achieving decentralized cooperation with efficient knowledge transfer. Li et al. [348] employ federated learning for model-level aggregation but remain centralized in cooperation, requiring a central controller for decision-making and creating single-point failure risks. Conversely, Ning et al. [37] achieve decentralized cooperation but rely on experience sharing, which lacks the abstraction benefits of model-level knowledge and exhibits lower sample efficiency compared to federated aggregation. Table 8.1 reveals that no existing work combines decentralized cooperation with federated learning, preventing systems from simultaneously achieving distributed autonomy and efficient model-level knowledge transfer. This gap is particularly critical for UAV networks requiring both operational independence and rapid collaborative learning.

Gap 4 - Systematic Neglect of Communication Efficiency. The table shows that all existing works report *None* for communication efficiency, reflecting an implicit as-

sumption that communication costs are negligible. However, UAV-MEC networks already face significant communication demands for task offloading and result delivery. Optimization-based and learning-based methods introduce additional communication overhead for coordination, state sharing, or model synchronization among UAVs. Given that UAV wireless links typically provide limited throughput and are severely energy-constrained [36], without explicit communication-aware mechanisms, the aggregate communication burden from both system operations and algorithmic overhead may fundamentally constrain the practical viability of these solutions in resource-limited UAV deployments.

Gap 5 - Incomplete Quality-of-Service Guarantee Mechanisms. The table reveals incomplete treatment of QoS guarantees in two critical dimensions. First, no existing work considers coverage guarantee, which ensures users remain within UAV service range. Existing works implicitly assume static user-UAV associations or focus solely on optimizing performance for already-connected users, neglecting the challenge of maintaining continuous coverage in dynamic scenarios. Second, only four works (He et al. [343], Tun et al. [336], Du et al. [345], Li et al. [348]) incorporate deadline constraints to guarantee timely task completion. No work achieves joint coverage-deadline guarantees, while real-world applications often demand both service availability (coverage) and responsiveness (deadline satisfaction), limiting the applicability of existing solutions to scenarios with comprehensive QoS requirements.

To address these gaps, the proposed AirFed framework provides systematic solutions. For Gap 1, AirFed supports multi-hop task offloading and employs Dual-layer GATs to explicitly model spatial-temporal relationships in UAV-MEC networks. For Gap 2, it adopts CMARL to replace traditional optimization, enabling online adaptive decision-making under system constraints through data-driven approaches. For Gap 3, it is the first to organically combine decentralized cooperation with federated learning, enabling each UAV to make autonomous decisions while performing efficient model-level knowledge sharing. For Gap 4, it integrates a gradient-sensitive adaptive quantization mechanism to significantly reduce communication overhead, ensuring deployability in bandwidth-limited UAV-MEC networks. For Gap 5, it simultaneously provides coverage and deadline guarantees through coordinated UAV trajectory planning and

multi-hop task offloading. As shown in Table 8.1, AirFed is the only work that provides comprehensive solutions across all critical dimensions.

8.3 System Model and Problem Formulation

This section establishes the mathematical model for the multi-UAV cooperative mobile edge computing system and formulates the optimization problem. Fig. 8.1 illustrates a representative deployment where four UAVs serve 50 computation requests from multiple IoT devices across agriculture, smart city, and disaster response domains.

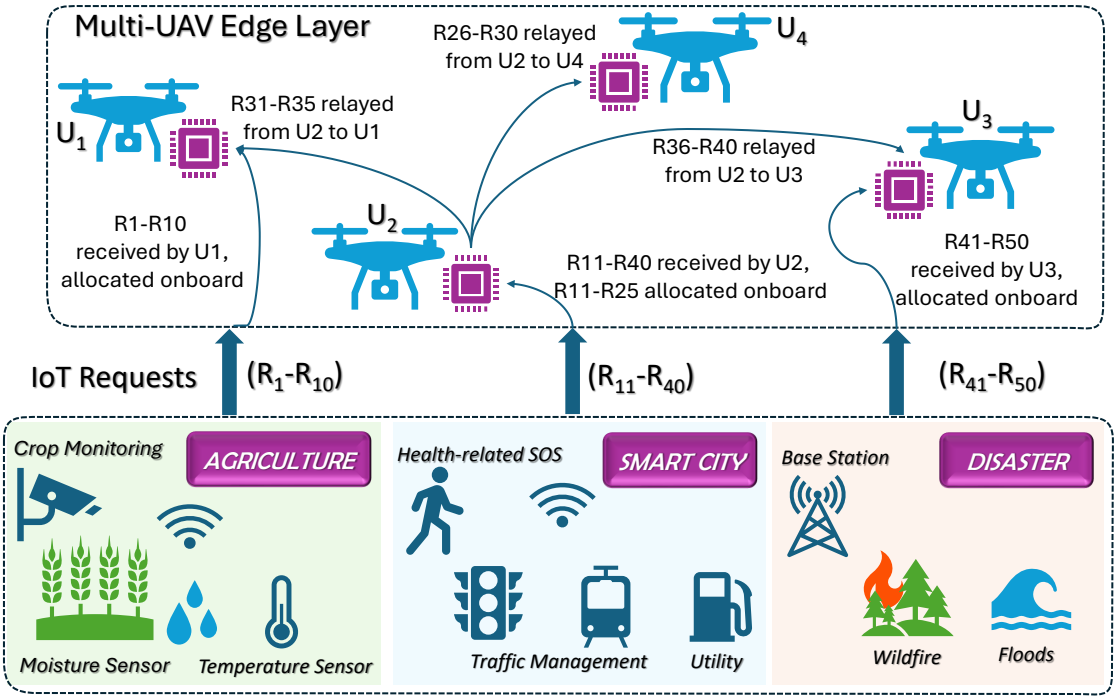


Figure 8.1: System model showing multi-UAV cooperative edge computing serving 50 requests across three application domains with heterogeneous spatial distributions.

8.3.1 Network and Spatial Model

We consider a multi-UAV cooperative system that provides mobile edge computing services to distributed IoT devices. This section establishes the mathematical model for the

network topology and spatial relationships.

The system comprises K UAVs, denoted as $\mathcal{U} = \{u_k \mid 1 \leq k \leq K\}$. The state of each UAV u_k at time t can be represented by a four-tuple:

$$s_k(t) = \langle pos_k(t), v_k(t), energy_k(t), load_k(t) \rangle, \tag{8.1}$$

where $pos_k(t) = (x_k(t), y_k(t), h_k)$ denotes the three-dimensional coordinate position, $v_k(t) = (v_{x,k}(t), v_{y,k}(t))$ is the current velocity vector, $energy_k(t)$ is the remaining battery level, and $load_k(t)$ is the current computational load.

The service area contains M IoT devices distributed across the region, denoted as $\mathcal{D} = \{d_m \mid 1 \leq m \leq M\}$. Each IoT device d_m has a fixed geographical location $loc_m = (x_m, y_m)$. The task generation rate of device d_m follows $\lambda_m(t) \sim \text{Poisson}(\mu_m)$, where μ_m is the parameter of the average task generation rate of device d_m, reflecting the random arrival characteristics of computational tasks.

The communication coverage capability of UAV u_k over IoT device d_m depends on their spatial distance. The Euclidean distance between the device and UAV is:

$$d_{m,k}(t) = \sqrt{(x_k(t) - x_m)^2 + (y_k(t) - y_m)^2 + h_k^2}. \tag{8.2}$$

Based on the free-space path loss model [351], the channel gain between the device and UAV is:

$$|h_{m,k}|^2 = \frac{G_0}{d_{m,k}(t)^2}, \tag{8.3}$$

where G_0 is the gain constant of the reference channel, reflecting the basic characteristics of the signal propagation environment. The Received Signal Strength Indicator (RSSI) at IoT device d_m from UAV u_k is:

$$RSSI_{k,m}(t) = P_k^{tx} \times |h_{m,k}|^2 = \frac{P_k^{tx} \cdot G_0}{d_{m,k}(t)^2}, \tag{8.4}$$

where P_k^{tx} is the transmission power of UAV u_k.

When the signal strength exceeds the minimum reception threshold $RSSI_{min}$, the

UAV is considered capable of providing effective service to that device. Therefore, the coverage status of UAV u_k over device d_m at time t can be expressed using an indicator function:

$$I_{k,m}^{cov}(t) = \begin{cases} 1, & \text{if } RSSI_{k,m}(t) \geq RSSI_{min} \\ 0, & \text{otherwise} \end{cases}. \tag{8.5}$$

8.3.2 Task Completion Time Model

In UAV-assisted mobile edge computing systems, task completion time is influenced by multiple factors including wireless channel conditions, UAV computational capabilities, network topology, and offloading decision strategies. This section establishes a path-based task completion time model to quantify task processing delay.

Each computational task generated by the IoT device d_m is represented by a four-tuple $\tau_l = \langle w_l, s_l, d_l, c_l \rangle$, where w_l denotes the required CPU cycles, s_l and d_l represent input and output data sizes, respectively, and c_l indicates the deadline constraint.

Considering the limited computational capabilities and simple network access mechanisms of IoT devices, devices employ a signal strength-based UAV selection strategy. The device d_m selects the UAV u_{k^*} that provides the strongest signal coverage as its service node:

$$u_{k^*} = \arg\max_{u_k \in \mathcal{U}} RSSI_{k,m}(t). \tag{8.6}$$

When a UAV receives a task request, it must decide whether to execute the task locally or forward it to other UAVs for processing. Each task follows a path model where the processing path for task τ_l is represented as an ordered sequence $\mathcal{P}_l = \{u_{k_1}, u_{k_2}, ..., u_{k_h}\}$, where u_{k_1} is the serving UAV that receives the task, u_{k_h} is the final UAV that executes the computation, and $h \geq 1$ is the path length. When $h = 1$, it represents local processing; when $h > 1$, it represents multi-hop forwarding processing. The end-to-end task completion time comprises three main stages:

$$T_l^{total} = T_l^{uplink} + T_l^{path} + T_l^{downlink}. \tag{8.7}$$

The uplink transmission time T_l^{uplink} represents the time required for the IoT device to transmit task data to the serving UAV:

$$T_l^{uplink} = \frac{s_l}{R_{m,k_1}^{up}}. \tag{8.8}$$

Based on the Shannon-Hartley theorem[351], the uplink channel capacity R_{m,k_1}^{up} is:

$$R_{m,k_1}^{up} = B \log_2 \left(1 + \frac{P_m^{tx} \cdot |h_{m,k_1}|^2}{N_0 + \sum_{m' \neq m} P_{m'}^{tx} \cdot |h_{m',k_1}|^2}\right), \tag{8.9}$$

where B is the channel bandwidth, P_m^{tx} is the IoT device transmission power, N_0 is the noise power, and the summation term $\sum_{m' \neq m} P_{m'}^{tx} \cdot |h_{m',k_1}|^2$ represents the interference power from other simultaneously transmitting devices.

The path processing time T_l^{path} can be decomposed into four distinct components:

$$T_l^{path} = T_l^{decision} + T_l^{forward} + T_l^{process} + T_l^{return}, \tag{8.10}$$

where each component represents a specific stage of the processing pipeline.

The cumulative decision time $T_l^{decision}$ accounts for the offloading decisions (local processing or forwarding) made by each UAV along the path:

$$T_l^{decision} = \sum_{i=1}^{h} T_{k_i}^{decision}. \tag{8.11}$$

The forwarding transmission time $T_l^{forward}$ represents the inter-UAV communication time for task data:

$$T_l^{forward} = \sum_{i=1}^{h-1} \frac{s_l}{R_{k_i,k_{i+1}}^{inter}}, \tag{8.12}$$

where the inter-UAV communication capacity $R_{k,k'}^{inter}$ is modeled based on the line-of-sight propagation characteristics [352] of air-to-air links:

$$R_{k,k'}^{inter} = B_{inter} \log_2 \left(1 + \frac{P_k^{tx} \cdot |h_{k,k'}|^2}{N_0}\right), \tag{8.13}$$

where B_{inter} is the inter-UAV communication bandwidth, typically much larger than the air-to-ground communication bandwidth B, and P_k^{tx} is the transmission power of UAV u_k. $|h_{k,k'}|^2$ is the inter-UAV channel gain, modeled as:

$$|h_{k,k'}|^2 = \frac{G_{inter}}{d_{k,k'}^2}, \tag{8.14}$$

where $d_{k,k'}$ is the Euclidean distance between two UAVs, and G_{inter} is the reference gain constant for inter-UAV communication.

The processing time $T_l^{process}$ at the final UAV includes both queuing delay and computation execution:

$$T_l^{process} = T_{k_h}^{queue} + \frac{w_l}{f_{k_h}}, \tag{8.15}$$

where f_{k_h} represents the computational frequency of the executing UAV.

The result return time T_l^{return} covers the transmission of computation results back along the reverse path:

$$T_l^{return} = \sum_{i=h}^{2} \frac{d_l}{R_{k_i,k_{i-1}}^{inter}}. \tag{8.16}$$

The downlink transmission time $T_l^{downlink}$ represents the time for transmitting computation results from the serving UAV back to the original IoT device:

$$T_l^{downlink} = \frac{d_l}{R_{k_1,m}^{down}}. \tag{8.17}$$

The downlink channel capacity $R_{k_1,m}^{down}$ is modeled similarly to the uplink, but UAVs typically have higher transmission power than IoT devices.

8.3.3 Energy Consumption Model

The energy consumption in UAV-assisted mobile edge computing systems primarily stems from the multi-dimensional operational overhead of UAVs. To accurately reflect the different impacts of trajectory optimization and task processing decisions, this sec-

tion establishes a UAV energy consumption model.

The total energy consumption of a UAV comprises two independent components: trajectory flight energy consumption and task processing energy consumption:

$$E_k^{total} = E_k^{trajectory} + \sum_{l \in \mathcal{L}_k} E_{k,l}^{task}, \tag{8.18}$$

where $\mathcal{L}_k$ represents the set of all tasks processed by UAV u_k.

The trajectory flight energy consumption $E_k^{trajectory}$ reflects the total energy consumption of UAVs executing mobility strategies. UAV flight energy consumption depends on the instantaneous velocity, with higher velocities resulting in increased aerodynamic drag and power requirements. The instantaneous flight power is modeled based on aerodynamic characteristics:

$$P_k^{flight}(||v_k(t)||_2) = P_k^{hover} + \frac{1}{2}\rho A C_D ||v_k(t)||_2^3, \tag{8.19}$$

where P_k^{hover} is the hovering power, ρ is the air density, A is the effective drag area, C_D is the drag coefficient, and $||v_k(t)||_2$ is the flight speed. The trajectory energy consumption is calculated by integrating power consumption over the flight duration:

$$E_k^{trajectory} = \int_0^{T_k^{flight}} P_k^{flight}(||v_k(t)||_2) dt \tag{8.20}$$

The task processing energy consumption is based on the three-stage structure:

$$E_{k,l}^{task} = E_{k,l}^{uplink} + E_{k,l}^{path} + E_{k,l}^{downlink}. \tag{8.21}$$

The uplink communication energy consumption $E_{k,l}^{uplink}$ represents the energy consumption for the serving UAV to receive task data from IoT devices:

$$E_{k,l}^{uplink} = P_k^{rx} \cdot T_l^{uplink}, \tag{8.22}$$

where P_k^{rx} is the air-to-ground communication receiving power of the UAV.

The path processing energy consumption $E_{k,l}^{path}$ can be decomposed into four compo-

nents:

$$E_{k,l}^{path} = E_{k,l}^{decision} + E_{k,l}^{forward} + E_{k,l}^{process} + E_{k,l}^{return}. \tag{8.23}$$

The decision energy consumption $E_{k,l}^{decision}$ represents the cumulative energy consumption of all UAVs along the path executing offloading decision algorithms:

$$E_{k,l}^{decision} = \sum_{i=1}^{h} P_{k_i}^{cpu} \cdot T_{k_i}^{decision}, \tag{8.24}$$

where $P_{k_i}^{cpu}$ is the CPU power consumption of UAV u_{k_i} when executing decision algorithms.

The forwarding communication energy consumption $E_{k,l}^{forward}$ represents the energy consumption for inter-UAV task data transmission, including both transmission and reception components:

$$E_{k,l}^{forward} = \sum_{i=1}^{h-1} \left(P_{k_i}^{tx} \cdot \frac{s_l}{R_{k_i,k_{i+1}}^{inter}} + P_{k_{i+1}}^{rx} \cdot \frac{s_l}{R_{k_i,k_{i+1}}^{inter}} \right), \tag{8.25}$$

where $P_{k_i}^{tx}$ is the inter-UAV communication transmission power of sending UAV u_{k_i}, and $P_{k_{i+1}}^{rx}$ is the inter-UAV communication reception power of receiving UAV $u_{k_{i+1}}$.

The processing energy consumption $E_{k,l}^{process}$ occurs at the final executing UAV, including the basic power consumption during queuing delay and actual computation energy consumption:

$$E_{k,l}^{process} = P_{k_h}^{idle} \cdot T_{k_h}^{queue} + E_{k_h,l}^{compute}. \tag{8.26}$$

The computation energy consumption $E_{k_h,l}^{compute}$ is modeled based on the dynamic power characteristics of CMOS digital circuits. According to the CMOS power formula [353], the dynamic power consumption is given by $P = \alpha C V^2 f$, where α is the activity factor, C is the load capacitance, V is the supply voltage, and f is the operating frequency. Under Dynamic Voltage and Frequency Scaling (DVFS) [354], the voltage scales approximately linearly with frequency, i.e., $V \propto f$. Therefore, the computation power consumption can

be expressed as:

$$P^{compute} = \kappa \cdot (f_{k_h})^3, \tag{8.27}$$

where κ is the effective capacitance coefficient reflecting the processor technology characteristics. For a task executing w_l CPU cycles, the required time is $\frac{w_l}{f_{k_h}}$, thus the total computation energy consumption is:

$$E^{compute}_{k_h,l} = P^{compute} \cdot \frac{w_l}{f_{k_h}} = \kappa \cdot (f_{k_h})^2 \cdot w_l. \tag{8.28}$$

The result return energy consumption $E^{return}_{k,l}$ represents the energy consumption for transmitting computation results along the reverse path, also including both transmission and reception components:

$$E^{return}_{k,l} = \sum_{i=h}^{2} \left(P^{tx}_{k_i} \cdot \frac{d_l}{R^{inter}_{k_i,k_{i-1}}} + P^{rx}_{k_{i-1}} \cdot \frac{d_l}{R^{inter}_{k_i,k_{i-1}}} \right), \tag{8.29}$$

where $P^{tx}_{k_i}$ is the transmission power of sending UAV u_{k_i} and $P^{rx}_{k_{i-1}}$ is the reception power of receiving UAV $u_{k_{i-1}}$.

The downlink communication energy consumption $E^{downlink}_{k,l}$ represents the energy consumption for the serving UAV to transmit final results to IoT devices:

$$E^{downlink}_{k,l} = P^{tx}_{k} \cdot T^{downlink}_{l}, \tag{8.30}$$

where P^{tx}_{k} is the air-to-ground communication transmission power of the UAV.

8.3.4 Problem Formulation

Based on the aforementioned models, this section formulates the joint optimization problem for UAV-assisted mobile edge computing systems. The core objective of the system is to minimize the comprehensive cost that simultaneously considers task completion time and UAV energy consumption as two key performance indicators, while satisfying QoS requirements.

Decision Variables

The optimization problem involves the following main decision variables:

UAV Velocity Decisions: $\mathbf{V} = \{v_{x,k}(t), v_{y,k}(t) \mid k \in [1, K], t \in [1, T]\}$ represents the velocity decisions of all UAVs within the time window.

Processing Path Decisions: $\mathbf{P} = \{\mathcal{P}_l \mid l \in \mathcal{L}\}$ represents the complete processing path for each task, where $\mathcal{P}_l = \{u_{k_1}, u_{k_2}, ..., u_{k_h}\}$. The first UAV u_{k_1} in the path is the serving UAV that receives the task, and the last UAV u_{k_h} is the final UAV that executes the computation. When $|\mathcal{P}_l| = 1$, it represents local processing.

Objective Function

The optimization objective of the system is to minimize the weighted combination of task completion time and UAV energy consumption:

$$\min_{\mathbf{V},\mathbf{P}} \quad F_{total} = \alpha \cdot \bar{F}_{time} + \beta \cdot \bar{F}_{energy}, \tag{8.31}$$

where α and β represent the weights for time and energy. $\bar{F}_{time}$ and $\bar{F}_{energy}$ represent the normalized time and energy costs, respectively.

To effectively compare time and energy objectives with different dimensions, we adopt a min-max normalization method. The normalized objective functions are defined as:

$$\bar{F}_{time} = \frac{F_{time} - F_{time}^{min}}{F_{time}^{max} - F_{time}^{min}}, \tag{8.32}$$

$$\bar{F}_{energy} = \frac{F_{energy} - F_{energy}^{min}}{F_{energy}^{max} - F_{energy}^{min}}, \tag{8.33}$$

where the time cost function is:

$$F_{time} = \frac{1}{|\mathcal{L}|} \sum_{l \in \mathcal{L}} T_l^{total}, \tag{8.34}$$

representing the average task completion time across all completed tasks in an episode.

The energy cost function is:

$$F_{energy} = \frac{1}{K} \sum_{k=1}^{K} E_k^{total}, \tag{8.35}$$

representing the average energy consumption per UAV in an episode.

F_{time}^{min} and F_{time}^{max} are the minimum and maximum average task completion times observed across all episodes and algorithms during training. Similarly, F_{energy}^{min} and F_{energy}^{max} are the minimum and maximum per-UAV energy consumption values. This normalization ensures both objectives are scaled to the range $[0, 1]$, enabling fair weighted combination in the optimization objective.

Constraints

UAV Velocity Constraints: UAV speed cannot exceed the maximum velocity:

$$||v_k(t)||_2 \leq v_k^{max}, \quad \forall k, t. \tag{8.36}$$

UAV Position Update: UAV position follows kinematic relationship:

$$pos_k(t+1) = pos_k(t) + v_k(t) \cdot \Delta t, \quad \forall k, t, \tag{8.37}$$

where Δt is the time step duration.

Time Discretization Constraint: The time step Δt must ensure realistic UAV acceleration limits:

$$\Delta t \geq \max_{k \in \mathcal{U}} \frac{2v_k^{max}}{a_k}, \tag{8.38}$$

where a_k is the acceleration capability of UAV u_k.

Processing Path Constraints: Each task must have one and only one processing path:

$$|\mathcal{P}_l| \geq 1, \quad \forall l \in \mathcal{L}. \tag{8.39}$$

Service Admission Constraints: The first UAV in the path must be within communica-

tion range of the source device:

$$I_{k_1,m}^{cov}(t) = 1, \quad \forall l \in \mathcal{L}_m, \mathcal{P}_l = \{u_{k_1}, ...\}, \tag{8.40}$$

where $\mathcal{L}_m$ represents the task set generated by device d_m.

UAV Energy Constraints: The total energy consumption cannot exceed the available battery energy:

$$E_k^{total} \leq energy_k(0), \quad \forall k \in \mathcal{U} \tag{8.41}$$

where $energy_k(0)$ represents the initial battery capacity of UAV u_k.

UAV Computational Capacity Constraints: The computational load must not exceed UAV capacity:

$$load_k(t) \leq load_k^{max}, \quad \forall k \in \mathcal{U}, t \tag{8.42}$$

where $load_k^{max}$ represents the maximum computational capacity of UAV u_k.

Path Connectivity Constraints: Adjacent UAVs in the task processing path must have effective communication links:

$$d_{k_i,k_{i+1}} \leq R_{comm}, \quad \forall u_{k_i}, u_{k_{i+1}} \in \mathcal{P}_l, \tag{8.43}$$

where R_{comm} is the maximum effective distance for inter-UAV communication.

Deadline Constraints: Each task should be completed before its deadline:

$$T_l^{total} \leq c_l, \quad \forall l \in \mathcal{L}. \tag{8.44}$$

Coverage Constraints: Each IoT device should be covered by at least one UAV to ensure service availability:

$$\sum_{k \in \mathcal{U}} I_{k,m}^{cov}(t) \geq 1, \quad \forall m \in \mathcal{D}, t. \tag{8.45}$$

This optimization problem is a complex Mixed-Integer Nonlinear Programming (MINLP)

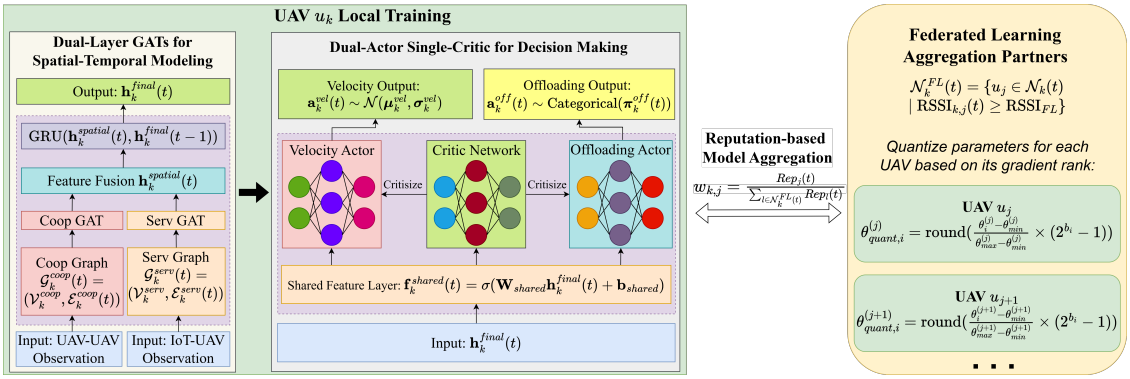


Figure 8.2: Overall architecture of AirFed framework. The framework integrates dual-layer GATs for spatial-temporal modeling, dual-Actor single-Critic for hybrid decision making, and decentralized federated learning featuring reputation-based aggregation and gradient-sensitive quantization.

problem involving joint optimization of continuous variables (UAV velocities) and discrete variables (path selection). The complexity of the problem stems from the coupling of UAV mobility, dynamic task arrivals, multi-hop cooperative processing, and other factors, requiring the design of efficient approaches to obtain near-optimal solutions.

8.4 Proposed AirFed Framework

We now present AirFed, our proposed framework for multi-UAV cooperative edge computing. As illustrated in Fig. 8.2, AirFed consists of three key components: spatial-temporal feature extraction using dual-layer GATs, hybrid decision making via dual-actor single-critic architecture, and decentralized efficient federated learning through reputation-based aggregation and adaptive quantization. Each UAV operates autonomously while collaborating with neighbors to improve collective performance. The following subsections detail each component.

8.4.1 Dynamic Graph Attention Networks for Spatial-Temporal Modeling

This section designs the dynamic GATs to model the complex spatial-temporal relationships in UAV-IoT systems. The architecture captures UAV cooperation relationships and

UAV-IoT service relationships through a dual-layer graph structure, and achieves real-time network situational awareness through distributed graph attention computation.

Dual-Layer Dynamic Graph Construction

To comprehensively model the complex interaction relationships in UAV-assisted edge computing systems, we design a dual-layer dynamic graph structure. Each UAV u_k maintains a local dual-layer graph $\mathcal{G}_k^{local}(t) = \{\mathcal{G}_k^{coop}(t), \mathcal{G}_k^{serv}(t)\}$, modeling cooperation relationships and service relationships respectively.

The UAV cooperation layer graph $\mathcal{G}_k^{coop}(t) = (\mathcal{V}_k^{coop}, \mathcal{E}_k^{coop}(t))$ models the communication and cooperation relationships between UAV u_k and its neighbors. The node set $\mathcal{V}_k^{coop} = \{u_k\} \cup \mathcal{N}_k(t)$ contains the UAV itself and its communication neighbors, with the edge set defined as $(u_k, u_{k'}) \in \mathcal{E}_k^{coop}(t)$ if and only if $d_{k,k'}(t) \leq R_{comm}$. Each UAV node feature vector is:

$$\mathbf{h}_{k'}^{coop}(t) = [pos_{k'}(t), v_{k'}(t), energy_{k'}(t), load_{k'}(t), f_{k'}], \tag{8.46}$$

containing position coordinates, flight velocity, remaining energy, current computational load, and computational capacity information. The cooperation edge feature vector is:

$$\mathbf{e}_{k,k'}^{coop}(t) = [d_{k,k'}(t), RSSI_{k,k'}(t), B_{inter}, \eta_{k,k'}], \tag{8.47}$$

where $d_{k,k'}(t)$ represents the Euclidean distance between UAVs; $RSSI_{k,k'}(t)$ is the communication signal strength; B_{inter} is the available bandwidth; $\eta_{k,k'}$ is the historical cooperation frequency.

The UAV-IoT service layer graph $\mathcal{G}_k^{serv}(t) = (\mathcal{V}_k^{serv}, \mathcal{E}_k^{serv}(t))$ models the coverage and service relationships between UAV u_k and IoT devices. The node set $\mathcal{V}_k^{serv} = \{u_k\} \cup \mathcal{D}_k^{cov}(t)$ contains the UAV itself and its covered IoT devices. The UAV node uses the same feature representation as in the UAV cooperation layer, and the IoT device node features are:

$$\mathbf{h}_m^{serv}(t) = [loc_m, |\mathcal{Q}_m(t)|, \lambda_m, \bar{c}_m(t)], \tag{8.48}$$

where loc_m is the device location coordinates; $|\mathcal{Q}_m(t)|$ is the task queue length; λ_m is the task generation rate; $\bar{c}_m(t)$ is the deadline of the most urgent task. Service edge features are:

$$\mathbf{e}_{k,m}^{serv}(t) = [d_{k,m}(t), RSSI_{k,m}(t), R_{k,m}^{up}(t)], \tag{8.49}$$

where $d_{k,m}(t)$ is the distance between UAV and IoT device; $RSSI_{k,m}(t)$ is the signal strength; $R_{k,m}^{up}(t)$ is the uplink capacity.

Considering UAV mobility and task dynamics, each UAV's local graph structure requires real-time updates:

$$\mathcal{G}_k^{local}(t + \Delta t) = \text{Update}(\mathcal{G}_k^{local}(t)). \tag{8.50}$$

To balance real-time performance and computational overhead, we adopt an adaptive update strategy based on mobility speed, where each UAV determines its local graph update interval:

$$\Delta t_k^{update} = \frac{\Delta t_{base}}{1 + \alpha_{speed} \cdot ||v_k(t)||_2}, \tag{8.51}$$

where α_{speed} is the speed sensitivity parameter, and UAVs with higher mobility speeds have higher graph structure update frequencies.

Spatial-Temporal Graph Attention Network

We design multi-scale spatial-temporal GATs where each UAV performs distributed graph attention computation based on its local graph. The dual-layer GATs adopt inter-layer interaction:

$$\begin{aligned}\mathbf{H}_k^{coop,(l+1)}(t) = \sigma\Big(&\mathbf{A}_{att}^{coop}(t)\mathbf{H}_k^{coop,(l)}(t)\mathbf{W}^{coop,(l)} \\ &+\mathbf{M}_k^{serv\to coop}(t)\mathbf{H}_k^{serv,(l)}(t)\mathbf{W}^{cross,(l)}\Big),\end{aligned} \tag{8.52}$$

$$\mathbf{H}_k^{serv,(l+1)}(t) = \sigma\left(\mathbf{A}_{att}^{serv}(t)\mathbf{H}_k^{serv,(l)}(t)\mathbf{W}^{serv,(l)}\right), \tag{8.53}$$

where $\mathbf{A}_{att}$ is the attention-based dynamic adjacency matrix, and the information transfer matrix $\mathbf{M}_k^{serv\to coop}(t)$ transfers task urgency information from the service layer to the cooperation layer:

$$\mathbf{M}_{k,m}^{serv\to coop}(t) = \frac{I_{k,m}^{cov}(t)\cdot w_{k,m}(t)}{\sum_{m'\in\mathcal{D}_k^{cov}(t)} I_{k,m'}^{cov}(t)\cdot w_{k,m'}(t)}, \tag{8.54}$$

where the weight function $w_{k,m}(t) = \exp(-\gamma_{urg}\cdot\bar{c}_m(t))$ is determined based on task urgency.

To adaptively learn the cooperation relationships between UAVs and the service relationships between UAVs and IoT devices, we design a multi-head attention mechanism. The cooperation layer attention weights are computed as:

$$Attention_{k,k'}^{coop,(h)}(t) = \frac{\exp(\mathbf{q}_k^{coop,(h)T}\mathbf{k}_{k'}^{coop,(h)}/\sqrt{d_k})}{\sum_{k''\in\mathcal{N}_k(t)}\exp(\mathbf{q}_k^{coop,(h)T}\mathbf{k}_{k''}^{coop,(h)}/\sqrt{d_k})}. \tag{8.55}$$

The service layer adopts a similar attention computation. Through this design, UAVs can dynamically adjust their attention to different neighboring UAVs and IoT devices based on multi-dimensional information including geographical distance, load status, communication quality, and task urgency, achieving adaptive cooperation decisions.

To capture the temporal evolution patterns of the system, we model temporal dependencies through Gated Recurrent Unit (GRU) networks:

$$\mathbf{h}_k^{final}(t) = \text{GRU}(\mathbf{h}_k^{spatial}(t), \mathbf{h}_k^{final}(t-1)), \tag{8.56}$$

where $\mathbf{h}_k^{spatial}(t)$ is the spatial feature representation fusing information from $\mathbf{H}_k^{coop}$ and $\mathbf{H}_k^{serv}$. The GRU is chosen for its computational efficiency while maintaining effective temporal modeling capabilities, which is crucial for real-time UAV edge computing applications. This dynamic GATs provides each UAV with a comprehensive state representation $\mathbf{h}_k^{final}(t)$ containing local status, neighborhood context, task urgency, and co-

operation opportunities, serving as high-quality decision input for subsequent CMARL.

8.4.2 Constrained Multi-Agent Reinforcement Learning

This section reformulates the original optimization problem as a Constrained Multi-Agent Reinforcement Learning (CMARL) problem and designs a policy network architecture for handling hybrid action spaces. Each UAV acts as an independent agent, making joint decisions on trajectory planning and task offloading based on the state representation $\mathbf{h}_k^{final}(t)$ provided by GATs.

Problem Reformulation as CMAMDP

We reformulate the MINLP problem from Section 8.3 as a Constrained Multi-Agent Markov Decision Process (CMAMDP) $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \mathcal{C} \rangle$. The system state space $\mathcal{S}$ contains comprehensive state representations of all UAVs provided by GATs:

$$\mathcal{S} = \{\mathbf{h}_k^{final}(t) \mid k = 1, 2, ..., K\}. \tag{8.57}$$

Each UAV agent faces a hybrid action space $\mathcal{A}_k = \mathcal{A}_k^{vel} \times \mathcal{A}_k^{off}$, where continuous velocity actions $\mathbf{a}_k^{vel} \in \mathcal{A}_k^{vel} \subset \mathbb{R}^2$ represent UAV movement decisions executed at each time step, and discrete task actions $\mathbf{a}_k^{off} \in \mathcal{A}_k^{off}$ represent task offloading strategy executed upon task arrivals. The state transition probability $\mathcal{P}$ is determined by the joint actions executed by UAVs and environmental dynamics, and $\mathcal{R}$ is the reward function. According to physical feasibility and service requirements, the constraint set $\mathcal{C}$ is divided into hard constraints and soft constraints:

- **Hard constraints** (must be strictly satisfied):

$$\mathcal{C}_{hard} = \{\text{Eq. (8.36)}, \text{Eq. (8.37)}, \text{Eq. (8.38)}, \text{Eq. (8.39)}, \tag{8.58}$$

$$\text{Eq. (8.40)}, \text{Eq. (8.41)}, \text{Eq. (8.42)}, \text{Eq. (8.43)}\}, \tag{8.59}$$

  strictly enforced during system operation.

- **Soft constraints** (violations allowed but penalized):

$$\mathcal{C}_{soft} = \{\text{Eq. (8.44)}, \text{Eq. (8.45)}\}, \tag{8.60}$$

representing QoS requirements, handled through penalty and reward terms in the reward function.

Dual-Actor Single-Critic Architecture

To simultaneously handle continuous trajectory optimization and discrete task offloading decisions, we design a dual-Actor single-Critic policy network architecture. This architecture unifies the processing of hybrid action spaces through shared feature extraction layers, ensuring coordination between the two types of decisions.

The shared feature layer converts GATs output state representations into decision features:

$$\mathbf{f}_k^{shared}(t) = \sigma(\mathbf{W}_{shared}\mathbf{h}_k^{final}(t) + \mathbf{b}_{shared}). \tag{8.61}$$

The architecture employs two specialized Actor networks operating on the shared feature representation:

- **Velocity Actor** outputs Gaussian distribution parameters for continuous velocity decisions, enabling smooth exploration-exploitation tradeoff through adjustable variance:

$$\boldsymbol{\mu}_k^{vel}(t) = \pi_{vel}(\mathbf{f}_k^{shared}(t); \theta_{vel}), \tag{8.62}$$

$$\boldsymbol{\sigma}_k^{vel}(t) = \sigma_{vel}(\mathbf{f}_k^{shared}(t); \theta_{vel}), \tag{8.63}$$

where velocity actions are sampled as $\mathbf{a}_k^{vel}(t) \sim \mathcal{N}(\boldsymbol{\mu}_k^{vel}, \boldsymbol{\sigma}_k^{vel})$ and constrained by velocity limits (Eq. (8.36)) to ensure feasible UAV movement.

- **Offloading Actor** outputs probability distributions for discrete processing choices:

$$\boldsymbol{\pi}_k^{off}(t) = \text{softmax}(\pi_{off}(\mathbf{f}_k^{shared}(t); \theta_{off})), \tag{8.64}$$

$$\mathbf{a}_k^{off}(t) \sim \text{Categorical}(\boldsymbol{\pi}_k^{off}(t)), \tag{8.65}$$

  allowing flexible selection between local computation and forwarding to neighboring UAVs based on current system states. Task offloading actions are executed only when new computational tasks arrive.

Both Actors share a unified Critic network for value estimation:

$$V_k(t) = V_{critic}(\mathbf{f}_k^{shared}(t); \theta_{critic}), \tag{8.66}$$

providing a common baseline for policy gradient updates across both decision types.

QoS-Aware Reward Design

We design a multi-dimensional reward function that integrates the optimization objective with QoS guarantees:

$$R_k(t) = R_k^{performance}(t) + R_k^{QoS}(t), \tag{8.67}$$

where $R_k^{QoS}(t) = R_k^{deadline}(t) + R_k^{coverage}(t)$ encompasses two key QoS aspects.

The performance reward corresponds to the negative incremental contribution to the optimization objective (Eq. (8.31)):

$$R_k^{performance}(t) = -\Delta F_{total,k}(t). \tag{8.68}$$

The deadline reward imposes penalties on tasks that violate timing constraints:

$$R_k^{deadline}(t) = -\lambda \max(0, T_l^{total} - c_l), \tag{8.69}$$

where λ is the penalty weight for deadline violations.

The coverage reward incentivizes UAVs to maintain service availability for IoT de-

vices:

$$R_k^{coverage}(t) = \eta \sum_{m \in \mathcal{D}} I_{k,m}^{cov}(t), \tag{8.70}$$

where η is the coverage reward weight, ensuring adequate service coverage across the deployment area.

Distributed Policy Optimization

Each UAV independently maintains a complete dual-Actor single-Critic architecture and conducts distributed training. Both Actor networks are updated using policy gradient methods:

$$\nabla_{\theta_{vel}} J_{vel} = \mathbb{E}[\nabla_{\theta_{vel}} \log \pi_{\theta_{vel}}(\mathbf{a}_k^{vel} | \mathbf{s}_k) A_k(t)], \tag{8.71}$$

$$\nabla_{\theta_{off}} J_{off} = \mathbb{E}[\nabla_{\theta_{off}} \log \pi_{\theta_{off}}(\mathbf{a}_k^{off} | \mathbf{s}_k) A_k(t)], \tag{8.72}$$

where the advantage function $A_k(t) = R_k(t) + \gamma V_k(t+1) - V_k(t)$ provides unified training signals for both Actors.

The Critic network is updated through temporal difference error:

$$L_{critic} = \mathbb{E}[(R_k(t) + \gamma V_k(t+1) - V_k(t))^2]. \tag{8.73}$$

The total loss function integrates policy losses from both Actors and the value loss from the Critic:

$$L_{total} = L_{vel} + L_{off} + L_{critic} + \beta_{entropy} H(\pi), \tag{8.74}$$

where $H(\pi)$ is the entropy regularization term that promotes sufficient policy exploration.

Algorithm 8.1 summarizes the overall CMARL training process for UAV cooperative edge computing. The algorithm operates in four main phases: Lines 1-5 perform system

initialization including network parameters and local graph structures. During online execution (Lines 9-27), each UAV adaptively updates its local dual-layer graph based on mobility speed (Lines 10-13), processes spatial-temporal features through GATs and GRU (Lines 15-17), and generates hybrid actions via the dual-actor architecture (Lines 20-27), with velocity decisions made continuously and task offloading triggered by task arrivals. The environment interaction phase (Lines 29-30) executes joint UAV velocity actions and task offloading decisions when applicable. The experience collection (Lines 31-38) computes multi-dimensional rewards including performance, constraint, and coverage components. Finally, distributed policy updates (Lines 40-55) enable each UAV to independently update its dual-actor and critic networks using policy gradients and temporal difference learning.

8.4.3 Multi-UAV Decentralized Federated Learning

While the CMARL enables each UAV to learn policies independently, the lack of knowledge sharing mechanisms among UAVs limits the overall learning efficiency of the system. We adopt a decentralized federated learning approach where each UAV serves both as a DRL agent and a federated learning participant. The federated learning parameter set for UAV u_k is:

$$\Theta_k^{FL} = \{\theta_{vel}^{(k)}, \theta_{off}^{(k)}, \theta_{critic}^{(k)}\}, \tag{8.75}$$

where $\theta_{vel}^{(k)}$ represents the velocity decision network parameters, $\theta_{off}^{(k)}$ denotes the task offloading decision network parameters, $\theta_{critic}^{(k)}$ indicates the value estimation network parameters. The federated learning process leverages the UAV cooperation layer graph $\mathcal{G}_k^{coop}(t)$ to establish communication topology for distributed parameter exchange, enabling collaborative learning without centralized coordination.

Algorithm 8.1: Constrained Multi-Agent Reinforcement Learning for UAV Cooperative Edge Computing

Input: UAV set $\mathcal{U}$, IoT device set $\mathcal{D}$, communication range R_{comm}, Network parameters $\theta_{vel}, \theta_{off}, \theta_{critic}$, learning rates $\alpha_{vel}, \alpha_{off}, \alpha_{critic}$

/* Initialization */
Initialize GATs parameters and network weights
Initialize UAV positions $pos_k(0)$ and states $s_k(0)$ for all $k \in \mathcal{U}$
Initialize experience buffer $\mathcal{B}_k$ for each UAV k
Initialize local graphs $\mathcal{G}_k^{local}(0)$ for all UAVs
for *training iteration* $e = 1$ *to* E **do**
  for *time step* $t = 1$ *to* T **do**
    for *each UAV* $k \in \mathcal{U}$ **do**
      /* Adaptive Graph Update */
      Compute update interval:
      $\Delta t_k^{update} = \frac{\Delta t_{base}}{1+\alpha_{speed}\cdot||v_k(t)||_2}$
      if $t \mod \Delta t_k^{update} = 0$ **then**
        Update local dual-layer graph:
        $\mathcal{G}_k^{local}(t) = \{\mathcal{G}_k^{coop}(t), \mathcal{G}_k^{serv}(t)\}$
      end if
      /* Feature Processing */
      Apply GATs on $\mathcal{G}_k^{local}(t)$ to obtain $\mathbf{h}_k^{spatial}(t)$
      Update temporal state:
      $\mathbf{h}_k^{final}(t) = \text{GRU}(\mathbf{h}_k^{spatial}(t), \mathbf{h}_k^{final}(t-1))$
      Compute shared decision features:
      $\mathbf{f}_k^{shared}(t) = \sigma(\mathbf{W}_{shared}\mathbf{h}_k^{final}(t) + \mathbf{b}_{shared})$
      /* Dual-Actor Action Selection */
      /* Velocity Actor */
      $\boldsymbol{\mu}_k^{vel}(t) = \pi_{vel}(\mathbf{f}_k^{shared}(t); \theta_{vel})$
      $\boldsymbol{\sigma}_k^{vel}(t) = \sigma_{vel}(\mathbf{f}_k^{shared}(t); \theta_{vel})$
      Sample $\mathbf{a}_k^{vel}(t) \sim \mathcal{N}(\boldsymbol{\mu}_k^{vel}, \boldsymbol{\sigma}_k^{vel})$
      /* Offloading Actor */
      $\boldsymbol{\pi}_k^{off}(t) = \text{softmax}(\pi_{off}(\mathbf{f}_k^{shared}(t); \theta_{off}))$
      Sample $\mathbf{a}_k^{off}(t) \sim \text{Categorical}(\boldsymbol{\pi}_k^{off}(t))$
      /* Critic Value Estimation */
      $V_k(t) = V_{critic}(\mathbf{f}_k^{shared}(t); \theta_{critic})$
    end for
    /* Environment Interaction */
    Execute joint actions $\{\mathbf{a}_k^{vel}(t), \mathbf{a}_k^{off}(t)\}_{k\in\mathcal{U}}$
    /* Experience Collection */
    for *each UAV* $k \in \mathcal{U}$ **do**
      Compute performance reward:
      $R_k^{performance}(t) = -\Delta F_{total,k}(t)$
      Compute constraint penalty:
      $R_k^{constraint}(t) = -\lambda \max(0, T_l^{total} - c_l)$
      Compute coverage reward:
      $R_k^{coverage}(t) = \eta \sum_{m\in\mathcal{D}} I_{k,m}^{cov}(t)$
      Total reward:
      $R_k(t) = R_k^{performance}(t) + R_k^{constraint}(t) + R_k^{coverage}(t)$
      Store transition $(\mathbf{h}_k^{final}(t), \mathbf{a}_k^{vel}(t), \mathbf{a}_k^{off}(t), R_k(t), \mathbf{h}_k^{final}(t+1))$ in $\mathcal{B}_k$
    end for
  end for
  /* Distributed Policy Updates */
  for *each UAV* $k \in \mathcal{U}$ **do**
    Sample mini-batch from experience buffer $\mathcal{B}_k$
    /* Advantage Computation */
    for *each transition in mini-batch* **do**
      Compute advantage:
      $A_k(t) = R_k(t) + \gamma V_k(t+1) - V_k(t)$
    end for
    /* Actor Networks Update */
    Compute velocity policy gradient:
    $\nabla_{\theta_{vel}} J_{vel} = \mathbb{E}[\nabla_{\theta_{vel}} \log \pi_{\theta_{vel}}(\mathbf{a}_k^{vel}|\mathbf{s}_k) A_k(t)]$
    Update:
    $\theta_{vel} \leftarrow \theta_{vel} + \alpha_{vel}\nabla_{\theta_{vel}} J_{vel}$
    Compute task policy gradient:
    $\nabla_{\theta_{off}} J_{off} = \mathbb{E}[\nabla_{\theta_{off}} \log \pi_{\theta_{off}}(\mathbf{a}_k^{off}|\mathbf{s}_k) A_k(t)]$
    Update:
    $\theta_{off} \leftarrow \theta_{off} + \alpha_{off}\nabla_{\theta_{off}} J_{off}$
    /* Critic Network Update */
    Compute value loss:
    $L_{critic} = \mathbb{E}[(R_k(t) + \gamma V_k(t+1) - V_k(t))^2]$
    Update:
    $\theta_{critic} \leftarrow \theta_{critic} - \alpha_{critic}\nabla_{\theta_{critic}} L_{critic}$
  end for
end for
return Trained policy networks π_{vel}, π_{off}, and value network V_{critic}

Reputation-based Model Aggregation

UAV u_k selects aggregation partners from its communication neighbors:

$$\mathcal{N}_k^{FL}(t) = \{u_j \in \mathcal{N}_k(t) \mid \text{RSSI}_{k,j}(t) \geq \text{RSSI}_{FL}\}, \tag{8.76}$$

where RSSI_{FL} is the minimum signal strength threshold required for federated learning.

To evaluate the trustworthiness and contribution value of different UAVs, we design a reputation mechanism based on historical performance. Considering the characteristics of UAV systems, reputation assessment needs to reflect two key aspects: the reliability of UAVs in executing computational tasks and the stability of UAVs in participating in federated learning. The former ensures learning from high-quality data and experiences, while the latter guarantees the continuity and effectiveness of parameter exchange.

The task execution reputation of UAV j is defined as:

$$Succ_j(t) = \frac{\text{successfully completed tasks}}{\text{total assigned tasks}}, \tag{8.77}$$

where the numerator represents the number of tasks successfully completed by UAV j within deadlines, and the denominator represents the total number of tasks assigned to UAV j. This metric reflects the computational capability and task processing reliability of the UAV.

The communication reputation of UAV j is defined as:

$$Stab_j(t) = \frac{\text{successful FL communications}}{\text{total FL attempts}}, \tag{8.78}$$

where the numerator represents the number of successful federated learning parameter exchanges completed by UAV j, and the denominator represents the total number of federated learning communication attempts. This metric reflects the network stability and collaborative reliability of the UAV.

At each time step, we first calculate the instantaneous reputation based on current

performance metrics:

$$\tilde{Rep}_j(t) = \alpha_{succ} \cdot Succ_j(t) + \alpha_{stab} \cdot Stab_j(t), \tag{8.79}$$

where $\alpha_{succ} + \alpha_{stab} = 1$ are weight parameters balancing task execution and communication reliability.

To prevent abrupt reputation changes caused by temporary performance fluctuations, we apply exponential moving average to smooth the reputation updates:

$$Rep_j(t) = \rho \cdot Rep_j(t-1) + (1-\rho) \cdot \tilde{Rep}_j(t), \tag{8.80}$$

where $\rho \in [0,1]$ is the forgetting factor controlling the influence of historical reputation versus current performance. And the reputation-based aggregation weights are defined as:

$$w_{k,j} = \frac{Rep_j(t)}{\sum_{l \in \mathcal{N}_k^{FL}(t)} Rep_l(t)}. \tag{8.81}$$

Communication-Efficient Adaptive Updates

Due to the typically much smaller bandwidth of inter-UAV communication compared to terrestrial networks, directly transmitting full-precision model parameters would incur enormous communication overhead. To reduce communication burden while maintaining learning effectiveness, we propose a gradient-sensitive adaptive quantization mechanism. According to the first-order Taylor expansion of the loss function, the gradient magnitude $|\partial \mathcal{L}/\partial \theta_i|$ directly reflects the impact of quantization error in parameter value on the loss function [355]. Parameters with large gradient magnitudes are more sensitive to quantization errors and require higher transmission precision, while parameters with small gradient magnitudes can tolerate more aggressive quantization compression [355]. Based on this observation, we apply high-bit quantization to parameters with large gradient magnitudes and low-bit quantization to parameters with small gradient magnitudes.

Specifically, when UAV u_j prepares to transmit parameters, it computes the gradi-

ent rank $r_i = \text{rank}(g_i)$ for each parameter based on the gradient absolute value $g_i = |\partial\mathcal{L}/\partial\theta_i|$, where the rank function assigns values in descending order. Based on the gradient rank, the quantization bit width for each parameter is determined as:

$$b_i = b_{min} + \left\lfloor \left(1 - \frac{r_i - 1}{n_p - 1}\right) \times (b_{max} - b_{min}) \right\rfloor, \tag{8.82}$$

where b_{min} and b_{max} are the minimum and maximum bit widths respectively, defining the range of quantization precision, and n_p is the total number of network parameters. This mapping ensures that parameters with higher gradient ranks receive high-precision quantization close to b_{max}, while parameters with lower gradient ranks receive low-precision quantization close to b_{min}. The parameter quantization process is:

$$\theta_{quant,i} = \text{round}\left(\frac{\theta_i - \theta_{min}}{\theta_{max} - \theta_{min}} \times (2^{b_i} - 1)\right), \tag{8.83}$$

where θ_{max} and θ_{min} are the extreme values of the parameter vector. During transmission, the bit width identifiers $\{b_i\}$ of each parameter, along with θ_{max} and θ_{min}, are attached to the quantized parameters. Upon reception, the receiving UAV reconstructs the full-precision parameters through dequantization:

$$\theta_i = \theta_{min} + \frac{\theta_{quant,i}}{2^{b_i} - 1} \times (\theta_{max} - \theta_{min}), \tag{8.84}$$

which linearly maps the quantized integer values back to the original floating-point space. This mechanism allocates transmission precision according to the relative sensitivity of parameters, achieving a balance between parameter transmission efficiency and model convergence quality in resource-constrained UAV networks.

Considering the high mobility and dynamic network topology of UAVs, strict synchronous federated learning is difficult to achieve. We design an asynchronous update mechanism that allows different UAVs to perform model aggregation at different frequencies. The aggregation frequency of each UAV k is adaptively adjusted according to its movement speed:

$$f_{agg}^{(k)} = f_{base} \times (1 + \alpha_{mobility} \times ||v_k(t)||_2), \tag{8.85}$$

where f_{base} is the baseline aggregation frequency, and $\alpha_{mobility}$ is the mobility sensitivity parameter. The rationale behind this design is that fast-moving UAVs experience more frequent changes in network neighbors and require more timely model updates to adapt to new collaborative environments, while relatively stationary UAVs can adopt lower aggregation frequencies to reduce communication overhead.

When UAVs move beyond communication range or encounter signal interference, the federated learning process needs to handle communication failures. We adopt a best-effort aggregation strategy: UAV k collects neighbor parameters within a predetermined time window and performs aggregation based on received parameters after timeout. Each UAV includes its own locally trained parameters in the aggregation process, with all participating UAVs (including itself) weighted by their respective reputations. Aggregation weights are normalized according to actually participating neighbors and the UAV itself:

$$w_{k,j}^{normalized} = \frac{Rep_j(t)}{\sum_{l \in \{k\} \cup \mathcal{N}_k^{received}(t)} Rep_l(t)}, \tag{8.86}$$

where $\mathcal{N}_k^{received}(t) \subseteq \mathcal{N}_k^{FL}(t)$ is the set of neighbors from which parameters were successfully received. This fault-tolerance mechanism ensures that the federated learning process can continue under partial communication failures while preserving local training outcomes, improving system robustness. The parameter aggregation update rules are:

$$\theta_{vel}^{(k)} \leftarrow \sum_{j \in \{k\} \cup \mathcal{N}_k^{received}(t)} w_{k,j}^{normalized} \theta_{vel}^{(j)}, \tag{8.87}$$

$$\theta_{off}^{(k)} \leftarrow \sum_{j \in \{k\} \cup \mathcal{N}_k^{received}(t)} w_{k,j}^{normalized} \theta_{off}^{(j)}, \tag{8.88}$$

$$\theta_{critic}^{(k)} \leftarrow \sum_{j \in \{k\} \cup \mathcal{N}_k^{received}(t)} w_{k,j}^{normalized} \theta_{critic}^{(j)}. \tag{8.89}$$

Algorithm 8.2 presents the decentralized federated learning for UAV cooperative edge computing. The algorithm operates through three concurrent processes that run independently for each UAV. The local CMARL training process (Lines 4-6) continuously executes DRL and updates local parameters. The reputation management (Lines

7-10) operates in an event-driven manner, updating task execution and communication reliability statistics, computing instantaneous reputation values, and applying exponential moving average smoothing to maintain stable reputation assessments. The core federated learning process (Lines 11-30) implements adaptive aggregation scheduling based on UAV mobility, where fast-moving UAVs aggregate more frequently to adapt to changing network topologies (Line 12). During each aggregation cycle, UAVs select qualified neighbors based on signal strength thresholds (Line 14), apply gradient-sensitive adaptive quantization to compress parameters based on gradient sensitivity (Lines 16-20), and send quantized parameters along with bit width identifiers and normalization bounds (Line 21). Upon reception, UAVs reconstruct full-precision parameters through dequantization (Lines 23-25) and perform reputation-weighted aggregation that includes both locally trained parameters and received neighbor parameters, with weights normalized across all participants (Lines 26-29). This decentralized approach enables knowledge sharing among UAVs while maintaining communication efficiency and robustness against communication failures and dynamic network conditions.

8.5 Performance Evaluation

We evaluate AirFed through extensive experiments, analyzing convergence behavior, QoS guarantees, component contributions, and scalability across different system configurations.

8.5.1 Experimental Setup

We describe the evaluation environment, baseline approaches, and hyperparameter configurations.

Simulation Environment and Parameters

We implement AirFed using ReinFog [31], a modular DRL framework with native support for distributed heterogeneous agent deployment and flexible communication mechanisms, making it well-suited for our decentralized multi-UAV system. We leverage Re-

Algorithm 8.2: Decentralized Federated Learning for UAV Cooperative Edge Computing

Input: UAV set $\mathcal{U}$, reputation parameters α_{succ}, α_{stab}, ρ, FL parameters $RSSI_{FL}$, b_{min}, b_{max}, f_{base}, $\alpha_{mobility}$, $T_{timeout}$

`/* Note: CMARL training, reputation updates, and FL aggregation run concurrently and independently */`

for *each UAV* $k \in \mathcal{U}$ **do in parallel**

  while *system operating* **do**

    `/* Local CMARL Training (Continuous) */`

    Execute local CMARL training (Algorithm 8.1)

    Update local parameters:

    $\{\boldsymbol{\theta}_{vel}^{(k)}, \boldsymbol{\theta}_{off}^{(k)}, \boldsymbol{\theta}_{critic}^{(k)}\}$

    `/* Reputation Management (Event-driven) */`

    Update performance statistics $Succ_k(t)$ and $Stab_k(t)$ based on task and communication events

    Compute instantaneous reputation:

    $\tilde{Rep}_k(t) = \alpha_{succ} \cdot Succ_k(t) + \alpha_{stab} \cdot Stab_k(t)$

    Update smoothed reputation:

    $Rep_k(t) = \rho \cdot Rep_k(t-1) + (1-\rho) \cdot \tilde{Rep}_k(t)$

    `/* Adaptive FL Aggregation (Periodic) */`

    Compute aggregation frequency:

    $f_{agg}^{(k)} = f_{base} \times (1 + \alpha_{mobility} \times ||v_k(t)||_2)$

    if *aggregation condition triggered based on* $f_{agg}^{(k)}$ **then**

      Select FL neighbors:

      $\mathcal{N}_k^{FL}(t) = \{u_j \in \mathcal{N}_k(t) \mid RSSI_{k,j}(t) \geq RSSI_{FL}\}$

      `/* Parameter Exchange */`

      for *each neighbor* $j \in \mathcal{N}_k^{FL}(t)$ **do**

        UAV j computes gradient rank:

        $r_i = \text{rank}(|\partial \mathcal{L} / \partial \theta_i^{(j)}|)$ in descending order

        Determine bit width:

        $b_i = b_{min} + \lfloor (1 - \frac{r_i - 1}{n-1}) \times (b_{max} - b_{min}) \rfloor$

        Quantize parameters:

        $\theta_{quant,i}^{(j)} = \text{round}(\frac{\theta_i^{(j)} - \theta_{min}^{(j)}}{\theta_{max}^{(j)} - \theta_{min}^{(j)}} \times (2^{b_i} - 1))$

        UAV j sends $\{\boldsymbol{\theta}_{quant}^{(j)}, \{b_i\}, \theta_{max}^{(j)}, \theta_{min}^{(j)}, Rep_j(t)\}$ to UAV k

      end for

      `/* Parameter Reconstruction and Aggregation */`

      for *each successfully received neighbor* $j \in \mathcal{N}_k^{received}$ **do**

        Dequantize:

        $\theta_i^{(j)} = \theta_{min}^{(j)} + \frac{\theta_{quant,i}^{(j)}}{2^{b_i} - 1} \times (\theta_{max}^{(j)} - \theta_{min}^{(j)})$

      end for

      Compute normalized weights:

      $w_{k,j}^{normalized} = \frac{Rep_j(t)}{\sum_{l \in \{k\} \cup \mathcal{N}_k^{received}} Rep_l(t)}$ for $j \in \{k\} \cup \mathcal{N}_k^{received}$

      $\theta_{vel}^{(k)} \leftarrow \sum_{j \in \{k\} \cup \mathcal{N}_k^{received}} w_{k,j}^{normalized} \theta_{vel}^{(j)}$

      $\theta_{off}^{(k)} \leftarrow \sum_{j \in \{k\} \cup \mathcal{N}_k^{received}} w_{k,j}^{normalized} \theta_{off}^{(j)}$

      $\theta_{critic}^{(k)} \leftarrow \sum_{j \in \{k\} \cup \mathcal{N}_k^{received}} w_{k,j}^{normalized} \theta_{critic}^{(j)}$

    end if

  end while

end forpar

inFog's distributed learner infrastructure to deploy each UAV as an autonomous agent, comprising the GATs, trajectory/offloading actors, and the unified critic. Each UAV interacts with the environment, performs policy updates, and periodically exchanges model parameters with neighboring UAVs. To realize the federated learning protocol, we extend ReinFog's parameter synchronization mechanism by integrating reputation-weighted aggregation and gradient-sensitive adaptive quantization. Additionally, we develop a discrete-event simulator in Python following the system model in Section 8.3, which models UAV mobility, channel dynamics, task arrivals, and resource constraints, working in conjunction with ReinFog to provide environmental feedback for training. Parameter configurations adhere to typical UAV-MEC deployment scenarios.

System Deployment. The service area spans 1000×1000 m^2 with $K = 6$ UAVs operating at altitudes uniformly distributed in [80, 150] m [356]. UAV initial positions are determined via k-means clustering on IoT device locations [357]. $M = 40$ IoT devices are randomly distributed on the ground, simulating smart city sensor deployments [358]. For scalability analysis, we vary $K \in \{2, 4, 6, 8, 10\}$ and $M \in \{20, 40, 60, 80, 100\}$.

UAV Specifications. The computational frequency f_k of each UAV is uniformly sampled from [1, 3] GHz, reflecting heterogeneous edge computing platforms [359]. Maximum flight velocity $v_k^{\max} = 20$ m/s and acceleration capability $a_k = 5$ m/s^2 follow [360, 361]. Initial battery capacity $energy_k(0) = 500$ kJ follows current lithium polymer battery specifications [362].

Communication Environment. Inter-UAV communication range $R_{\text{comm}} = 400$ m is based on 2.4 GHz WiFi air-to-air transmission characteristics [363]. RSSI threshold $\text{RSSI}_{\min} = -90$ dBm follows IEEE 802.11 receiver sensitivity standards [364]. Channel gains are $G_0 = -30$ dB (air-to-ground) and $G_{\text{inter}} = -20$ dB (air-to-air) following free-space path loss models [365, 366]. Bandwidth allocation is $B = 10$ MHz for device-to-UAV links and $B_{\text{inter}} = 20$ MHz for inter-UAV links [367, 368]. Thermal noise power is $N_0 = -114$ dBm, derived from standard noise spectral density and bandwidth [369].

Power Configuration. UAV transmission and reception power are $P_k^{\text{tx}} = 0.5$ W and $P_k^{\text{rx}} = 0.1$ W, with IoT device transmission power $P_m^{\text{tx}} = 0.1$ W [362, 370, 371]. Hovering power $P_{\text{hover}} = 80$ W is based on medium-sized quadrotor measurements carrying computational payloads [372]. Aerodynamic parameters are air density $\rho = 1.225$ kg/m^3,

effective drag area $A = 0.1m^2$, and drag coefficient $C_D = 0.3$ [373, 374]. Computational energy uses CMOS effective switched capacitance $\kappa = 10^{-18}$ [375].

Task Model. IoT devices generate tasks following independent Poisson processes with rates λ_m uniformly sampled from [0.3, 0.8] tasks/s [376]. Task parameters include computational requirements $w_l \in [50, 200]$ Mcycles, input data $s_l \in [1, 3]$ MB, output data $d_l \in [0.1, 0.5]$ MB, and deadlines $c_l \in [5, 20]$ s, following [377–380].

Simulation Protocol. System performance is evaluated over 100 operational episodes, each spanning 300 s. Episode initialization employs k-means clustering on IoT device locations for UAV positioning. Each configuration undergoes 15 independent runs with different random seeds, with results reported as mean values.

Baseline Approaches

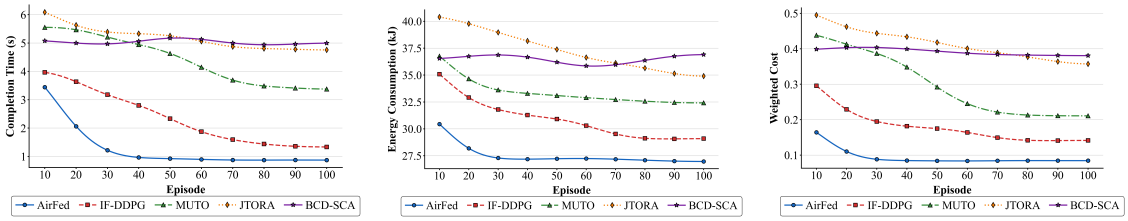
We compare our proposed AirFed against four state-of-the-art approaches:

- **IF-DDPG** [348]: A federated DDPG-based approach featuring dual experience replay and mixed noise exploration.
- **MUTO** [37]: A multi-agent Actor-Critic approach with prioritized experience replay for UAV trajectory optimization in differentiated service scenarios.
- **JTORA** [349]: A hybrid approach combining Lyapunov optimization for queue stability with SAC for UAV trajectory planning and resource allocation in dynamic MEC systems.
- **BCD-SCA** [346]: An iterative convex optimization combining BCD and SCA for joint trajectory and resource allocation.

These baselines collectively represent different solution paradigms: federated learning with independent agents (IF-DDPG), prioritized multi-agent DRL (MUTO), hybrid stochastic optimization with DRL (JTORA), and non-learning convex optimization (BCD-SCA). This selection enables comprehensive evaluation of AirFed across different methodological approaches.

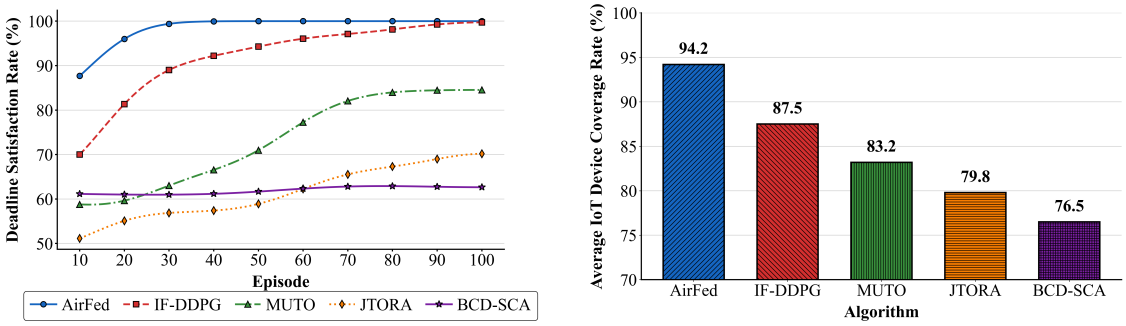
Table 8.2: Hyperparameter Configuration Summary

Parameter	Value
GATs Architecture	
GATs layers	2
GATs hidden layers	[128, 64]
Attention heads	4
GRU hidden dimension	128
Speed sensitivity α_{speed}	0.1
Task urgency weight γ_{urg}	0.5
Actor-Critic Architecture	
Shared feature layer	128
Velocity Actor hidden layers	[128, 128]
Offloading Actor hidden layers	[128, 128]
Critic hidden layers	[128, 64]
Activation function	ReLU
DRL Training	
Velocity Actor learning rate α_{vel}	3×10^{-4}
Offloading Actor learning rate α_{off}	3×10^{-4}
Critic learning rate α_{critic}	5×10^{-4}
Discount factor γ	0.95
Optimizer	Adam
Entropy regularization $\beta_{entropy}$	0.01
Reward Function Weights	
Time weight α	0.5
Energy weight β	0.5
Deadline penalty λ	10
Coverage reward η	0.1
Federated Learning	
FL communication RSSI threshold	-85 dBm
Minimum bit width b_{min}	4
Maximum bit width b_{max}	16
Base aggregation frequency f_{base}	0.03 Hz
Mobility sensitivity $\alpha_{mobility}$	0.05
Task success weight α_{succ}	0.6
Communication stability weight α_{stab}	0.4
Reputation smoothing factor ρ	0.75



(a) Task completion time **(b)** Energy consumption **(c)** Weighted cost

Figure 8.3: Convergence analysis of AirFed and baseline approaches across key performance metrics.



(a) Deadline satisfaction rate **(b)** IoT device coverage rate

Figure 8.4: Quality of Service analysis of AirFed and baseline approaches.

Hyperparameter Configuration

The network architecture employs 2-layer GATs (hidden layers [128, 64]) with 4-head attention mechanism, and GRU hidden dimension of 128. In the dual-Actor single-Critic architecture, the shared feature layer has 128 dimensions, two Actors each have 2 layers (hidden layers [128, 128]), and the Critic has 2 layers (hidden layers [128, 64]). Training parameters include discount factor $\gamma = 0.95$, Velocity Actor learning rate $\alpha_{vel} = 3 \times 10^{-4}$, Offloading Actor learning rate $\alpha_{off} = 3 \times 10^{-4}$, Critic learning rate $\alpha_{critic} = 5 \times 10^{-4}$, and entropy regularization coefficient $\beta_{entropy} = 0.01$. Reward function weights $\alpha = \beta = 0.5$ balance time and energy objectives, deadline penalty $\lambda = 10$, coverage reward $\eta = 0.1$. Federated learning parameters include quantization bit widths $b_{min} = 4$ and $b_{max} = 16$, reputation weights $\alpha_{succ} = 0.6$ and $\alpha_{stab} = 0.4$, smoothing factor $\rho = 0.75$. All key hyperparameters are optimized through grid search to ensure optimal performance. Detailed parameters are shown in Table 8.2.

Baseline approaches are implemented following their original designs. All baselines undergo the same hyperparameter tuning through grid search in our specific experimental setting. For hyperparameters reported in the original papers, we search within ranges around the reported values. For unreported hyperparameters, we conduct broader grid search to find optimal configurations. This ensures fair comparison under identical environmental conditions.

8.5.2 Convergence Analysis

Fig. 8.3 presents the convergence characteristics of AirFed and baseline approaches on the optimization objective defined in Eq. (8.31), evaluated through three key metrics: average task completion time (F_{time}), per-UAV energy consumption (F_{energy}), and weighted cost (F_{total}). AirFed demonstrates superior convergence behavior, achieving stable performance within 30-40 episodes across all metrics. In contrast, IF-DDPG and MUTO require 70-80 episodes to converge, while JTORA exhibits slower convergence, typically stabilizing after 80-100 episodes. As a non-learning approach, BCD-SCA maintains constant performance throughout the training process.

For task completion time (Fig. 8.3a), AirFed converges to around 0.9s, while IF-DDPG, MUTO, and JTORA converge to approximately 1.4s, 3.3s, and 4.7s respectively, and BCD-SCA remains at around 5.0s. Similarly, AirFed converges to around 27.0 kJ for energy consumption (Fig. 8.3b), compared to approximately 29.0 kJ (IF-DDPG), 32.5 kJ (MUTO), 35.0 kJ (JTORA), and 36.5 kJ (BCD-SCA). For weighted cost (Fig. 8.3c), AirFed converges to around 0.08, achieving approximately 42.9%, 61.9%, 77.8%, and 80.0% reduction compared to IF-DDPG (0.14), MUTO (0.21), JTORA (0.36), and BCD-SCA (0.38) respectively.

The rapid convergence of AirFed can be attributed to three key factors: the dual-layer GATs that capture spatial-temporal dependencies for informed decision-making, the dual-Actor architecture that efficiently handles hybrid action spaces, and the reputation-based federated learning that enables knowledge sharing across the UAV network. These mechanisms collectively accelerate the learning process compared to baseline approaches.

8.5.3 Quality of Service Analysis

Fig. 8.4 evaluates the QoS performance through deadline satisfaction rate and average IoT device coverage rate across episodes, driven by the QoS rewards Eq. (8.69) and Eq. (8.70).

For deadline satisfaction rate (Fig. 8.4a), AirFed reaches over 99% satisfaction rate within 30 episodes and maintains stability, while IF-DDPG eventually converges to the same level but requires approximately 90 episodes. MUTO converges to around 85%,

JTORA to around 70%, and BCD-SCA remains at approximately 63%. The superior performance of AirFed is attributed to the synergy between deadline penalty terms and federated learning: the penalty drives individual UAVs to learn deadline-aware policies, while federated learning enables rapid propagation of effective strategies across the network. Moreover, the dual-Actor architecture coordinates trajectory and task offloading decisions to satisfy timing constraints through dynamic position adjustment and multi-hop offloading paths.

For IoT device coverage rate (Fig. 8.4b), AirFed achieves 94.2%, outperforming IF-DDPG (87.5%), MUTO (83.2%), JTORA (79.8%), and BCD-SCA (76.5%) by 7.7%, 13.2%, 18.0%, and 23.2% respectively. This advantage stems from the coverage reward mechanism that incentivizes UAVs to optimize trajectories for broader device coverage, while the service layer in the dual-layer graph structure enables real-time awareness of coverage status.

8.5.4 Ablation Analysis

Fig. 8.5 illustrates the ablation study results of AirFed's key components. We evaluate three variants: the version without GATs (w/o GATs, replaced by MLP), the version without the reputation mechanism (w/o Reputation, using equal-weight aggregation), and the version without federated learning (w/o Fed, completely independent training).

Removing federated learning (w/o Fed) results in the most significant performance degradation, with the weighted cost increasing from 0.08 to approximately 0.13, representing a 62.5% increase. This variant requires about 70 episodes to converge, while the complete AirFed only needs 40 episodes, achieving a 42.9% improvement in convergence speed. Without federated learning, each UAV must independently explore the state space, leading to inefficient learning and difficulty in discovering globally optimal strategies.

Removing the reputation mechanism (w/o Reputation) increases the weighted cost to approximately 0.11, representing a 37.5% increase compared to the complete version, with convergence time extended to about 60 episodes. Without the reputation mecha-

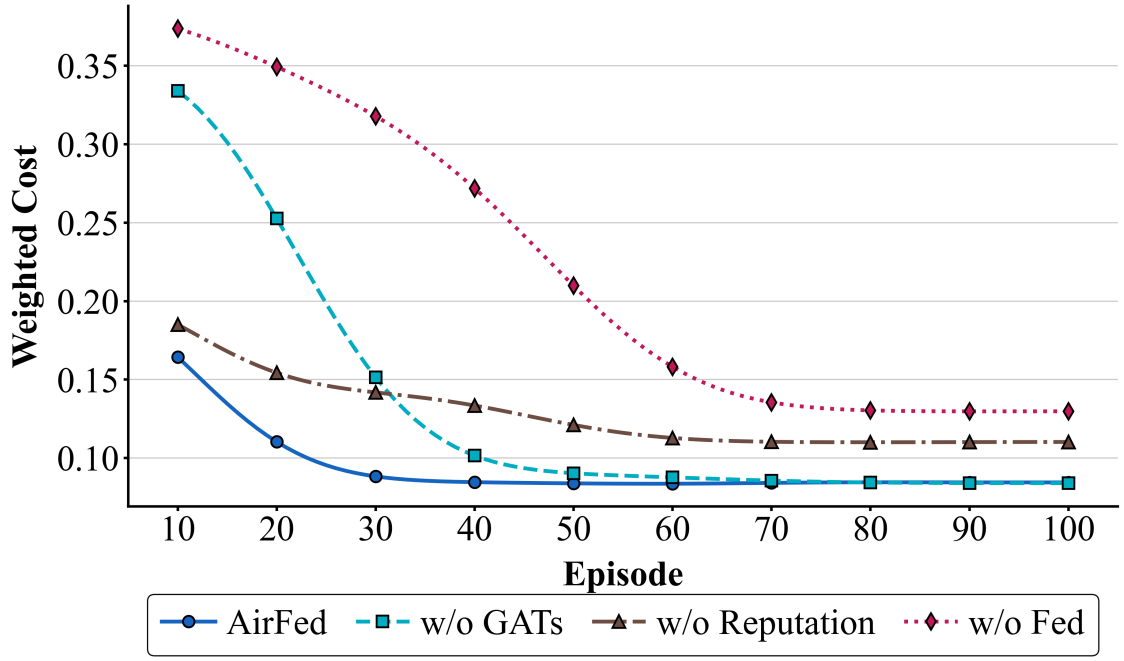


Figure 8.5: Ablation analysis of AirFed key components.

nism, all UAVs' model parameters are treated equally, causing low-quality or unstable policies to contaminate global knowledge, delaying convergence and reducing final performance. The reputation mechanism ensures that only high-quality policies propagate through the network by filtering unreliable updates.

Removing GATs (w/o GATs) achieves similar final performance to the complete version, but converges significantly slower, requiring about 70 episodes compared to AirFed's 40 episodes. This indicates that GATs play an important role in accelerating the learning process by effectively modeling spatial collaboration relationships among UAVs and dynamic network topology, enabling the system to explore and converge to optimal strategies more quickly.

8.5.5 Communication Efficiency Analysis

Fig. 8.6 presents the communication overhead comparison between AirFed and two baseline methods. Since MUTO, JTORA, and BCD-SCA do not employ federated learning frameworks and thus do not exchange model parameters among UAVs, they in-

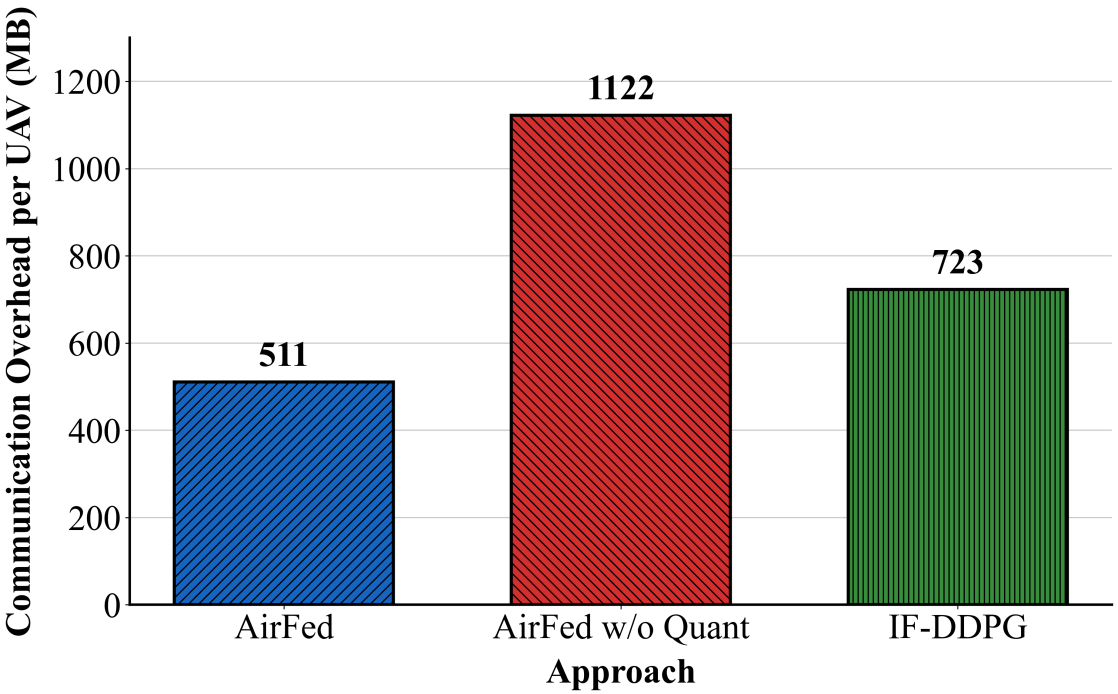
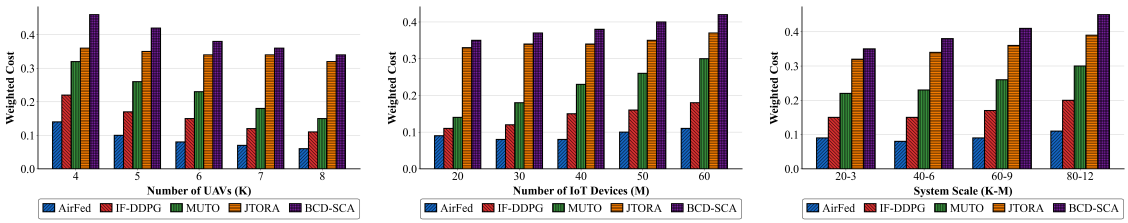


Figure 8.6: Communication overhead comparison of federated learning approaches.

cur no federated learning communication overhead. Therefore, to evaluate the impact of the gradient-sensitive adaptive quantization mechanism, we compare AirFed with AirFed w/o Quant (full-precision 32-bit parameter transmission) and IF-DDPG. AirFed employs gradient-sensitive adaptive quantization with $b_{min} = 4$ and $b_{max} = 16$ to reduce communication burden in federated learning.

The result shows that during the 100 episodes, AirFed achieves 511 MB communication overhead per UAV, outperforming AirFed w/o Quant (1122 MB) and IF-DDPG (723 MB) by 54.5% and 29.3% reduction, respectively. This stems from AirFed's gradient-sensitive adaptive quantization mechanism, which allocates higher bit precision to parameters with large gradients while compressing less sensitive parameters with lower bit widths, reducing bandwidth consumption while preserving critical learning information. This communication efficiency is particularly important for UAV-MEC systems with limited bandwidth and energy budgets.



(a) Varying UAV numbers **(b)** Varying IoT device numbers **(c)** Varying system scale

Figure 8.7: Scalability analysis of AirFed across UAV numbers, IoT device numbers, and system scale.

8.5.6 Scalability Analysis

To evaluate AirFed's scalability, we analyze its performance across three dimensions: UAV numbers, IoT device numbers, and system scale.

Fig. 8.7a shows the performance variation when UAV numbers increase from 4 to 8 with fixed IoT devices (M=40). The weighted cost of all approaches decreases as UAV numbers increase. At the 8-UAV configuration, AirFed achieves around 0.06, outperforming IF-DDPG (0.11), MUTO (0.15), JTORA (0.32), and BCD-SCA (0.34) by 45.5%, 60.0%, 81.3%, and 82.4%, respectively. Fig. 8.7b shows the impact when IoT devices increase from 20 to 60 with fixed UAVs (K=6). As device numbers increase, the weighted cost of all approaches rises. AirFed exhibits the smallest increase (from 0.09 to 0.11), and at the M=60 configuration outperforms IF-DDPG (0.18), MUTO (0.30), JTORA (0.37), and BCD-SCA (0.42) by by 38.9%, 63.3%, 70.3%, and 73.8%, respectively. Fig. 8.7c shows the performance when system scale expands from 20-3 to 80-12 (maintaining K/M$\approx$6.7). The weighted cost of all approaches increases with scale growth. At the 80-12 configuration, AirFed achieves 0.11, outperforming IF-DDPG (0.20), MUTO (0.30), JTORA (0.39), and BCD-SCA (0.45) by 45.0%, 63.3%, 71.8%, and 75.6%, respectively.

The superior scalability of AirFed across all three dimensions is attributed to several key architectural features. First, the decentralized federated learning mechanism fully exploits the collaborative potential of additional UAVs, with denser knowledge sharing networks accelerating policy propagation and global convergence as UAV numbers increase. Second, the coverage-aware reward mechanism drives UAVs to proactively adjust trajectories for broader device coverage, while the dual-layer GATs effectively

model complex spatial relationships arising from increased device density, maintaining robustness as IoT device numbers grow. Finally, the distributed decision-making architecture enables parallel processing across UAVs, and maintains computational efficiency as system scale expands. In contrast, baseline approaches suffer from performance saturation or significant degradation as system scale increases due to a lack of efficient collaboration mechanisms.

8.6 Summary

In this chapter, we propose AirFed, a federated graph-enhanced multi-agent reinforcement learning framework for multi-UAV cooperative mobile edge computing. It models spatial-temporal dependencies through a dual-layer dynamic graph attention network, handles hybrid action spaces via a dual-Actor single-Critic architecture, and achieves efficient knowledge sharing through reputation-based decentralized federated learning. Experimental results demonstrate that the AirFed framework reduces weighted cost by 42.9% compared to the best baseline, converges within 30-40 episodes, and achieves over 99% deadline satisfaction rate and 94.2% IoT device coverage rate. Ablation, communication efficiency and scalability analysis demonstrate the framework's effectiveness of each component and robust performance across different system scales.

Chapter 9
Conclusions and Future Directions

This chapter concludes the thesis by synthesizing the research contributions towards AI-driven resource management within the IoT-Edge-Cloud computing continuum. It first recapitulates the seven core contributions, ranging from the establishment of a theoretical taxonomy to the development of specific algorithms and frameworks that address challenges such as DAG task dependencies, large-scale concurrency, device heterogeneity, and cross-domain data privacy. Subsequently, the chapter distills methodological insights derived from these works, emphasizing the critical role of neural architecture specificity, the necessity of bridging the gap between algorithmic theory and system deployment, and the intrinsic trade-off between efficiency and robustness in decentralized architectures. Finally, envisioning the transition from the Internet of Everything to the Intelligence of Everything, the chapter outlines a roadmap for future research, highlighting directions such as foundation model-driven edge intelligence, causal reinforcement learning, space-air-ground integration, and carbon-aware sustainable computing to build a ubiquitous, trustworthy, and evolving intelligent infrastructure.

9.1 Summary of the Thesis

With the deep integration of the IoT, edge computing, and cloud computing technologies, the computing paradigm is undergoing a fundamental shift from a solitary cloud center to the IoT-Edge-Cloud continuum. This shift significantly expands the boundaries of computing but also brings severe challenges such as the exponential growth in system scale, extreme infrastructure heterogeneity, highly dynamic workloads, and data privacy constraints. With AI-driven approaches as the core methodology, this thesis constructs a complete resource management solution suite to address these challenges through seven interrelated research works. These seven main contributions are summarized as follows:

1. **Construction of a Two-Dimensional Taxonomy for DRL-based Resource Management (corresponding to Chapter 2).** To address the fragmentation of theoretical research in this field, this thesis first establishes an orthogonal two-dimensional taxonomy. By systematically reconstructing the existing literature landscape through two dimensions: control scope (single-agent vs. multi-agent) and training paradigm (standard vs. federated), this work not only clarifies the applicability boundaries of different architectures but also precisely identifies the gaps in federated and decentralized architectures in current research, laying a theoretical foundation for the subsequent research paths of this thesis.

2. **Proposal of the Adaptive Cost Optimization DRLIS Scheduling Algorithm (corresponding to Chapter 3).** Addressing fundamental scheduling difficulties in edge-fog computing environments, this thesis proposes the DRLIS algorithm. Facing IoT applications with complex DAG dependency structures, the algorithm optimizes the trade-off between response time and system load balancing in stochastic dynamic environments through adaptive decision-making policies and composite reward functions.

3. **Design of the TF-DDRL Distributed Technique for Large-Scale Concurrency (corresponding to Chapter 4).** To break through scalability bottlenecks in large-scale heterogeneous scenarios, this thesis develops the Transformer-fused distributed DRL technique TF-DDRL. By utilizing Gated Transformer-XL to capture long-term spatiotemporal dependencies of task flows, and combining the IMPALA distributed architecture with prioritized experience replay mechanisms, this technique significantly reduces exploration costs when processing massive concurrent requests, substantially improving sample efficiency and convergence speed.

4. **Implementation of the Unified Resource Management Framework ReinFog and Deployment Optimization (corresponding to Chapter 5).** To bridge the gap between algorithmic theory and system implementation, this thesis designs and implements the ReinFog framework. This is a modular, container-supported software platform capable of seamlessly integrating various centralized and distributed DRL algorithms. Furthermore, addressing the deployment overhead of DRL com-

ponents themselves, the MADCP memetic algorithm is proposed to minimize communication latency and startup time of the intelligent decision-making system by optimizing the physical placement of Learners and Workers.

5. **Proposal of the KD-AFRL Federated Framework Supporting Device Heterogeneity (corresponding to Chapter 6).** Addressing the model incompatibility problem caused by differences in device computational power in multi-domain collaboration, this thesis proposes the KD-AFRL framework. By introducing resource-aware adaptive architecture generation mechanisms and environment-oriented cross-architecture knowledge distillation technologies, the framework successfully breaks structural barriers between heterogeneous devices, allowing lightweight edge devices to inherit the decision-making intelligence of cloud-based large models through federated learning, while simultaneously accounting for data privacy protection.

6. **Construction of the Attack-Resistant Decentralized DeFRiS Collaboration Framework (corresponding to Chapter 7).** Facing data silos and adversarial risks in cross-organizational collaboration, this thesis develops the fully decentralized DeFRiS framework. The framework innovatively adopts action-space-agnostic policy networks to adapt to heterogeneous infrastructures and designs a dual-track robust aggregation protocol based on gradient fingerprints and gradient tracking. Experiments demonstrate that DeFRiS ensures robust system convergence and efficient collaboration even under harsh environments with Non-IID data distributions and malicious node attacks.

7. **Development of the AirFed Multi-Agent Framework for High-Dynamic Topologies (corresponding to Chapter 8).** Targeting extreme dynamic scenarios such as aerial computing networks, this thesis proposes the AirFed framework. The framework fuses dual-layer dynamic Graph Attention Networks (GATs) to explicitly model spatiotemporal topology evolution features and designs a dual-Actor single-Critic architecture to jointly optimize continuous UAV trajectory control and discrete task offloading. Combined with a reputation-based federated mechanism, AirFed effectively guarantees the QoS of mobile edge computing services under strict communication and coverage constraints.

In summary, the research work of this thesis covers a full-stack exploration from basic theory and core algorithms to system implementation and complex cross-domain collaboration. Through the seven innovations described above, this research systematically addresses the efficiency, scalability, compatibility, robustness, and adaptability issues of resource management in the IoT-Edge-Cloud continuum. This not only verifies the superiority of AI-driven approaches in handling highly complex distributed systems but also provides solid theoretical grounds and technical support for building the next generation of ubiquitous, intelligent, and trustworthy computing infrastructure.

9.2 Key Findings and Insights

Through a systematic investigation of resource management problems in the IoT-Edge-Cloud continuum, this thesis not only proposes a series of specific solutions but also synthesizes several methodological insights with universal significance derived from these seven research works. These findings reveal critical laws that must be adhered to when building the next generation of distributed intelligent systems.

1. **Multidimensional Taxonomy as a Compass for Algorithm Design (based on Taxonomy Research).** In this highly fragmented research field, establishing an orthogonal taxonomy serves not merely as a tool for literature review but as a starting point for algorithmic innovation. This research finds that explicitly distinguishing between the two dimensions of control scope and training paradigm enables researchers to precisely locate structural defects in existing solutions. It is based on this theoretical awareness that this thesis was able to systematically fill the gaps in the high-difficulty quadrants of federated multi-agent and decentralized collaboration in subsequent chapters.

2. **Neural Architecture Specificity Determines Performance Ceilings in Complex Environments (based on DRLIS, TF-DDRL, and AirFed Research).** By comparing the basic networks in DRLIS, the Transformer in TF-DDRL, and the GNNs in AirFed, this research reveals an evolutionary pattern: as environmental complexity increases, generic DRL models rapidly encounter performance bottlenecks. To

resolve large-scale concurrency or high-dynamic topology problems, prior knowledge of the task, such as temporal dependencies or spatial structures, must be explicitly encoded into the neural architecture as inductive bias. This implies that in continuum resource management, specialized architecture adaptation is currently the optimal solution rather than generic artificial intelligence.

3. **A Non-Ignorable Deployment Gap Exists Between Algorithmic Theory and System Implementation (based on ReinFog Research).** The development process of the ReinFog framework reveals a fact often overlooked by theoretical research: intelligent algorithms themselves are resource-consuming heavy workloads. In resource-constrained edge environments, if the inference latency, model loading time, and communication overhead of DRL agents are not considered, theoretical scheduling gains may be completely offset by system losses. The success of the MADCP algorithm proves that only by incorporating the physical deployment of intelligent components into the optimization scope and achieving algorithm-system co-design can the potential of AI be truly unleashed.

4. **Knowledge Abstraction is an Effective Bridge Over the Heterogeneity Gap (based on KD-AFRL Research).** In multi-domain collaboration scenarios, physical differences in device computational power are typically viewed as the greatest obstacle to federated learning deployment. However, the research on KD-AFRL brings a critical turning point: although physical resources and model structures cannot be unified, decision logic can be abstracted and transferred. By compressing complex policies into lightweight soft targets through knowledge distillation, we demonstrate that efficient knowledge sharing can be achieved without sharing raw data or enforcing unified model structures. This indicates that in heterogeneous systems, collaboration based on knowledge abstraction is more viable than collaboration based on parameter synchronization.

5. **Decentralized Architectures Achieve a Dynamic Trade-off Between Efficiency and Robustness (based on DeFRiS and AirFed Research).** By comparing centralized and decentralized schemes, specifically DeFRiS, this research discovers a profound trade-off relationship between the two. While centralized architectures

converge faster in ideal environments, they appear fragile when facing Non-IID data distribution shifts and adversarial attacks. Conversely, decentralized architectures, despite higher initial communication overhead, demonstrate strong self-healing capabilities through defense mechanisms such as gradient fingerprints. This insight suggests that future trustworthy computing infrastructures must accept moderate communication redundancy as a necessary cost at the design stage to obtain system-level security and robustness.

9.3 Future Research Directions

Building upon the research findings of this thesis in resource management for the IoT-Edge-Cloud continuum, future work can be further deepened across multiple dimensions including algorithmic theoretical innovation, network architecture evolution, trustworthy security assurance, and underlying hardware-software co-design. These directions aim to propel computing infrastructure from single-dimensional resource optimization toward a next-generation intelligent ecosystem that is more general-purpose, green, ubiquitous, trustworthy, and capable of continuous evolution.

9.3.1 Foundation Model-Driven General-Purpose Edge Intelligence

Current DRL scheduling algorithms are mostly specialized models trained for specific task scenarios, exhibiting limited generalization capabilities and facing cold-start challenges when encountering unseen scenarios. With breakthroughs in generative artificial intelligence, future research should explore utilizing Large Language Models (LLMs) or Large Multimodal Models (LMMs) as the core brain of edge agents [381]. Leveraging the powerful zero-shot reasoning capabilities and commonsense knowledge bases of foundation models can, on one hand, assist agents in rapidly generating initial scheduling policies or reward functions in unknown heterogeneous environments, significantly reducing the trial-and-error cost of online exploration. On the other hand, it enables more intuitive intent-driven resource management through natural language interaction, allowing operators to control complex underlying systems via natural language

directives, thereby realizing a paradigm shift from specialized algorithm optimization to general-purpose intelligent agents.

9.3.2 Explainable Reinforcement Learning with Causal Inference

Current DRL models make decisions primarily based on statistical correlations between data, making them susceptible to spurious correlations in the environment and lacking intrinsic interpretability [57]. Future research should commit to introducing causal inference theory [382] into the resource scheduling domain to construct Causal Reinforcement Learning (Causal RL) frameworks. By building Structural Causal Models (SCM) of the environment, agents can not only answer "what" to do but also understand "why" and reason about counterfactual "what-if" scenarios. This will vastly enhance the model's generalization capabilities in Out-of-Distribution (OOD) scenarios and provide human-logical causal explanations for scheduling decisions, thereby strengthening trust in human-machine collaboration within critical infrastructure.

9.3.3 Overcoming Catastrophic Forgetting via Lifelong Learning

IoT environments possess high non-stationarity, with task distributions and network topologies evolving continuously over time. Current DRL models often suffer from catastrophic forgettinglosing old knowledge when adapting to new environments. Future research should introduce lifelong learning or continual learning mechanisms [383] to endow scheduling agents with memory consolidation capabilities similar to biological systems. Through techniques such as Elastic Weight Consolidation (EWC) or experience replay, agents can continuously accumulate knowledge over long lifecycles, maintaining efficient scheduling capabilities for legacy services while adapting to new businesses, thus achieving continuous evolution of intelligence.

9.3.4 Space-Air-Ground-Sea Integrated Ubiquitous Computing Continuum

Current research on aerial computing is predominantly confined to near-ground UAV networks. Future research should further break through physical boundaries, expand-

ing the horizon of resource management to Space-Air-Ground-Sea Integrated Networks [384]. In this vision, Low Earth Orbit (LEO) satellites, High Altitude Platform Stations (HAPS), UAV swarms, and terrestrial/underwater IoT will constitute a multi-dimensional, extremely dynamic heterogeneous computing continuum. Future research needs to address resource orchestration problems at planetary spatiotemporal scales, cope with intermittent connectivity challenges caused by orbital dynamics, and realize seamless task handover and continuity assurance within multi-tier heterogeneous networks, thereby achieving truly global ubiquitous computing.

9.3.5 Deep Convergence of Semantic Communication and Computation

In bandwidth-constrained edge environments, the traditional bit-transmission mode has become a bottleneck for distributed intelligent collaboration. Future research should explore the deep convergence of semantic communication and edge computing [385], shifting from transmitting data itself to transmitting the semantic information of data. By designing task-oriented semantic encoders, agents need only exchange compressed semantic vectors containing key decision features rather than raw sensor data or complete model parameters. This joint optimization of communication and computation will significantly reduce communication overhead in distributed collaboration, enhancing system efficiency in extreme network environments.

9.3.6 Digital Twin-Driven Sim-to-Real Transfer and Closed-Loop Optimization

To address the uncertainty risks brought by online exploration of RL algorithms in real environments, future research should deeply integrate Digital Twin technology [306]. By constructing virtual twins with high-fidelity mapping to physical systems, agents can conduct large-scale parallel trial-and-error and stress testing in virtual space at extremely low costs. Research emphasis should be placed on Sim-to-Real transfer technologies to ensure that policies trained in twin environments can seamlessly migrate to physical environments, while utilizing Inverse Reinforcement Learning (IRL) to continuously calibrate twin models with real feedback, forming a closed loop of sensing,

simulation, decision-making, and optimization [386].

9.3.7 Privacy Protection and Secure Collaboration Based on Zero-Trust Architecture

With the deepening of cross-domain collaboration, traditional perimeter security models are no longer applicable. Future resource management systems should evolve towards a Zero Trust architecture. Building upon decentralized federated learning, future research should explore full-stack privacy protection schemes [33] combining Multi-Party Computation (MPC), Zero-Knowledge Proofs (ZKP), and blockchain technologies [387]. System designs must not only defend against adversarial attacks such as model poisoning but also mathematically guarantee that data remains usable but invisible during the scheduling process. For instance, using ZKP to verify the authenticity of computing resources at edge nodes without revealing specific task details can construct a trusted resource collaboration ecosystem that is transparent, verifiable, and privacy-preserving.

9.3.8 Quantum-Resistant Resource Orchestration via PQC-Aware Scheduling

With the rapid advancement of quantum computing, traditional cryptographic primitives (e.g., RSA, ECC) widely used in IoT are facing existential threats. Transitioning to Post-Quantum Cryptography (PQC) is inevitable but introduces significant computational and communication overheads, which can be prohibitive for resource-constrained edge devices [388]. Future research should investigate PQC-aware intelligent scheduling. By utilizing DRL to dynamically select appropriate PQC algorithm levels (e.g., Lattice-based vs. Hash-based) or offload computationally intensive cryptographic operations to trusted edge/cloud nodes based on real-time threat levels and resource availability, a balance can be struck between long-term quantum security and immediate system performance (latency and energy).

9.3.9 Neuromorphic Computing and Extreme Edge Intelligence for Microcontrollers

Targeting the resource-constrained devices widely present in the IoT ecosystem, future work should extend intelligent decision-making capabilities down to Microcontroller Unit (MCU) devices with KB-level memory, namely the TinyML domain [51]. Beyond more aggressive model compression technologies, novel computing paradigms surpassing traditional deep learning, such as Spiking Neural Networks (SNNs) [389], should be explored. Due to their event-driven and low-power characteristics, SNNs naturally align with asynchronous data streams at the edge. Future research can explore utilizing neuromorphic hardware to achieve nano-watt level online inference and enable training at the edge, allowing terminal devices like sensors to possess local lifelong learning capabilities, thereby significantly alleviating bandwidth bottlenecks.

9.3.10 Hardware-Software Co-Design for AI-Native Network Infrastructure

Current resource management mostly remains at the application or operating system layer. Future research should penetrate to the underlying layers, exploring hardware-software co-design [390], considering nodes (cloud and/or edge) with Quantum computing capabilities. This includes designing Network Interface Cards (SmartNICs) or Reconfigurable Intelligent Surfaces (RIS) specifically optimized for DRL inference, endowing the network infrastructure itself with native intelligent scheduling capabilities. By offloading parts of the scheduling logic to programmable switches (e.g., P4) in the data plane, nanosecond-level ultra-low latency scheduling can be achieved, breaking through the performance bottlenecks of traditional control plane decision-making to build an AI-native network infrastructure.

9.3.11 Shift from Energy Efficiency to Carbon-Aware Sustainable Computing

Traditional resource management efforts are primarily dedicated to reducing the absolute magnitude of system energy consumption. However, against the macro background of global commitment to net-zero emissions, future resource management must ascend to the dimension of carbon footprint. Future research should combine real-time carbon

intensity data from power grids with the fluctuation characteristics of distributed renewable energy, designing spatiotemporally aware carbon-aware scheduling algorithms [391]. By deferring non-critical tasks to periods of abundant green power in the time dimension, or migrating computing loads to data center nodes with lower carbon emission intensity in the spatial dimension, refined full-lifecycle carbon emission quantification models can be established. This will establish carbon reduction as a core optimization objective of equal importance to latency and energy consumption.

9.4 Final Remarks

The journey of this doctoral research began with the exploration of scheduling efficiency for heterogeneous computing resources. However, as the research deepened, its destination has far transcended single algorithm optimization, pointing towards a profound reconstruction of the entire computing paradigm. The IoT-Edge-Cloud continuum is no longer merely a physical stack composed of silicon, fiber optics, and radio waves; it is evolving into a global digital organism possessing an autonomous nervous system. The theoretical models, algorithmic techniques, and system frameworks proposed in this thesis are essentially attempts to imbue this colossal organism with the capabilities of perception, cognition, and evolution.

Our research indicates that AI is no longer an optional auxiliary component in IoT-Edge-Cloud computing architectures, but an inseparable intrinsic gene of next-generation infrastructure. Through system-level architectural implementations, cross-domain knowledge abstraction, and decentralized collaboration mechanisms, we have demonstrated that building an adaptive, self-organizing, and robust intelligent resource management system in an extremely complex, dynamic, and constrained physical world is not only feasible but inevitable. This AI-native transformation marks that we are transcending the initial stage of the Internet of Everything and striding towards the new era of the Intelligence of Everything.

However, technological progress must be accompanied by reflections on responsibility. While pursuing millisecond-level latency and extreme computing power, future computing infrastructure must uphold the bottom line of trustworthiness and sustain-

ability. It should be a guardian of privacy rather than an intruder; it should be a contributor to a green earth rather than an excessive consumer of energy. The ultimate resource management should be invisible. Similar to electricity and air, intelligent computing power should permeate every corner of our lives, available on demand and unobtrusive in operation. The research work of this thesis serves as a solid cornerstone leading towards this future vision of ubiquitous, trustworthy, and invisible computing.

Bibliography

[1] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of things (iot): A vision, architectural elements, and future directions," *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1645–1660, 2013.

[2] L. Kong, J. Tan, J. Huang, G. Chen, S. Wang, X. Jin, P. Zeng, M. Khan, and S. K. Das, "Edge-computing-driven internet of things: A survey," *ACM Computing Surveys*, vol. 55, no. 8, pp. 1–41, 2022.

[3] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Generation computer systems*, vol. 25, no. 6, pp. 599–616, 2009.

[4] R. Mahmud, R. Kotagiri, and R. Buyya, "Fog computing: A taxonomy, survey and future directions," *Internet of everything: algorithms, methodologies, technologies and perspectives*, pp. 103–130, 2018.

[5] G. K. Walia, M. Kumar, and S. S. Gill, "Ai-empowered fog/edge resource management for iot applications: A comprehensive review, research challenges, and future perspectives," *IEEE Communications Surveys & Tutorials*, vol. 26, no. 1, pp. 619–669, 2024.

[6] Y. Wang, C. Yang, S. Lan, L. Zhu, and Y. Zhang, "End-edge-cloud collaborative computing for deep learning: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 26, no. 4, pp. 2647–2683, 2024.

[7] J. Ren, D. Zhang, S. He, Y. Zhang, and T. Li, "A survey on end-edge-cloud orchestrated network computing paradigms: Transparent computing, mobile edge computing, fog computing, and cloudlet," *ACM Computing Surveys*, vol. 52, no. 6, pp. 1–36, 2019.

[8] N. Yang, S. Chen, H. Zhang, and R. Berry, "Beyond the edge: An advanced exploration of reinforcement learning for mobile edge computing, its applications, and future research trajectories," *IEEE Communications Surveys & Tutorials*, vol. 27, no. 1, pp. 546–594, 2025.

[9] M. Goudarzi, M. Palaniswami, and R. Buyya, "Scheduling iot applications in edge and fog computing environments: a taxonomy and future directions," *ACM Computing Surveys*, vol. 55, no. 7, pp. 1–41, 2022.

[10] X. Zhang and S. Debroy, "Resource management in mobile edge computing: A comprehensive survey," *ACM Computing Surveys*, vol. 55, no. 13s, pp. 1–37, 2023.

[11] X. Zhou, S. Ge, P. Liu, and T. Qiu, "Dag-based dependent tasks offloading in mec-enabled iot with soft cooperation," *IEEE Transactions on Mobile Computing*, 2023.

[12] N. Rasouli, C. Klein, and E. Elmroth, "Resource management for mission-critical applications in edge computing: systematic review on recent research and open issues," *ACM Computing Surveys*, vol. 58, no. 3, pp. 1–37, 2025.

[13] B. Jamil, H. Ijaz, M. Shojafar, K. Munir, and R. Buyya, "Resource allocation and task scheduling in fog computing and internet of everything environments: A taxonomy, review, and future directions," *ACM Computing Surveys (CSUR)*, vol. 54, no. 11s, pp. 1–38, 2022.

[14] M. Ye, X. Fang, B. Du, P. C. Yuen, and D. Tao, "Heterogeneous federated learning: State-of-the-art and research challenges," *ACM Computing Surveys*, vol. 56, no. 3, pp. 1–44, 2023.

[15] K. Pfeiffer, M. Rapp, R. Khalili, and J. Henkel, "Federated learning for computationally constrained heterogeneous devices: A survey," *ACM Computing Surveys*, vol. 55, no. 14s, pp. 1–27, 2023.

[16] R. Jeyaraj, A. Balasubramaniam, A. K. MA, N. Guizani, and A. Paul, "Resource management in cloud and cloud-influenced technologies for internet of things applications," *ACM Computing Surveys*, vol. 55, no. 12, pp. 1–37, 2023.

[17] S. Padakandla, "A survey of reinforcement learning algorithms for dynamically varying environments," *ACM Computing Surveys*, vol. 54, no. 6, pp. 1–25, 2021.

[18] M. Goudarzi, M. S. Palaniswami, and R. Buyya, "A distributed deep reinforcement learning technique for application placement in edge and fog computing en-

vironments," *IEEE Transactions on Mobile Computing*, vol. 22, no. 5, pp. 2491–2505, 2023.

[19] W. Chen, X. Qiu, T. Cai, H.-N. Dai, Z. Zheng, and Y. Zhang, "Deep reinforcement learning for internet of things: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 3, pp. 1659–1692, 2021.

[20] R. S. Sutton, A. G. Barto *et al.*, *Reinforcement learning: An introduction*. MIT press Cambridge, 1998, vol. 1, no. 1.

[21] Z. Zabihi, A. M. Eftekhari Moghadam, and M. H. Rezvani, "Reinforcement learning methods for computation offloading: a systematic review," *ACM Computing Surveys*, vol. 56, no. 1, pp. 1–41, 2024.

[22] Z. Lu, H. Pan, Y. Dai, X. Si, and Y. Zhang, "Federated learning with non-iid data: A survey," *IEEE Internet of Things Journal*, vol. 11, no. 11, pp. 19 188–19 209, 2024.

[23] D. Hortelano, I. de Miguel, R. J. D. Barroso, J. C. Aguado, N. Merayo, L. Ruiz, A. Asensio, X. Masip-Bruin, P. Fernández, R. M. Lorenzo *et al.*, "A comprehensive survey on reinforcement-learning-based computation offloading techniques in edge computing systems," *Journal of Network and Computer Applications*, vol. 216, p. 103669, 2023.

[24] Z. Liu, L. Huang, Z. Gao, M. Luo, S. Hosseinalipour, and H. Dai, "Ga-drl: Graph neural network-augmented deep reinforcement learning for dag task scheduling over dynamic vehicular clouds," *IEEE Transactions on Network and Service Management*, vol. 21, no. 4, pp. 4226–4242, 2024.

[25] R. Xie, L. Feng, Q. Tang, T. Huang, Z. Xiong, T. Chen, and R. Zhang, "Delay-prioritized and reliable task scheduling with long-term load balancing in computing power networks," *IEEE Transactions on Services Computing*, vol. 17, no. 6, pp. 3359–3372, 2024.

[26] Z. Wang, M. Goudarzi, M. Gong, and R. Buyya, "Deep reinforcement learning-based scheduling for optimizing system load and response time in edge and fog

computing environments," *Future Generation Computer Systems*, vol. 152, pp. 55–69, 2024.

[27] Z. Zhang, F. Zhang, Z. Xiong, K. Zhang, and D. Chen, "Lsia3cs: Deep-reinforcement-learning-based cloudedge collaborative task scheduling in large-scale iiot," *IEEE Internet of Things Journal*, vol. 11, no. 13, pp. 23 917–23 930, 2024.

[28] S. Tang, Y. Yu, H. Wang, G. Wang, W. Chen, Z. Xu, S. Guo, and W. Gao, "A survey on scheduling techniques in computing and network convergence," *IEEE Communications Surveys & Tutorials*, vol. 26, no. 1, pp. 160–195, 2024.

[29] S. Moreschini, E. Younesian, D. Hästbacka, M. Albano, J. Hošek, and D. Taibi, "Edge to cloud tools: A multivocal literature review," *Journal of Systems and Software*, vol. 210, p. 111942, 2023.

[30] M. Goudarzi, M. A. Rodriguez, M. Sarvi, and R. Buyya, "μ-ddrl: A qos-aware distributed deep reinforcement learning technique for service offloading in fog computing environments," *IEEE Transactions on Services Computing*, vol. 17, no. 1, pp. 47–59, 2024.

[31] Z. Wang, M. Goudarzi, and R. Buyya, "Reinfog: A deep reinforcement learning empowered framework for resource management in edge and cloud computing environments," *Journal of Network and Computer Applications*, vol. 242, p. 104250, 2025.

[32] W.-C. Chung, C.-A. Lo, Y.-H. Lin, Z.-H. Chen, and C.-L. Hung, "Decentralized federated learning with non-iid data: Challenges, trends, and future opportunities," *ACM Computing Surveys*, 2025.

[33] J. Zhao, S. Bagchi, S. Avestimehr, K. Chan, S. Chaterji, D. Dimitriadis, J. Li, N. Li, A. Nourian, and H. Roth, "The federation strikes back: A survey of federated learning privacy attacks, defenses, applications, and policy landscape," *ACM Computing Surveys*, vol. 57, no. 9, pp. 1–37, 2025.

[34] M. P. Uddin, Y. Xiang, M. Hasan, J. Bai, Y. Zhao, and L. Gao, "A systematic liter-

ature review of robust federated learning: Issues, solutions, and future research directions," *ACM Computing Surveys*, vol. 57, no. 10, pp. 1–62, 2025.

[35] Y. Huang, S. Gupta, Z. Song, K. Li, and S. Arora, "Evaluating gradient inversion attacks and defenses in federated learning," *Advances in neural information processing systems*, vol. 34, pp. 7232–7241, 2021.

[36] Z. Ning, H. Hu, X. Wang, L. Guo, S. Guo, G. Wang, and X. Gao, "Mobile edge computing and machine learning in the internet of unmanned aerial vehicles: a survey," *ACM Computing Surveys*, vol. 56, no. 1, pp. 1–31, 2023.

[37] Z. Ning, Y. Yang, X. Wang, Q. Song, L. Guo, and A. Jamalipour, "Multi-agent deep reinforcement learning based uav trajectory optimization for differentiated services," *IEEE Transactions on Mobile Computing*, vol. 23, no. 5, pp. 5818–5834, 2024.

[38] Z. Tong, J. Wang, J. Chen, X. Zhang, H. Du, and J. Liu, "Airfrl: Topology-aware decentralized federated reinforcement learning for uav networks," in *Proceedings of the IEEE Vehicular Technology Conference*, 2025, pp. 1–6.

[39] Y. Zhou, L. Feng, X. Jiang, W. Li, and F. Zhou, "Predictable wireless networked scheduling for bridging hybrid time-sensitive and real-time services," *IEEE Transactions on Communications*, vol. 72, no. 6, pp. 3664–3680, 2024.

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

[41] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *Proceedings of the International conference on machine learning*. PMLR, 2015, pp. 1889–1897.

[42] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[43] L. Chen, K. Lu, A. Rajeswaran, K. Lee, A. Grover, M. Laskin, P. Abbeel, A. Srinivas, and I. Mordatch, "Decision transformer: Reinforcement learning via sequence

modeling," *Advances in neural information processing systems*, vol. 34, pp. 15 084–15 097, 2021.

[44] Q. Deng, M. Goudarzi, and R. Buyya, "Fogbus2: a lightweight and distributed container-based framework for integration of iot-enabled systems with edge and cloud computing," in *Proceedings of the International Workshop on Big Data in Emergent Distributed Environments*, 2021, pp. 1–8.

[45] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[46] Y. Chen, J. Zhang, Z. Wang, W. Dai, H. Gao, and S. Deng, "Privacy-preserving knowledge distillation in latency-critical federated task offloading for consumer iot networks," *IEEE Transactions on Consumer Electronics*, vol. 71, no. 1, pp. 239–251, 2025.

[47] E. T. M. Beltrán, M. Q. Pérez, P. M. S. Sánchez, S. L. Bernal, G. Bovet, M. G. Pérez, G. M. Pérez, and A. H. Celdrán, "Decentralized federated learning: Fundamentals, state of the art, frameworks, trends, and challenges," *IEEE Communications Surveys & Tutorials*, vol. 25, no. 4, pp. 2983–3013, 2023.

[48] H. Li and Z. Lin, "Accelerated gradient tracking over time-varying graphs for decentralized optimization," *Journal of Machine Learning Research*, vol. 25, no. 274, pp. 1–52, 2024.

[49] P. Velikovi, G. Cucurull, A. Casanova, A. Romero, P. Li, and Y. Bengio, "Graph attention networks," in *International Conference on Learning Representations*, 2018.

[50] H. Wu, L. Tian, H. Tang, R. Li, and P. Jiao, "Graph convolutional reinforcement learning-guided joint trajectory optimization and task offloading for aerial edge computing," *IEEE Transactions on Intelligent Transportation Systems*, vol. 26, no. 10, pp. 17 487–17 498, 2025.

[51] X. Wang, Z. Tang, J. Guo, T. Meng, C. Wang, T. Wang, and W. Jia, "Empowering edge intelligence: A comprehensive survey on on-device ai models," *ACM Computing Surveys*, vol. 57, no. 9, pp. 1–39, 2025.

[52] I. Taleb, J.-L. Guillaume, and B. Duthil, "A survey on services placement algorithms in integrated cloud-fog/edge computing," *ACM Computing Surveys*, vol. 57, no. 11, pp. 1–36, 2025.

[53] X. He, Y. Jiang, H. Cui, Y. Liu, M. Chen, M. Guizani, and S. Mumtaz, "Qoe-driven proactive caching with drl in sustainable cloud-to-edge continuum," *IEEE Transactions on Mobile Computing*, vol. 24, no. 10, pp. 10 992–11 004, 2025.

[54] Z. Wang, M. Goudarzi, and R. Buyya, "Tf-ddrl: A transformer-enhanced distributed drl technique for scheduling iot applications in edge and cloud computing environments," *IEEE Transactions on Services Computing*, vol. 18, no. 2, pp. 1039–1053, 2025.

[55] Z. Wei, J. Mao, B. Li, and R. Zhang, "Privacy-preserving hierarchical reinforcement learning framework for task offloading in low-altitude vehicular fog computing," *IEEE Open Journal of the Communications Society*, vol. 6, pp. 3389–3403, 2025.

[56] H. Yuan, T. Wang, M. Fu, and Y. Shi, "Girp: Energy-efficient qos-oriented microservice resource provisioning via multi-objective multi-task reinforcement learning," *IEEE Transactions on Mobile Computing*, vol. 24, no. 7, pp. 5793–5807, 2025.

[57] S. Milani, N. Topin, M. Veloso, and F. Fang, "Explainable reinforcement learning: A survey and comparative review," *ACM Computing Surveys*, vol. 56, no. 7, pp. 1–36, 2024.

[58] H. Chen, Y. Lin, M. Fu, L. Yao, and M. Sheng, "A survey on reinforcement learning methods for uav systems," *ACM Computing Surveys*, vol. 58, no. 4, pp. 1–37, 2025.

[59] Z. Tong, J. Deng, J. Mei, Y. Zhang, and K. Li, "Multi-objective dag task offloading in mec environment based on federated dqn with automated hyperparameter optimization," *IEEE Transactions on Services Computing*, vol. 17, no. 6, pp. 3999–4012, 2024.

[60] Y. Chiang, Y. Zhang, H. Luo, T.-Y. Chen, G.-H. Chen, H.-T. Chen, Y.-J. Wang, H.-Y. Wei, and C.-T. Chou, "Management and orchestration of edge computing for

iot: A comprehensive survey," *IEEE Internet of Things Journal*, vol. 10, no. 16, pp. 14 307–14 331, 2023.

[61] X. Wang, J. Li, Z. Ning, Q. Song, L. Guo, S. Guo, and M. S. Obaidat, "Wireless powered mobile edge computing networks: A survey," *ACM Computing Surveys*, vol. 55, no. 13s, pp. 1–37, 2023.

[62] S. Dong, J. Tang, K. Abbas, R. Hou, J. Kamruzzaman, L. Rutkowski, and R. Buyya, "Task offloading strategies for mobile edge computing: A survey," *Computer Networks*, vol. 254, p. 110791, 2024.

[63] A. Asghari and M. K. Sohrabi, "Server placement in mobile cloud computing: A comprehensive survey for edge computing, fog computing and cloudlet," *Computer Science Review*, vol. 51, p. 100616, 2024.

[64] A. Maia, A. Boutouchent, Y. Kardjadja, M. Gherari, E. G. Soyak, M. Saqib, K. Boussekar, I. Cilbir, S. Habibi, S. O. Ali *et al.*, "A survey on integrated computing, caching, and communication in the cloud-to-edge continuum," *Computer Communications*, vol. 219, pp. 128–152, 2024.

[65] M. Zolghadri, P. Asghari, S. E. Dashti, and A. Hedayati, "Resource allocation in fog–cloud environments: State of the art," *Journal of Network and Computer Applications*, vol. 227, p. 103891, 2024.

[66] D. Vetriveeran and L. S. R, "Resource provisioning in fog computing - a survey," *ACM Computing Surveys*, vol. 57, no. 12, 2025.

[67] Y. Wu, S. Tang, C. Yu, B. Yang, C. Sun, J. Xiao, H. Wu, and J. Feng, "Task scheduling in geo-distributed computing: A survey," *IEEE Transactions on Parallel and Distributed Systems*, vol. 36, no. 10, pp. 2073–2088, 2025.

[68] A. Aljohani, O. Rana, and C. Perera, "Self-adaptive federated learning in internet of things systems: A review," *ACM Computing Surveys*, vol. 57, no. 10, pp. 1–36, 2025.

[69] S. Yin, H. Jiang, C. Tang, S. Xiao, and H. Wu, "Mobitask: A federated learning-based task migration strategy for mobile crowdsensing," *IEEE Transactions on Computational Social Systems*, pp. 1–15, 2025.

[70] X. Li, T. Jing, R. Li, X. Wang, Y. Yan, X. Fan, Y. Huo, and F. R. Yu, "Self-learning based dependable offloading optimization in semi-trusted vehicular edge computing and networks," *IEEE Transactions on Vehicular Technology*, vol. 74, no. 5, pp. 8141–8157, 2025.

[71] F. E. Subhan, A. Yaqoob, C. H. Muntean, and G.-M. Muntean, "Edge360: Edge-enabled multi-agent drl for region-aware rate adaptation solution to enhance quality of 360 video streaming," *IEEE Transactions on Mobile Computing*, pp. 1–18, 2025.

[72] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[73] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 30, no. 1, 2016.

[74] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, "Dueling network architectures for deep reinforcement learning," in *Proceedings of the International conference on machine learning*. PMLR, 2016, pp. 1995–2003.

[75] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, "Rainbow: Combining improvements in deep reinforcement learning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, no. 1, 2018.

[76] X. Xiong, K. Zheng, L. Lei, and L. Hou, "Resource allocation based on deep reinforcement learning in iot edge computing," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 6, pp. 1133–1146, 2020.

[77] L. Liao, Y. Lai, F. Yang, and W. Zeng, "Online computation offloading with double reinforcement learning algorithm in mobile edge computing," *Journal of Parallel and Distributed Computing*, vol. 171, pp. 28–39, 2023.

[78] S. Birhanu Tadele, B. Kar, F. Guteta Wakgra, and A. Uddin Khan, "Optimization of end-to-end aoi in edge-enabled vehicular fog systems: A dueling-dqn approach," *IEEE Internet of Things Journal*, vol. 12, no. 1, pp. 843–853, 2025.

[79] P. Zhang, S. Li, L. Tan, N. Kumar, J. Wang, and K. Liu, "In-situ data scheduling optimization based on rainbow dqn for iiot," *Future Generation Computer Systems*, vol. 174, p. 107940, 2026.

[80] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," in *Proceedings of the International Conference on Learning Representations*, 2016.

[81] Priyadarshni, D. Kadavala, S. Tripathi, P. Kumar, and R. Misra, "Deep meta reinforcement learning for efficient task offloading in edge computing environments," *Cluster Computing*, vol. 28, no. 11, p. 712, 2025.

[82] J. Wang, J. Hu, G. Min, A. Y. Zomaya, and N. Georgalas, "Fast adaptive task offloading in edge computing based on meta reinforcement learning," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 1, pp. 242–253, 2021.

[83] S. Fujimoto, H. Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *Proceedings of the International conference on machine learning*. PMLR, 2018, pp. 1587–1596.

[84] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *Proceedings of the International conference on machine learning*. PMLR, 2018, pp. 1861–1870.

[85] L. Ale, S. A. King, N. Zhang, A. R. Sattar, and J. Skandaraniyam, "D3pg: Dirichlet ddpg for task partitioning and offloading with constrained hybrid action space in mobile-edge computing," *IEEE Internet of Things Journal*, vol. 9, no. 19, pp. 19 260–19 272, 2022.

[86] C. Li, S. Zhang, Y. Wu, K. Jiang, W. Wu, S. Yuan, and S. Wan, "Improved td3 based resource allocation optimization for latency-sensitive tasks in isac-aided vec," *ACM Transactions on Sensor Networks*, vol. 21, no. 6, pp. 1–27, 2025.

[87] Q. Tang, R. Xie, F. R. Yu, T. Chen, R. Zhang, T. Huang, and Y. Liu, "Collective deep reinforcement learning for intelligence sharing in the internet of intelligence-empowered edge computing," *IEEE Transactions on Mobile Computing*, vol. 22, no. 11, pp. 6327–6342, 2022.

[88] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *Proceedings of the International conference on machine learning*. PmLR, 2016, pp. 1928–1937.

[89] J. Zou, T. Hao, C. Yu, and H. Jin, "A3c-do: A regional resource scheduling framework based on deep reinforcement learning in edge scenario," *IEEE Transactions on Computers*, vol. 70, no. 2, pp. 228–239, 2021.

[90] J. Lu, J. Yang, S. Li, Y. Li, W. Jiang, J. Dai, and J. Hu, "A2c-drl: Dynamic scheduling for stochastic edgecloud environments using a2c and deep reinforcement learning," *IEEE Internet of Things Journal*, vol. 11, no. 9, pp. 16 915–16 927, 2024.

[91] Y. Lin, S. Han, H. Mao, Y. Wang, and B. Dally, "Deep gradient compression: Reducing the communication bandwidth for distributed training," in *Proceedings of the International Conference on Learning Representations*, 2018.

[92] D. Horgan, J. Quan, D. Budden, G. Barth-Maron, M. Hessel, H. van Hasselt, and D. Silver, "Distributed prioritized experience replay," in *Proceedings of the International Conference on Learning Representations*, 2018.

[93] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning *et al.*, "Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures," in *Proceedings of the International conference on machine learning*. PMLR, 2018, pp. 1407–1416.

[94] S. Kapturowski, G. Ostrovski, J. Quan, R. Munos, and W. Dabney, "Recurrent experience replay in distributed reinforcement learning," in *Proceedings of the International conference on learning representations*, 2018.

[95] X. Zhang, D. Biagioni, M. Cai, P. Graf, and S. Rahman, "An edge-cloud integrated solution for buildings demand response using reinforcement learning," *IEEE Transactions on Smart Grid*, vol. 12, no. 1, pp. 420–431, 2020.

[96] L. Wang, Y. Xu, H. Xu, M. Chen, and L. Huang, "Accelerating decentralized federated learning in heterogeneous edge computing," *IEEE Transactions on Mobile Computing*, vol. 22, no. 9, pp. 5001–5016, 2022.

[97] A. Tampuu, T. Matiisen, D. Kodelja, I. Kuzovkin, K. Korjus, J. Aru, J. Aru, and R. Vicente, "Multiagent cooperation and competition with deep reinforcement learning," *PloS one*, vol. 12, no. 4, p. e0172395, 2017.

[98] J. Cui, Y. Liu, and A. Nallanathan, "Multi-agent reinforcement learning-based resource allocation for uav networks," *IEEE Transactions on Wireless Communications*, vol. 19, no. 2, pp. 729–743, 2020.

[99] N. Waqar, S. A. Hassan, H. Pervaiz, H. Jung, and K. Dev, "Deep multi-agent reinforcement learning for resource allocation in noma-enabled mec," *Computer Communications*, vol. 196, pp. 1–8, 2022.

[100] S. M. Shahid, J.-H. Na, and S. Kwon, "Mobility robustness optimization for dynamic cell on/off small-cell networks: Independent dqn-based multi-agent reinforcement learning approach," *IEEE Transactions on Vehicular Technology*, 2025.

[101] M. Landers and A. Doryab, "Deep reinforcement learning verification: A survey," *ACM Computing Surveys*, vol. 55, no. 14s, pp. 1–31, 2023.

[102] C. Yu, A. Velu, E. Vinitsky, J. Gao, Y. Wang, A. Bayen, and Y. Wu, "The surprising effectiveness of ppo in cooperative multi-agent games," *Advances in neural information processing systems*, vol. 35, pp. 24 611–24 624, 2022.

[103] Y. Lin, L. Xiao, Y. Tao, Y. Zhang, F. Shu, and J. Li, "Multi-agent computing-energy-efficiency optimization in vehicular edge computing: Non-cooperative versus cooperative solutions," *IEEE Transactions on Wireless Communications*, vol. 24, no. 7, pp. 5461–5476, 2025.

[104] P. Liu, D. Zhang, F. Wang, S. Shi, Y. Zhang, and Y. Li, "Priority-aware resource allocation in aoi-oriented ul-ofdma wi-fi networks based on multiagent reinforcement learning," *IEEE Internet of Things Journal*, vol. 12, no. 20, pp. 42 045–42 058, 2025.

[105] R. Lowe, Y. I. Wu, A. Tamar, J. Harb, O. Pieter Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," *Advances in neural information processing systems*, vol. 30, 2017.

[106] G. Papoudakis, F. Christianos, L. Schäfer, and S. V. Albrecht, "Benchmarking multi-agent deep reinforcement learning algorithms in cooperative tasks," in *Proceedings of the Conference on Neural Information Processing Systems*, 2021.

[107] K. Zheng, R. Luo, X. Liu, J. Qiu, and J. Liu, "Distributed ddpg-based resource allocation for age of information minimization in mobile wireless-powered internet of things," *IEEE Internet of Things Journal*, vol. 11, no. 17, pp. 29 102–29 115, 2024.

[108] P. Hernandez-Leal, B. Kartal, and M. E. Taylor, "A survey and critique of multiagent deep reinforcement learning," *Autonomous Agents and Multi-Agent Systems*, vol. 33, no. 6, pp. 750–797, 2019.

[109] P. Sunehag, G. Lever, A. Gruslys, W. M. Czarnecki, V. Zambaldi, M. Jaderberg, M. Lanctot, N. Sonnerat, J. Z. Leibo, K. Tuyls, and T. Graepel, "Value-decomposition networks for cooperative multi-agent learning based on team reward," in *Proceedings of the International Conference on Autonomous Agents and MultiAgent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 2018, p. 20852087.

[110] T. Rashid, M. Samvelyan, C. S. De Witt, G. Farquhar, J. Foerster, and S. White-

son, "Monotonic value function factorisation for deep multi-agent reinforcement learning," *Journal of Machine Learning Research*, vol. 21, no. 178, pp. 1–51, 2020.

[111] A. M. Raivi and S. Moh, "Jdaco: Joint data aggregation and computation offloading in uav-enabled internet of things for post-disaster scenarios," *IEEE Internet of Things Journal*, vol. 11, no. 9, pp. 16 529–16 544, 2024.

[112] H. Yu, S. Leng, and F. Wu, "Joint cooperative computation offloading and trajectory optimization in heterogeneous uav-swarm-enabled aerial edge computing networks," *IEEE Internet of Things Journal*, vol. 11, no. 10, pp. 17 700–17 711, 2024.

[113] K. Son, D. Kim, W. J. Kang, D. E. Hostallero, and Y. Yi, "Qtran: Learning to factorize with transformation for cooperative multi-agent reinforcement learning," in *Proceedings of the International conference on machine learning*. PMLR, 2019, pp. 5887–5896.

[114] J. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson, "Counterfactual multi-agent policy gradients," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, no. 1, 2018.

[115] S. Yao, M. Wang, J. Ren, T. Xia, W. Wang, K. Xu, M. Xu, and H. Zhang, "Multi-agent reinforcement learning for task offloading in crowd-edge computing," *IEEE Transactions on Mobile Computing*, vol. 24, no. 10, pp. 9289–9302, 2025.

[116] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *ACM Transactions on Intelligent Systems and Technology*, vol. 10, no. 2, pp. 1–19, 2019.

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

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

[119] J. Wu, F. Dong, H. Leung, Z. Zhu, J. Zhou, and S. Drew, "Topology-aware federated learning in edge computing: A comprehensive survey," *ACM Computing Surveys*, vol. 56, no. 10, pp. 1–41, 2024.

[120] S. J. Reddi, Z. Charles, M. Zaheer, Z. Garrett, K. Rush, J. Konečný, S. Kumar, and H. B. McMahan, "Adaptive federated optimization," in *Proceedings of the International Conference on Learning Representations*, 2021.

[121] T. Yang, S. Cen, Y. Wei, Y. Chen, and Y. Chi, "Federated natural policy gradient and actor critic methods for multi-task reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 37, pp. 121 304–121 375, 2024.

[122] M. Zhao and M. R. Nakhai, "A unified federated deep q learning caching scheme for scalable collaborative edge networks," *IEEE Transactions on Mobile Computing*, vol. 23, no. 12, pp. 10 855–10 866, 2024.

[123] J. Woo, G. Joshi, and Y. Chi, "The blessing of heterogeneity in federated q-learning: Linear speedup and beyond," *Journal of Machine Learning Research*, vol. 26, no. 26, pp. 1–85, 2025.

[124] S. Shen, G. Shen, Z. Dai, K. Zhang, X. Kong, and J. Li, "Asynchronous federated deep-reinforcement-learning-based dependency task offloading for uav-assisted vehicular networks," *IEEE Internet of Things Journal*, vol. 11, no. 19, pp. 31 561–31 574, 2024.

[125] H. Zhou, H. Wang, Z. Yu, G. Bin, M. Xiao, and J. Wu, "Federated distributed deep reinforcement learning for recommendation-enabled edge caching," *IEEE Transactions on Services Computing*, vol. 17, no. 6, pp. 3640–3656, 2024.

[126] J. Zheng, K. Li, N. Mhaisen, W. Ni, E. Tovar, and M. Guizani, "Exploring deep-reinforcement-learning-assisted federated learning for online resource allocation in privacy-preserving edgeiot," *IEEE Internet of Things Journal*, vol. 9, no. 21, pp. 21 099–21 110, 2022.

[127] H. Huang, Z. Duan, W. Zhan, G. Min, Z. Wang, and Y. Lei, "Federated reinforcement learning for distributed dependent task offloading with batch processing in

cpu-gpu heterogeneous mec," *IEEE Transactions on Vehicular Technology*, pp. 1–16, 2025.

[128] D. Rengarajan, N. Ragothaman, D. Kalathil, and S. Shakkottai, "Federated ensemble-directed offline reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 37, pp. 6154–6179, 2024.

[129] S. Lei, H. Tang, C. Li, X. Zhang, C. Xu, and H. Wu, "Federated maddpg-based collaborative scheduling strategy in vehicular edge computing," *IEEE Transactions on Mobile Computing*, vol. 25, no. 1, pp. 54–66, 2026.

[130] M. Zeng, Y. Zhao, J. Wang, and Z. Fei, "Multiple reconfigurable intelligent surfaces aided v2x offloading networks: A federated reinforcement learning-based approach," *IEEE Transactions on Vehicular Technology*, pp. 1–14, 2025.

[131] X. Lyu, A. Baisero, Y. Xiao, B. Daley, and C. Amato, "On centralized critics in multi-agent reinforcement learning," *Journal of Artificial Intelligence Research*, vol. 77, pp. 295–354, 2023.

[132] A. Hashemi, A. Acharya, R. Das, H. Vikalo, S. Sanghavi, and I. Dhillon, "On the benefits of multiple gossip steps in communication-constrained decentralized federated learning," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 11, pp. 2727–2739, 2021.

[133] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah, "Randomized gossip algorithms," *IEEE transactions on information theory*, vol. 52, no. 6, pp. 2508–2530, 2006.

[134] Y. Qu, M. P. Uddin, C. Gan, Y. Xiang, L. Gao, and J. Yearwood, "Blockchain-enabled federated learning: A survey," *ACM Computing Surveys*, vol. 55, no. 4, pp. 1–35, 2022.

[135] G. Qu and N. Li, "Harnessing smoothness to accelerate distributed optimization," *IEEE Transactions on Control of Network Systems*, vol. 5, no. 3, pp. 1245–1260, 2017.

[136] L. Kong, T. Lin, A. Koloskova, M. Jaggi, and S. Stich, "Consensus control for decentralized deep learning," in *Proceedings of the International Conference on Machine Learning*. PMLR, 2021, pp. 5686–5696.

[137] J. Jeong and J. Lee, "A federated reinforcement learning framework via a committee mechanism for resource management in 5g networks," *Sensors*, vol. 24, no. 21, p. 7031, 2024.

[138] J. Chai, Z. Wang, C. Ma, G. Gao, and L. Shi, "Personalized federated reinforcement learning for multi-aav assisted edge computing," *IEEE Wireless Communications Letters*, vol. 14, no. 7, pp. 2074–2078, 2025.

[139] M. Assran, J. Romoff, N. Ballas, J. Pineau, and M. Rabbat, "Gossip-based actor-learner architectures for deep reinforcement learning," *Advances in neural information processing systems*, vol. 32, 2019.

[140] Y. Fan, J. Ge, S. Zhang, J. Wu, and B. Luo, "Decentralized scheduling for concurrent tasks in mobile edge computing via deep reinforcement learning," *IEEE Transactions on Mobile Computing*, vol. 23, no. 4, pp. 2765–2779, 2024.

[141] S. Agarwal, M. A. Rodriguez, and R. Buyya, "A deep recurrent-reinforcement learning method for intelligent autoscaling of serverless functions," *IEEE Transactions on Services Computing*, vol. 17, no. 5, pp. 1899–1910, 2024.

[142] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*. MIT press Cambridge, 2016, vol. 1, no. 2.

[143] Y. Gal and Z. Ghahramani, "A theoretically grounded application of dropout in recurrent neural networks," *Advances in neural information processing systems*, vol. 29, 2016.

[144] Y. Tay, M. Dehghani, D. Bahri, and D. Metzler, "Efficient transformers: A survey," *ACM Computing Surveys*, vol. 55, no. 6, 2022.

[145] G. Wang, P. Cheng, Z. Chen, B. Vucetic, and Y. Li, "Inverse reinforcement learning with graph neural networks for full-dimensional task offloading in edge computing," *IEEE Transactions on Mobile Computing*, vol. 23, no. 6, pp. 6490–6507, 2024.

[146] S. Abadal, A. Jain, R. Guirado, J. López-Alonso, and E. Alarcón, "Computing graph neural networks: A survey from algorithms to accelerators," *ACM Computing Surveys*, vol. 54, no. 9, pp. 1–38, 2021.

[147] P. Dai, Y. Huang, K. Hu, X. Wu, H. Xing, and Z. Yu, "Meta reinforcement learning for multi-task offloading in vehicular edge computing," *IEEE Transactions on Mobile Computing*, vol. 23, no. 3, pp. 2123–2138, 2024.

[148] N. Sharma, A. Ghosh, R. Misra, and S. K. Das, "Deep meta q-learning based multi-task offloading in edge-cloud systems," *IEEE Transactions on Mobile Computing*, vol. 23, no. 4, pp. 2583–2598, 2024.

[149] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *Proceedings of the International conference on machine learning*. PMLR, 2017, pp. 1126–1135.

[150] H. Gharoun, F. Momenifar, F. Chen, and A. H. Gandomi, "Meta-learning approaches for few-shot learning: A survey of recent advances," *ACM Computing Surveys*, vol. 56, no. 12, pp. 1–41, 2024.

[151] F. Niknia and P. Wang, "Edge caching optimization with ppo and transfer learning for dynamic environments," *IEEE Internet of Things Journal*, vol. 12, no. 16, pp. 33 605–33 620, 2025.

[152] Y. Peng, X. Tang, Y. Zhou, J. Li, Y. Qi, L. Liu, and H. Lin, "Computing and communication cost-aware service migration enabled by transfer reinforcement learning for dynamic vehicular edge computing networks," *IEEE Transactions on Mobile Computing*, vol. 23, no. 1, pp. 257–269, 2024.

[153] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, and Q. He, "A comprehensive survey on transfer learning," *Proceedings of the IEEE*, vol. 109, no. 1, pp. 43–76, 2020.

[154] S. Chen, Y. Zhang, and Q. Yang, "Multi-task learning in natural language processing: An overview," *ACM Computing Surveys*, vol. 56, no. 12, pp. 1–32, 2024.

[155] T. Zeng, X. Zhang, J. Duan, C. Yu, C. Wu, and X. Chen, "An offline-transfer-online framework for cloud-edge collaborative distributed reinforcement learning," *IEEE Transactions on Parallel and Distributed Systems*, vol. 35, no. 5, pp. 720–731, 2024.

[156] Z. Zhang, Y. Zhao, H. Li, C. Lin, and J. Liu, "Dvfo: Learning-based dvfs for energy-efficient edge-cloud collaborative inference," *IEEE Transactions on Mobile Computing*, vol. 23, no. 10, pp. 9042–9059, 2024.

[157] C. Zhang, W. Zhang, Q. Wu, P. Fan, Q. Fan, J. Wang, and K. B. Letaief, "Distributed deep reinforcement learning-based gradient quantization for federated learning enabled vehicle edge computing," *IEEE Internet of Things Journal*, vol. 12, no. 5, pp. 4899–4913, 2025.

[158] G. Menghani, "Efficient deep learning: A survey on making deep learning models smaller, faster, and better," *ACM Computing Surveys*, vol. 55, no. 12, pp. 1–37, 2023.

[159] F. Jiang, Y. Peng, K. Wang, L. Dong, and K. Yang, "Mars: A drl-based multi-task resource scheduling framework for uav with irs-assisted mobile edge computing system," *IEEE Transactions on Cloud Computing*, vol. 11, no. 4, pp. 3700–3712, 2023.

[160] V. Niazmand and Q. Ye, "Joint task offloading, dnn pruning, and computing resource allocation for fault detection with dynamic constraints in industrial iot," *IEEE Transactions on Cognitive Communications and Networking*, vol. 11, no. 5, pp. 3486–3501, 2025.

[161] X. Hao, P. L. Yeoh, C. She, B. Vucetic, and Y. Li, "Secure deep reinforcement learning for dynamic resource allocation in wireless mec networks," *IEEE Transactions on Communications*, vol. 72, no. 3, pp. 1414–1427, 2024.

[162] J. Garcıa and F. Fernández, "A comprehensive survey on safe reinforcement learning," *Journal of Machine Learning Research*, vol. 16, no. 1, pp. 1437–1480, 2015.

[163] Y. Zhang, J. Hu, G. Min, X. Chen, and N. Georgalas, "Joint charging scheduling and computation offloading in ev-assisted edge computing: A safe drl approach," *IEEE Transactions on Mobile Computing*, vol. 23, no. 9, pp. 8757–8772, 2024.

[164] X. Lu, L. Xiao, Y. Xiao, Z. Xiong, Z. Liu, Y. Zhang, and W. Zhuang, "Blockchain-enabled secure offloading for vec: A multi-agent reinforcement learning approach," *IEEE Transactions on Dependable and Secure Computing*, vol. 22, no. 3, pp. 2978–2995, 2025.

[165] J. Xue, J. Yao, and J. Wang, "A dynamic pricing scheme for secure offloading and resource allocation based on the internet of vehicles," *Ad Hoc Networks*, vol. 161, p. 103545, 2024.

[166] J. Zhang, X. Cheng, L. Yang, J. Hu, X. Liu, and K. Chen, "Sok: Fully homomorphic encryption accelerators," *ACM Computing Surveys*, vol. 56, no. 12, pp. 1–32, 2024.

[167] B. C. Das, M. H. Amini, and Y. Wu, "Security and privacy challenges of large language models: A survey," *ACM Computing Surveys*, vol. 57, no. 6, pp. 1–39, 2025.

[168] G. S. S. Chalapathi, V. Chamola, A. Vaish, and R. Buyya, "Industrial internet of things (iiot) applications of edge and fog computing: A review and future directions," *Fog/Edge Computing for Security, Privacy, and Applications*, pp. 293–325, 2021.

[169] S. Azizi, M. Shojafar, J. Abawajy, and R. Buyya, "Deadline-aware and energy-efficient iot task scheduling in fog computing systems: A semi-greedy approach," *Journal of Network and Computer Applications*, vol. 201, p. 103333, 2022.

[170] A. J. Ferrer, J. M. Marquès, and J. Jorba, "Towards the decentralised cloud: Survey on approaches and challenges for mobile, ad hoc, and edge computing," *ACM Computing Surveys (CSUR)*, vol. 51, no. 6, pp. 1–36, 2019.

[171] M. Goudarzi, H. Wu, M. Palaniswami, and R. Buyya, "An application placement technique for concurrent iot applications in edge and fog computing environments," *IEEE Transactions on Mobile Computing*, vol. 20, no. 4, pp. 1298–1311, 2020.

[172] L. Catarinucci, D. de Donno, L. Mainetti, L. Palano, L. Patrono, M. L. Stefanizzi, and L. Tarricone, "An iot-aware architecture for smart healthcare systems," *IEEE Internet of Things Journal*, vol. 2, no. 6, pp. 515–526, 2015.

[173] L. Liu, S. Lu, R. Zhong, B. Wu, Y. Yao, Q. Zhang, and W. Shi, "Computing systems for autonomous driving: State of the art and challenges," *IEEE Internet of Things Journal*, vol. 8, no. 8, pp. 6469–6486, 2021.

[174] A. Brogi and S. Forti, "Qos-aware deployment of iot applications through the fog," *IEEE internet of Things Journal*, vol. 4, no. 5, pp. 1185–1192, 2017.

[175] X. Ma, H. Gao, H. Xu, and M. Bian, "An iot-based task scheduling optimization scheme considering the deadline and cost-aware scientific workflow for cloud computing," *EURASIP Journal on Wireless Communications and Networking*, vol. 2019, no. 1, pp. 1–19, 2019.

[176] Z. Wang, M. Goudarzi, J. Aryal, and R. Buyya, "Container orchestration in edge and fog computing environments for real-time iot applications," in *Proceedings of the Computational Intelligence and Data Analytics (ICCIDA)*. Springer, 2022, pp. 1–21.

[177] E. Li, Z. Zhou, and X. Chen, "Edge intelligence: On-demand deep learning model co-inference with device-edge synergy," in *Proceedings of the 2018 Workshop on Mobile Edge Communications*, 2018, pp. 31–36.

[178] M. Goudarzi, Q. Deng, and R. Buyya, "Resource management in edge and fog computing using fogbus2 framework," *arXiv preprint arXiv:2108.00591*, 2021.

[179] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[180] K. Deb and H. Jain, "An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: solving problems with box constraints," *IEEE Transactions on Evolutionary Computation*, vol. 18, no. 4, pp. 577–601, 2013.

[181] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, pp. 279–292, 1992.

[182] J. Liu, Y. Mao, J. Zhang, and K. B. Letaief, "Delay-optimal computation task scheduling for mobile-edge computing systems," in *Proceedings of the IEEE International Symposium on Information Theory (ISIT)*, 2016, pp. 1451–1455.

[183] C. Wu, W. Li, L. Wang, and A. Y. Zomaya, "Hybrid evolutionary scheduling for energy-efficient fog-enhanced internet of things," *IEEE Transactions on Cloud Computing*, vol. 9, no. 2, pp. 641–653, 2021.

[184] Y. Sun, F. Lin, and H. Xu, "Multi-objective optimization of resource scheduling in fog computing using an improved nsga-ii," *Wireless Personal Communications*, vol. 102, no. 2, pp. 1369–1385, 2018.

[185] F. Hoseiny, S. Azizi, M. Shojafar, F. Ahmadiazar, and R. Tafazolli, "Pga: a priority-aware genetic algorithm for task scheduling in heterogeneous fog-cloud computing," in *Proceedings of the IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2021, pp. 1–6.

[186] I. M. Ali, K. M. Sallam, N. Moustafa, R. Chakraborty, M. Ryan, and K.-K. R. Choo, "An automated task scheduling model using non-dominated sorting genetic algorithm ii for fog-cloud systems," *IEEE Transactions on Cloud Computing*, vol. 10, no. 4, pp. 2294–2308, 2022.

[187] F. Ramezani Shahidani, A. Ghasemi, A. Toroghi Haghighat, and A. Keshavarzi, "Task scheduling in edge-fog-cloud architecture: a multi-objective load balancing approach using reinforcement learning algorithm," *Computing*, pp. 1–23, 2023.

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

[189] X. Jie, T. Liu, H. Gao, C. Cao, P. Wang, and W. Tong, "A dqn-based approach for online service placement in mobile edge computing," in *Proceedings of the 16th EAI International Conference on Collaborative Computing: Networking, Applications and Worksharing*. Springer, 2021, pp. 169–183.

[190] J. Wang, L. Zhao, J. Liu, and N. Kato, "Smart resource allocation for mobile edge computing: A deep reinforcement learning approach," *IEEE Transactions on Emerging Topics in Computing*, vol. 9, no. 3, pp. 1529–1541, 2021.

[191] L. Huang, X. Feng, C. Zhang, L. Qian, and Y. Wu, "Deep reinforcement learning-based joint task offloading and bandwidth allocation for multi-user mobile edge computing," *Digital Communications and Networks*, vol. 5, no. 1, pp. 10–17, 2019.

[192] X. Chen, H. Zhang, C. Wu, S. Mao, Y. Ji, and M. Bennis, "Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4005–4018, 2019.

[193] Y. Zheng, H. Zhou, R. Chen, K. Jiang, and Y. Cao, "Sac-based computation offloading and resource allocation in vehicular edge computing," in *Proceedings of the IEEE Conference on Computer Communications Workshops*. IEEE, 2022, pp. 1–6.

[194] T. Zhao, F. Li, and L. He, "Secure video offloading in mec-enabled iiot networks: A multi-cell federated deep reinforcement learning approach," *IEEE Transactions on Industrial Informatics*, pp. 1–12, 2023.

[195] V. Sethi and S. Pal, "Feddove: A federated deep q-learning-based offloading for vehicular fog computing," *Future Generation Computer Systems*, vol. 141, pp. 96–105, 2023.

[196] P. Li, W. Xie, Y. Yuan, C. Chen, and S. Wan, "Deep reinforcement learning for load balancing of edge servers in iov," *Mobile Networks and Applications*, vol. 27, no. 4, pp. 1461–1474, 2022.

[197] X. Chu, M. Zhu, H. Mao, and Y. Qiu, "Task offloading for multi-gateway-assisted mobile edge computing based on deep reinforcement learning," in *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2022, pp. 3234–3241.

[198] F. Xue, Q. Hai, T. Dong, Z. Cui, and Y. Gong, "A deep reinforcement learning based hybrid algorithm for efficient resource scheduling in edge computing environment," *Information Sciences*, vol. 608, pp. 362–374, 2022.

[199] S. Pallewatta, V. Kostakos, and R. Buyya, "Placement of microservices-based iot applications in fog computing: A taxonomy and future directions," *ACM Computing Surveys*, vol. 55, pp. 1–43, 2023.

[200] W. Zhu, M. Goudarzi, and R. Buyya, "Flight: A lightweight federated learning framework in edge and fog computing," *arXiv preprint arXiv:2308.02834*, 2023.

[201] X. Qiu, W. Zhang, W. Chen, and Z. Zheng, "Distributed and collective deep reinforcement learning for computation offloading: A practical perspective," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 5, pp. 1085–1101, 2020.

[202] N. H. Tran, W. Bao, A. Zomaya, M. N. Nguyen, and C. S. Hong, "Federated learning over wireless networks: Optimization model design and analysis," in *Proceedings of the IEEE INFOCOM*. IEEE, 2019, pp. 1387–1395.

[203] J. Ji, K. Zhu, and L. Cai, "Trajectory and communication design for cache-enabled uavs in cellular networks: A deep reinforcement learning approach," *IEEE Transactions on Mobile Computing*, 2022.

[204] S. Mysore, B. E. Mabsout, R. Mancuso, and K. Saenko, "Honey. i shrunk the actor: A case study on preserving performance with smaller actors in actor-critic rl," in *2021 IEEE Conference on Games (CoG)*, 2021, pp. 01–08.

[205] P. Christodoulou, "Soft actor-critic for discrete action settings," *arXiv preprint arXiv:1910.07207*, 2019.

[206] H. Wang, Y. Ye, J. Zhang, and B. Xu, "A comparative study of 13 deep reinforcement learning based energy management methods for a hybrid electric vehicle," *Energy*, vol. 266, p. 126497, 2023.

[207] J. Zhu, F. Wu, and J. Zhao, "An overview of the action space for deep reinforcement learning," in *Proceedings of the 4th International Conference on Algorithms, Computing and Artificial Intelligence*, 2021, pp. 1–10.

[208] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," *Advances in Neural Information Processing Systems*, vol. 12, 1999.

[209] R. Huang, T. Yu, Z. Ding, and S. Zhang, "Policy gradient," *Deep Reinforcement Learning: Fundamentals, Research and Applications*, pp. 161–212, 2020.

[210] T. Van Erven and P. Harremos, "Rényi divergence and kullback-leibler divergence," *IEEE Transactions on Information Theory*, vol. 60, no. 7, pp. 3797–3820, 2014.

[211] S. Shao and W. Luk, "Customised pearlmutter propagation: A hardware architecture for trust region policy optimisation," in *Proceedings of the 27th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2017, pp. 1–6.

[212] S. Li, R. Wang, M. Tang, and C. Zhang, "Hierarchical reinforcement learning with advantage-based auxiliary rewards," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[213] S. Aljanabi and A. Chalechale, "Improving iot services using a hybrid fog-cloud offloading," *IEEE Access*, vol. 9, pp. 13 775–13 788, 2021.

[214] L. Yliniemi and K. Tumer, "Multi-objective multiagent credit assignment in reinforcement learning and nsga-ii," *Soft Computing*, vol. 20, no. 10, pp. 3869–3887, 2016.

[215] J. Blank and K. Deb, "Pymoo: Multi-objective optimization in python," *IEEE Access*, vol. 8, pp. 89 497–89 509, 2020.

[216] X. Li, X. Li, K. Wang, S. Yang, and Y. Li, "Achievement scalarizing function sorting for strength pareto evolutionary algorithm in many-objective optimization," *Neural Computing and Applications*, vol. 33, pp. 6369–6388, 2021.

[217] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.

[218] D. Wei, N. Xi, X. Ma, M. Shojafar, S. Kumari, and J. Ma, "Personalized privacy-aware task offloading for edge-cloud-assisted industrial internet of things in automated manufacturing," *IEEE Transactions on Industrial Informatics*, vol. 18, no. 11, pp. 7935–7945, 2022.

[219] N. Bjorck, C. P. Gomes, B. Selman, and K. Q. Weinberger, "Understanding batch normalization," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[220] C. Huang, R. Mo, and C. Yuen, "Reconfigurable intelligent surface assisted multiuser miso systems exploiting deep reinforcement learning," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 8, pp. 1839–1850, 2020.

[221] F. Fu, Y. Kang, Z. Zhang, and F. R. Yu, "Transcoding for live streaming-based on vehicular fog computing: An actor-critic drl approach," in *Proceedings of the IEEE INFOCOM - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2020, pp. 1015–1020.

[222] R. Islam, P. Henderson, M. Gomrokchi, and D. Precup, "Reproducibility of benchmarked deep reinforcement learning tasks for continuous control," *arXiv preprint arXiv:1708.04133*, 2017.

[223] F. Al-Doghman, N. Moustafa, I. Khalil, Z. Tari, and A. Zomaya, "Ai-enabled secure microservices in edge computing: Opportunities and challenges," *IEEE Transactions on Services Computing*, 2022.

[224] X. Su, L. An, Z. Cheng, and Y. Weng, "Cloud–edge collaboration-based bi-level optimal scheduling for intelligent healthcare systems," *Future Generation Computer Systems*, vol. 141, pp. 28–39, 2023.

[225] W. Wen, U. Demirbaga, A. Singh, A. Jindal, R. S. Batth, P. Zhang, and G. S. Aujla, "Health monitoring and diagnosis for geo-distributed edge ecosystem in smart city," *IEEE Internet of Things Journal*, vol. 10, no. 21, pp. 18 571–18 578, 2023.

[226] X. Dai, Z. Xiao, H. Jiang, M. Alazab, J. C. S. Lui, S. Dustdar, and J. Liu, "Task co-offloading for d2d-assisted mobile edge computing in industrial internet of things," *IEEE Transactions on Industrial Informatics*, vol. 19, no. 1, pp. 480–490, 2023.

[227] H. Ma, R. Li, X. Zhang, Z. Zhou, and X. Chen, "Reliability-aware online scheduling for dnn inference tasks in mobile edge computing," *IEEE Internet of Things Journal*, 2023.

[228] X. Wang, S. Wang, X. Liang, D. Zhao, J. Huang, X. Xu, B. Dai, and Q. Miao, "Deep reinforcement learning: A survey," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–15, 2022.

[229] E. Parisotto, F. Song, J. Rae, R. Pascanu, C. Gulcehre, S. Jayakumar, M. Jaderberg, R. L. Kaufman, A. Clark, S. Noury *et al.*, "Stabilizing transformers for reinforce-

ment learning," in *International conference on machine learning*. PMLR, 2020, pp. 7487–7498.

[230] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *arXiv preprint arXiv:1511.05952*, 2015.

[231] M. Bansal, I. Chana, and S. Clarke, "Urbanenqosplace: A deep reinforcement learning model for service placement of real-time smart city iot applications," *IEEE Transactions on Services Computing*, vol. 16, no. 4, pp. 3043–3060, 2023.

[232] L. T. Hoang, C. T. Nguyen, and A. T. Pham, "Deep reinforcement learning-based online resource management for uav-assisted edge computing with dual connectivity," *IEEE/ACM Transactions on Networking*, pp. 1–16, 2023.

[233] L. Zhao, E. Zhang, S. Wan, A. Hawbani, A. Y. Al-Dubai, G. Min, and A. Y. Zomaya, "Meson: A mobility-aware dependent task offloading scheme for urban vehicular edge computing," *IEEE Transactions on Mobile Computing*, vol. 23, no. 5, pp. 4259–4272, 2024.

[234] L.-T. Hsieh, H. Liu, Y. Guo, and R. Gazda, "Deep reinforcement learning-based task assignment for cooperative mobile edge computing," *IEEE Transactions on Mobile Computing*, vol. 23, no. 4, pp. 3156–3171, 2024.

[235] X. Chen, S. Hu, C. Yu, Z. Chen, and G. Min, "Real-time offloading for dependent and parallel tasks in cloud-edge environments using deep reinforcement learning," *IEEE Transactions on Parallel and Distributed Systems*, vol. 35, no. 3, pp. 391–404, 2024.

[236] S. Wen, R. Han, C. H. Liu, and L. Y. Chen, "Fast drl-based scheduler configuration tuning for reducing tail latency in edge-cloud jobs," *Journal of Cloud Computing*, vol. 12, no. 1, pp. 1–32, 2023.

[237] Z. Wang, M. Li, L. Zhao, H. Zhou, and N. Wang, "A3c-based computation offloading and service caching in cloud-edge computing networks," in *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2022, pp. 1–2.

[238] R. Garaali, C. Chaieb, W. Ajib, and M. Afif, "Learning-based task offloading for mobile edge computing," in *ICC 2022 - IEEE International Conference on Communications*, 2022, pp. 1659–1664.

[239] Y. Ju, Z. Cao, Y. Chen, L. Liu, Q. Pei, S. Mumtaz, M. Dong, and M. Guizani, "Noma-assisted secure offloading for vehicular edge computing networks with asynchronous deep reinforcement learning," *IEEE Transactions on Intelligent Transportation Systems*, vol. 25, no. 3, pp. 2627–2640, 2024.

[240] B. Sellami, A. Hakiri, and S. B. Yahia, "Deep reinforcement learning for energy-aware task offloading in join sdn-blockchain 5g massive iot edge network," *Future Generation Computer Systems*, vol. 137, pp. 363–379, 2022.

[241] S. Chen, J. Chen, Y. Miao, Q. Wang, and C. Zhao, "Deep reinforcement learning-based cloud-edge collaborative mobile computation offloading in industrial networks," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 8, pp. 364–375, 2022.

[242] J. Hao, T. Yang, H. Tang, C. Bai, J. Liu, Z. Meng, P. Liu, and Z. Wang, "Exploration in deep reinforcement learning: From single-agent to multiagent domain," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–21, 2023.

[243] E. Meskar, T. D. Todd, D. Zhao, and G. Karakostas, "Energy efficient offloading for competing users on a shared communication channel," in *Proceedings of the IEEE International Conference on Communications (ICC)*, 2015, pp. 3192–3197.

[244] X. Long, J. Wu, and L. Chen, "Energy-efficient offloading in mobile edge computing with edge-cloud collaboration," in *Proceedings of International conference on algorithms and architectures for parallel processing*, 2018, pp. 460–475.

[245] Y. Ding, K. Li, C. Liu, Z. Tang, and K. Li, "Budget-constrained service allocation optimization for mobile edge computing," *IEEE Transactions on Services Computing*, 2021.

[246] S. A. Budennyy, V. D. Lazarev, N. N. Zakharenko, A. N. Korovin, O. Plosskaya, D. V. Dimitrov, V. Akhripkin, I. Pavlov, I. V. Oseledets, I. S. Barsola *et al.*, "Eco2ai:

carbon emissions tracking of machine learning models as the first step towards sustainable ai," in *Doklady Mathematics*, 2022, pp. S118–S128.

[247] N. Scarlat, M. Prussi, and M. Padella, "Quantification of the carbon intensity of electricity produced and used in europe," *Applied Energy*, vol. 305, p. 117901, 2022.

[248] L. M. Meruje Ferreira, F. Coelho, and J. Pereira, "Databases in edge and fog environments: A survey," *ACM Computing Surveys*, vol. 56, no. 11, pp. 1–40, 2024.

[249] Y. Mansouri and M. A. Babar, "A review of edge computing: Features and resource virtualization," *Journal of Parallel and Distributed Computing*, vol. 150, pp. 155–183, 2021.

[250] J. Jin, K. Yu, J. Kua, N. Zhang, Z. Pang, and Q.-L. Han, "Cloud-fog automation: Vision, enabling technologies, and future research directions," *IEEE Transactions on Industrial Informatics*, vol. 20, no. 2, pp. 1039–1054, 2023.

[251] R. Buyya, S. N. Srirama, R. Mahmud, M. Goudarzi, L. Ismail, and V. Kostakos, "Quality of service (qos)-driven edge computing and smart hospitals: a vision, architectural elements, and future directions," in *International Conference on Communication, Electronics and Digital Technology*. Springer, 2023, pp. 1–23.

[252] S. Sharif, M. H. Y. Moghaddam, and S. A. H. Seno, "A hybrid bi-level management framework for caching and communication in edge-ai enabled iot," *Journal of Network and Computer Applications*, vol. 232, p. 104000, 2024.

[253] E. S. Ali, R. A. Saeed, I. K. Eltahir, and O. O. Khalifa, "A systematic review on energy efficiency in the internet of underwater things (iout): Recent approaches and research gaps," *Journal of Network and Computer Applications*, vol. 213, p. 103594, 2023.

[254] J. Huang, C. Yang, S. Zhang, F. Yang, O. Alfarraj, V. Frascolla, S. Mumtaz, and K. Yu, "Reinforcement learning based resource management for 6g-enabled miot with hypergraph interference model," *IEEE Transactions on Communications*, vol. 72, no. 7, pp. 4179–4192, 2024.

[255] G. Zhou, R. Wen, W. Tian, and R. Buyya, "Deep reinforcement learning-based algorithms selectors for the resource scheduling in hierarchical cloud computing," *Journal of Network and Computer Applications*, vol. 208, p. 103520, 2022.

[256] R. Siyadatzadeh, F. Mehrafrooz, M. Ansari, B. Safaei, M. Shafique, J. Henkel, and A. Ejlali, "Relief: A reinforcement-learning-based real-time task assignment strategy in emerging fault-tolerant fog computing," *IEEE Internet of Things Journal*, vol. 10, no. 12, pp. 10 752–10 763, 2023.

[257] L. Liu, J. Feng, X. Mu, Q. Pei, D. Lan, and M. Xiao, "Asynchronous deep reinforcement learning for collaborative task computing and on-demand resource allocation in vehicular edge computing," *IEEE Transactions on Intelligent Transportation Systems*, vol. 24, no. 12, pp. 15 513–15 526, 2023.

[258] E. Yigitoglu, M. Mohamed, L. Liu, and H. Ludwig, "Foggy: A framework for continuous automated iot application deployment in fog computing," in *Proceedings of the IEEE International Conference on AI & Mobile Services (AIMS)*. IEEE, 2017, pp. 38–45.

[259] G. Merlino, R. Dautov, S. Distefano, and D. Bruneo, "Enabling workload engineering in edge, fog, and cloud computing through openstack-based middleware," *ACM Transactions on Internet Technology*, vol. 19, no. 2, pp. 1–22, 2019.

[260] A. Yousefpour, A. Patil, G. Ishigaki, I. Kim, X. Wang, H. C. Cankaya, Q. Zhang, W. Xie, and J. P. Jue, "Fogplan: A lightweight qos-aware dynamic fog service provisioning framework," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 5080–5096, 2019.

[261] S. Ghosh, A. Mukherjee, S. K. Ghosh, and R. Buyya, "Mobi-iost: mobility-aware cloud-fog-edge-iot collaborative framework for time-critical applications," *IEEE Transactions on Network Science and Engineering*, vol. 7, no. 4, pp. 2271–2285, 2019.

[262] S. Pallewatta, V. Kostakos, and R. Buyya, "Microfog: a framework for scalable placement of microservices-based iot applications in federated fog environments," *Journal of Systems and Software*, vol. 209, p. 111910, 2024.

[263] A. N. Toosi, C. Agarwal, L. Mashayekhy, S. K. Moghaddam, R. Mahmud, and Z. Tari, "Greenfog: A framework for sustainable fog computing," in *Proceedings of the International Conference on Service-Oriented Computing*. Springer, 2022, pp. 540–549.

[264] L. Nkenyereye, K.-J. Baeg, and W. Chung, "Deep reinforcement learning for containerized edge intelligence inference request processing in iot edge computing," *IEEE Transactions on Services Computing*, vol. 16, no. 6, pp. 4328–4344, 2023.

[265] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.

[266] J.-M. Renders and S. P. Flasse, "Hybrid methods using genetic algorithms for global optimization," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 26, no. 2, pp. 243–258, 1996.

[267] J. Wu, Y.-G. Wang, K. Burrage, Y.-C. Tian, B. Lawson, and Z. Ding, "An improved firefly algorithm for global continuous optimization problems," *Expert Systems with Applications*, vol. 149, p. 113340, 2020.

[268] P. Moradi and M. Gholampour, "A hybrid particle swarm optimization for feature subset selection by integrating a novel local search strategy," *Applied Soft Computing*, vol. 43, pp. 117–130, 2016.

[269] Y. Tang and S. Agrawal, "Discretizing continuous action space for on-policy optimization," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 04, 2020, pp. 5981–5988.

[270] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the International Conference on Artificial Intelligence and Statistics*. JMLR Workshop and Conference Proceedings, 2010, pp. 249–256.

[271] F. Sun, J. Liu, J. Wu, C. Pei, X. Lin, W. Ou, and P. Jiang, "Bert4rec: Sequential recommendation with bidirectional encoder representations from transformer," in

Proceedings of the ACM International Conference on Information and Knowledge Management, 2019, pp. 1441–1450.

[272] J. D. Schaffer, R. Caruana, L. J. Eshelman, and R. Das, "A study of control parameters affecting online performance of genetic algorithms for function optimization," in *Proceedings of the International Conference on Genetic Algorithms*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1989, p. 5160.

[273] X.-S. Yang and X. He, "Firefly algorithm: recent advances and applications," *International Journal of Swarm Intelligence*, vol. 1, no. 1, pp. 36–50, 2013.

[274] F. Van den Bergh and A. P. Engelbrecht, "A study of particle swarm optimization particle trajectories," *Information Sciences*, vol. 176, no. 8, pp. 937–971, 2006.

[275] G. Yan, K. Liu, C. Liu, and J. Zhang, "Edge intelligence for internet of vehicles: A survey," *IEEE Transactions on Consumer Electronics*, vol. 70, no. 2, pp. 4858–4877, 2024.

[276] R. Andreoli, R. Mini, P. Skarin, H. Gustafsson, J. Harmatos, L. Abeni, and T. Cucinotta, "A multi-domain survey on time-criticality in cloud computing," *IEEE Transactions on Services Computing*, vol. 18, no. 2, pp. 1152–1170, 2025.

[277] C. Zhang, Q. He, F. Li, X. Wang, S. Garg, M. S. H. Z. Han, and W. Yuan, "Gai based resource and qoe aware service placement in next-generation multi-domain iot networks," *IEEE Transactions on Cognitive Communications and Networking*, vol. 11, no. 2, pp. 873–885, 2025.

[278] R. Cong, Z. Zhao, M. Wang, G. Min, J. Liu, and J. Mo, "Task-aware service placement for distributed learning in wireless edge networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 36, no. 4, pp. 731–744, 2025.

[279] Z. Zhu, K. Lin, A. K. Jain, and J. Zhou, "Transfer learning in deep reinforcement learning: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 11, pp. 13 344–13 362, 2023.

[280] V. P. Chellapandi, L. Yuan, C. G. Brinton, S. H. Żak, and Z. Wang, "Federated learning for connected and automated vehicles: A survey of existing approaches

and challenges," *IEEE Transactions on Intelligent Vehicles*, vol. 9, no. 1, pp. 119–137, 2023.

[281] Y. Liu, Y. Kang, T. Zou, Y. Pu, Y. He, X. Ye, Y. Ouyang, Y.-Q. Zhang, and Q. Yang, "Vertical federated learning: Concepts, advances, and challenges," *IEEE transactions on knowledge and data engineering*, vol. 36, no. 7, pp. 3615–3634, 2024.

[282] T. Fan *et al.*, "Ten challenging problems in federated foundation models," *IEEE Transactions on Knowledge and Data Engineering*, vol. 37, no. 7, pp. 4314–4337, 2025.

[283] J. Tang, W. Zhao, J. Jin, Y. Xiang, X. Wang, and Z. Zhou, "Adaptive search and collaborative offloading under device-to-device joint edge computing network," *IEEE Transactions on Mobile Computing*, vol. 24, no. 10, pp. 9852–9867, 2025.

[284] X. Zhu, F. Hao, L. Ma, C. Luo, G. Min, and L. T. Yang, "Drl-based joint optimization of wireless charging and computation offloading for multi-access edge computing," *IEEE Transactions on Services Computing*, vol. 18, no. 3, pp. 1352–1367, 2025.

[285] W. Fan, Y. Yu, C. Bao, and Y. Liu, "Vehicular edge intelligence: Drl-based resource orchestration for task inference in vehicle-rsu-edge collaborative networks," *IEEE Transactions on Mobile Computing*, 2025.

[286] J. Chi, X. Zhou, F. Xiao, Y. Lim, and T. Qiu, "Task offloading via prioritized experience-based double dueling dqn in edge-assisted iiot," *IEEE Transactions on Mobile Computing*, vol. 23, no. 12, pp. 14 575–14 591, 2024.

[287] C. Wu, Z. Xu, X. He, Q. Lou, Y. Xia, and S. Huang, "Proactive caching with distributed deep reinforcement learning in 6g cloud-edge collaboration computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 35, no. 8, pp. 1387–1399, 2024.

[288] W. Zhao, Y. Cheng, Z. Liu, X. Wu, and N. Kato, "Asynchronous drl based multi-hop task offloading in rsu-assisted iov networks," *IEEE Transactions on Cognitive Communications and Networking*, vol. 11, no. 1, pp. 546–555, 2025.

[289] H. Zhou, Z. Wang, H. Zheng, S. He, and M. Dong, "Cost minimization-oriented computation offloading and service caching in mobile cloud-edge computing: An a3c-based approach," *IEEE Transactions on Network Science and Engineering*, vol. 10, no. 3, pp. 1326–1338, 2023.

[290] H. Cui, Z. Tang, J. Lou, W. Jia, and W. Zhao, "Latency-aware container scheduling in edge cluster upgrades: A deep reinforcement learning approach," *IEEE Transactions on Services Computing*, vol. 17, no. 5, pp. 2530–2543, 2024.

[291] M. G. Arivazhagan, V. Aggarwal, A. K. Singh, and S. Choudhary, "Federated learning with personalization layers," *arXiv preprint arXiv:1912.00818*, 2019.

[292] L. Collins, H. Hassani, A. Mokhtari, and S. Shakkottai, "Exploiting shared representations for personalized federated learning," in *International conference on machine learning*. PMLR, 2021, pp. 2089–2099.

[293] B. McFee, C. Raffel, D. Liang, D. P. Ellis, M. McVicar, E. Battenberg, and O. Nieto, "librosa: Audio and music signal analysis in python." *SciPy*, vol. 2015, pp. 18–24, 2015.

[294] S. Loria *et al.*, "textblob documentation," *Release 0.15*, vol. 2, no. 8, p. 269, 2018.

[295] S. Bird, "Nltk: the natural language toolkit," in *Proceedings of the COLING/ACL 2006 interactive presentation sessions*, 2006, pp. 69–72.

[296] Y.-Y. Yang, M. Hira, Z. Ni, A. Astafurov, C. Chen, C. Puhrsch, D. Pollack, D. Genzel, D. Greenberg, E. Z. Yang *et al.*, "Torchaudio: Building blocks for audio and speech processing," in *2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2022)*. IEEE, 2022, pp. 6982–6986.

[297] J.-l. Gailly and M. Adler, "zlib: A massively spiffy yet delicately unobtrusive compression library," https://www.zlib.net/, 1995.

[298] P. Deutsch, "Gzip file format specification version 4.3," https://tools.ietf.org/html/rfc1952, RFC 1952, Internet Engineering Task Force, Tech. Rep., 1996.

[299] A. Clark and Contributors, "Pillow (pil fork) documentation," https://pillow.readthedocs.io/, 2015.

[300] G. Bradski, "The opencv library," *Dr. Dobb's Journal of Software Tools*, 2000.

[301] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: a system for large-scale machine learning," in *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, 2016, pp. 265–283.

[302] TensorFlow Team, "Tensorflow lite: On-device machine learning framework," https://www.tensorflow.org/lite, 2017.

[303] C. Lugaresi *et al.*, "Mediapipe: A framework for perceiving and processing reality," in *Third workshop on computer vision for AR/VR at IEEE computer vision and pattern recognition (CVPR)*, 2019.

[304] D. E. King, "Dlib-ml: A machine learning toolkit," http://dlib.net/, pp. 1755–1758, 2009.

[305] JaidedAI, "Easyocr: Ready-to-use ocr with 80+ supported languages," https://github.com/JaidedAI/EasyOCR, 2020.

[306] X. Chang, R. Zhang, J. Mao, and Y. Fu, "Digital twins in transportation infrastructure: an investigation of the key enabling technologies, applications, and challenges," *IEEE Transactions on Intelligent Transportation Systems*, vol. 25, no. 7, pp. 6449–6471, 2024.

[307] Z. Wu, J. Hou, Y. Diao, and B. He, "Federated transformer: Multi-party vertical federated learning on practical fuzzily linked data," *Advances in Neural Information Processing Systems*, vol. 37, pp. 45 791–45 818, 2024.

[308] J. Zhang, X. Cheng, L. Yang, J. Hu, H. Tian, and K. Chen, "High-performance hardware acceleration architecture for cross-silo federated learning," *IEEE Transactions on Parallel and Distributed Systems*, vol. 35, no. 8, pp. 1506–1523, 2024.

[309] Y. Li, J.-K. Hao, and L. Song, "Dynamic bin packing with heterogeneous dependent bins for regionless in geo-distributed clouds," *IEEE Transactions on Computers*, 2025.

[310] S. Tian, S. Xiang, Z. Zhou, H. Dai, E. Yu, and Q. Deng, "Task offloading and resource allocation based on reinforcement learning and load balancing in vehicular networking," *IEEE Transactions on Consumer Electronics*, vol. 71, no. 1, pp. 2217–2230, 2025.

[311] J. Fan, K. Wu, G. Tang, Y. Zhou, and S. Huang, "Taking advantage of the mistakes: Rethinking clustered federated learning for iot anomaly detection," *IEEE Transactions on Parallel and Distributed Systems*, vol. 35, no. 6, pp. 862–876, 2024.

[312] X. Li, W. Huangfu, X. Xu, J. Huo, and K. Long, "Secure offloading with adversarial multi-agent reinforcement learning against intelligent eavesdroppers in uav-enabled mobile edge computing," *IEEE Transactions on Mobile Computing*, vol. 23, no. 12, pp. 13 914–13 928, 2024.

[313] D. Yunis, J. Jung, F. Dai, and M. Walter, "Subwords as skills: Tokenization for sparse-reward reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 37, pp. 67 663–67 688, 2024.

[314] H. Li, C. Dou, D. Yue, G. P. Hancke, Z. Zeng, W. Guo, and L. Xu, "End-edge-cloud collaboration-based false data injection attack detection in distribution networks," *IEEE Transactions on Industrial Informatics*, vol. 20, no. 2, pp. 1786–1797, 2023.

[315] D. Jin, N. Kannengiesser, S. Rank, and A. Sunyaev, "Collaborative distributed machine learning," *ACM Computing Surveys*, vol. 57, no. 4, pp. 1–36, 2024.

[316] M. Gecer and B. Garbinato, "Federated learning for mobility applications," *ACM Computing Surveys*, vol. 56, no. 5, pp. 1–28, 2024.

[317] X. Deng, H. Yang, J. Zhang, J. Gui, S. Lin, X. Wang, and G. Min, "Task offloading in internet of vehicles: A drl-based approach with representation learning for dag scheduling," *IEEE Transactions on Mobile Computing*, vol. 24, no. 6, pp. 5045–5060, 2025.

[318] B. Zhang, T. Jing, Q. Gao, X. Li, and M. Zhu, "D3qn-enabled diversified-task co-offloading for synthetic-expense minimization in industrial internet of things (iiot)," *IEEE Transactions on Industrial Informatics*, vol. 21, no. 12, pp. 9491–9502, 2025.

[319] P. Li, Z. Xiao, H. Gao, X. Wang, and Y. Wang, "Reinforcement learning based edge-end collaboration for multi-task scheduling in 6g enabled intelligent autonomous transport systems," *IEEE Transactions on Intelligent Transportation Systems*, vol. 26, no. 10, pp. 17 624–17 637, 2025.

[320] Q. Chen, X. Song, T. Song, and Y. Yang, "Vehicular edge computing networks optimization via drl-based communication resource allocation and load balancing," *IEEE Transactions on Mobile Computing*, vol. 24, no. 9, pp. 9222–9237, 2025.

[321] Y. Liu, L. Jiang, C. Yuen, and Y. Zhang, "Vehicular computing power networks for iot-driven edge intelligence: Ma-ddpg-based robust task offloading and resource allocation," *IEEE Internet of Things Journal*, vol. 12, no. 18, pp. 36 868–36 879, 2025.

[322] M. R. Raju, S. K. Mothku, and M. K. Somesula, "Dmits: Dependency and mobility-aware intelligent task scheduling in socially-enabled vfc based on federated drl approach," *IEEE Transactions on Intelligent Transportation Systems*, vol. 25, no. 11, pp. 17 007–17 022, 2024.

[323] X. Chen, B. Xiao, X. Lin, Z. Chen, and G. Min, "Multi-agent collaboration for vehicular task offloading using federated deep reinforcement learning," *IEEE Transactions on Mobile Computing*, vol. 24, no. 9, pp. 8856–8871, 2025.

[324] R. T. Rockafellar, S. Uryasev *et al.*, "Optimization of conditional value-at-risk," *Journal of Risk*, vol. 2, pp. 21–42, 2000.

[325] R. S. Sutton, *Temporal credit assignment in reinforcement learning*. University of Massachusetts Amherst, 1984.

[326] M. H. Ahmadilivani, M. Taheri, J. Raik, M. Daneshtalab, and M. Jenihhin, "A systematic literature review on hardware reliability assessment methods for deep neural networks," *ACM Computing Surveys*, vol. 56, no. 6, pp. 1–39, 2024.

[327] E. Yu, D. Dong, and X. Liao, "Communication optimization algorithms for distributed deep learning systems: A survey," *IEEE Transactions on Parallel and Distributed Systems*, vol. 34, no. 12, pp. 3294–3308, 2023.

[328] H. Dui, H. Zhang, X. Dong, S. Wu, and Y. Wang, "Multi-stage control strategy of IoT-enabled unmanned vehicle detection systems," *IEEE Transactions on Intelligent Transportation Systems*, vol. 26, no. 5, pp. 6425–6440, 2025.

[329] A. Khochare, A. Krishnan, and Y. Simmhan, "A scalable platform for distributed object tracking across a many-camera network," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 6, pp. 1479–1493, 2021.

[330] Z. Lv, B. Hu, and H. Lv, "Infrastructure monitoring and operation for smart cities based on iot system," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 3, pp. 1957–1962, 2019.

[331] D. Vasisht, Z. Kapetanovic, J. Won, X. Jin, R. Chandra, S. Sinha, A. Kapoor, M. Sudarshan, and S. Stratman, "Farmbeats: an iot platform for data-driven agriculture," in *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation*, 2017, pp. 515–529.

[332] Q. He, J. Lin, H. Fang, X. Wang, M. Huang, X. Yi, and K. Yu, "Integrating iot and 6g: Applications of edge intelligence, challenges, and future directions," *IEEE Transactions on Services Computing*, vol. 18, no. 4, pp. 2471–2488, 2025.

[333] S. Raj, R. Singh, K. Astu, and Y. Simmhan, "Aerodaas: Towards an application programming framework for drones-as-a-service," in *Proceedings of the IEEE International Conference on Web Services (ICWS)*, 2025.

[334] "Nvidia jetson orin nano developer kit: Technical specifications," https://developer.nvidia.com/embedded/learn/getting-started-jetson, 2025.

[335] Z. Shen, J. Jin, C. Tan, A. Tagami, S. Wang, Q. Li, Q. Zheng, and J. Yuan, "A survey of next-generation computing technologies in space-air-ground integrated networks," *ACM Computing Surveys*, vol. 56, no. 1, pp. 1–40, 2023.

[336] Y. K. Tun, G. Dn, Y. M. Park, and C. S. Hong, "Joint uav deployment and resource allocation in thz-assisted mec-enabled integrated space-air-ground networks," *IEEE Transactions on Mobile Computing*, vol. 24, no. 5, pp. 3794–3808, 2025.

[337] L. Zeng, H. Chen, D. Feng, X. Zhang, and X. Chen, "A3d: Adaptive, accurate, and autonomous navigation for edge-assisted drones," *IEEE/ACM Transactions on Networking*, vol. 32, no. 1, pp. 713–728, 2023.

[338] Y. Liao, Y. Xu, H. Xu, L. Wang, C. Qian, and C. Qiao, "Decentralized federated learning with adaptive configuration for heterogeneous participants," *IEEE Transactions on Mobile Computing*, vol. 23, no. 6, pp. 7453–7469, 2023.

[339] M. Centenaro, C. E. Costa, F. Granelli, C. Sacchi, and L. Vangelista, "A survey on technologies, standards and open challenges in satellite iot," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 3, pp. 1693–1720, 2021.

[340] F. Pervez, A. Sultana, C. Yang, and L. Zhao, "Energy and latency efficient joint communication and computation optimization in a multi-uav-assisted mec network," *IEEE Transactions on Wireless Communications*, vol. 23, no. 3, pp. 1728–1741, 2024.

[341] Q. Qi, T. Shi, K. Qin, and G. Luo, "Completion time optimization in uav-relaying-assisted mec networks with moving users," *IEEE Transactions on Consumer Electronics*, vol. 70, no. 1, pp. 1246–1258, 2024.

[342] X. Dai, Z. Xiao, H. Jiang, and J. C. S. Lui, "Uav-assisted task offloading in vehicular edge computing networks," *IEEE Transactions on Mobile Computing*, vol. 23, no. 4, pp. 2520–2534, 2024.

[343] L. He, G. Sun, Z. Sun, P. Wang, J. Li, S. Liang, and D. Niyato, "An online joint optimization approach for qoe maximization in uav-enabled mobile edge computing," in *Proceedings of the IEEE Conference on Computer Communications (INFOCOM)*, 2024, pp. 101–110.

[344] J. Xu, X. Liu, A. G. Neiat, L. Chu, X. Li, and Y. Yang, "A holistic and hybrid service

selection strategy for mec-based uav last-mile delivery systems," *IEEE Transactions on Services Computing*, vol. 17, no. 6, pp. 3022–3036, 2024.

[345] A. Du, J. Jia, S. Dustdar, J. Chen, and X. Wang, "Online service placement, task scheduling, and resource allocation in hierarchical collaborative mec systems," *IEEE Transactions on Services Computing*, vol. 18, no. 2, pp. 983–997, 2025.

[346] Y. Gao, J. Tao, Y. Xu, Z. Wang, Y. Gao, and M. Wang, "Improving user qoe via joint trajectory and resource optimization in multi-uav assisted mec," *IEEE Transactions on Services Computing*, 2025.

[347] F. Song, Q. Yang, M. Deng, H. Xing, Y. Liu, X. Yu, K. Li, and L. Xu, "Aoi and energy tradeoff for aerial-ground collaborative mec: A multi-objective learning approach," *IEEE Transactions on Mobile Computing*, vol. 23, no. 12, pp. 11 278–11 294, 2024.

[348] C. Li, K. Jiang, G. He, F. Bing, and Y. Luo, "A computation offloading method for multi-uavs assisted mec based on improved federated ddpg algorithm," *IEEE Transactions on Industrial Informatics*, 2024.

[349] Y. Chen, Y. Yang, Y. Wu, J. Huang, and L. Zhao, "Joint trajectory optimization and resource allocation in uav-mec systems: A lyapunov-assisted drl approach," *IEEE Transactions on Services Computing*, 2025.

[350] F. Zhou, R. Q. Hu, Z. Li, and Y. Wang, "Mobile edge computing in unmanned aerial vehicle networks," *IEEE Wireless Communications*, vol. 27, no. 1, pp. 140–146, 2020.

[351] A. Goldsmith, *Wireless communications*. Cambridge press, 2005.

[352] L. Zeng, X. Liao, Z. Ma, B. Xiong, H. Jiang, and Z. Chen, "Three-dimensional uav-to-uav channels: Modeling, simulation, and capacity analysis," *IEEE Internet of Things Journal*, vol. 11, no. 6, pp. 10 054–10 068, 2024.

[353] J. M. Rabaey, A. Chandrakasan, and B. Nikolic, *Digital integrated circuits*. Prentice hall Englewood Cliffs, 2002, vol. 2.

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

[355] J. Lee, D. Kim, and B. Ham, "Network quantization with element-wise gradient scaling," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 6448–6457.

[356] M. Shabanighazikelayeh and E. Koyuncu, "Optimal placement of uavs for minimum outage probability," *IEEE Transactions on Vehicular Technology*, vol. 71, no. 9, pp. 9558–9570, 2022.

[357] Y. Liu, K. Liu, J. Han, L. Zhu, Z. Xiao, and X.-G. Xia, "Resource allocation and 3-d placement for uav-enabled energy-efficient iot communications," *IEEE Internet of Things Journal*, vol. 8, no. 3, pp. 1322–1333, 2020.

[358] M. Samir, S. Sharafeddine, C. M. Assi, T. M. Nguyen, and A. Ghrayeb, "Uav trajectory planning for data collection from time-constrained iot devices," *IEEE Transactions on Wireless Communications*, vol. 19, no. 1, pp. 34–46, 2019.

[359] L. Valente, A. Nadalini, A. H. C. Veeran, M. Sinigaglia, B. Sá, N. Wistoff, Y. Tortorella, S. Benatti, R. Psiakis, A. Kulmala *et al.*, "A heterogeneous risc-v based soc for secure nano-uav navigation," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 71, no. 5, pp. 2266–2279, 2024.

[360] Y. Ren, F. Zhu, G. Lu, Y. Cai, L. Yin, F. Kong, J. Lin, N. Chen, and F. Zhang, "Safety-assured high-speed navigation for mavs," *Science Robotics*, vol. 10, no. 98, p. eado6187, 2025.

[361] S. Liu, M. Watterson, S. Tang, and V. Kumar, "High speed navigation for quadrotors with limited onboard sensing," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 1484–1491.

[362] D. Tyrovolas, P.-V. Mekikis, S. A. Tegos, P. D. Diamantoulakis, C. K. Liaskos, and G. K. Karagiannidis, "Energy-aware design of uav-mounted ris networks for iot

data collection," *IEEE Transactions on Communications*, vol. 71, no. 2, pp. 1168–1178, 2022.

[363] L. M. B. Ribeiro and L. Buss Becker, "Performance analysis of ieee 802.11p and ieee 802.11n based on qos for uav networks," in *Proceedings of the ACM Symposium on Design and Analysis of Intelligent Vehicular Networks and Applications (DIVANet)*, 2019.

[364] C. Shepard, A. Javed, and L. Zhong, "Control channel design for many-antenna mu-mimo," in *Proceedings of the International Conference on Mobile Computing and Networking (MobiCom)*, 2015.

[365] P. Li, L. Xie, J. Yao, and J. Xu, "Cellular-connected uav with adaptive air-to-ground interference cancellation and trajectory optimization," *IEEE Communications Letters*, vol. 26, no. 6, pp. 1368–1372, 2022.

[366] R. Amorim, P. Mogensen, T. Sorensen, I. Z. Kovacs, and J. Wigard, "Pathloss measurements and modeling for uavs connected to cellular networks," in *Proceedings of the IEEE Vehicular Technology Conference (VTC)*, 2017, pp. 1–6.

[367] W. Chen, S. Zhao, R. Zhang, Y. Chen, and L. Yang, "Uav-assisted data collection with nonorthogonal multiple access," *IEEE Internet of Things Journal*, vol. 8, no. 1, pp. 501–511, 2021.

[368] W. Xu, Y. Sun, R. Zou, W. Liang, Q. Xia, F. Shan, T. Wang, X. Jia, and Z. Li, "Throughput maximization of uav networks," *IEEE/ACM Transactions on Networking*, vol. 30, no. 2, pp. 881–895, 2022.

[369] D. Shan, K. Zeng, W. Xiang, P. Richardson, and Y. Dong, "Phy-cram: Physical layer challenge-response authentication mechanism for wireless networks," *IEEE Journal on Selected Areas in Communications*, vol. 31, no. 9, pp. 1817–1827, 2013.

[370] D. Yang, Q. Wu, Y. Zeng, and R. Zhang, "Energy tradeoff in ground-to-uav communication via trajectory design," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 7, pp. 6721–6726, 2018.

[371] B. Shang, L. Liu, J. Ma, and P. Fan, "Unmanned aerial vehicle meets vehicle-to-everything in secure communications," *IEEE Communications Magazine*, vol. 57, no. 10, pp. 98–103, 2019.

[372] Q. Liu, L. Shi, L. Sun, J. Li, M. Ding, and F. Shu, "Path planning for uav-mounted mobile edge computing with deep reinforcement learning," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 5, pp. 5723–5728, 2020.

[373] X. Dai, B. Duo, X. Yuan, and M. Di Renzo, "Energy-efficient uav communications in the presence of wind: 3d modeling and trajectory design," *IEEE Transactions on Wireless Communications*, vol. 23, no. 3, pp. 1840–1854, 2023.

[374] T. Wan and M.-H. Tsai, "Numerical study of quad-rotor aircraft performance under adverse situations," in *Proceedings of the International Council of the Aeronautical Sciences (ICAS)*, 2020.

[375] A. Ho, E. Erdil, and T. Besiroglu, "Limits to the energy efficiency of cmos microprocessors," in *Proceedings of the IEEE International Conference on Rebooting Computing (ICRC)*, 2023, pp. 1–10.

[376] S. Chen, L. Wang, and F. Liu, "Optimal admission control mechanism design for time-sensitive services in edge computing," in *Proceedings of the IEEE Conference on Computer Communications (INFOCOM)*, 2022, pp. 1169–1178.

[377] H. Cheng, W. Xia, F. Yan, and L. Shen, "Balanced clustering and joint resources allocation in cooperative fog computing system," in *Proceedings of the IEEE Global Communications Conference (GLOBECOM)*, 2019, pp. 1–6.

[378] M. Hu, Z. Xie, D. Wu, Y. Zhou, X. Chen, and L. Xiao, "Heterogeneous edge offloading with incomplete information: A minority game approach," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 9, pp. 2139–2154, 2020.

[379] J. B. D. da Costa, A. M. de Souza, R. I. Meneguette, E. Cerqueira, D. Rosrio, C. Sommer, and L. Villas, "Mobility and deadline-aware task scheduling mechanism for vehicular edge computing," *IEEE Transactions on Intelligent Transportation Systems*, vol. 24, no. 10, pp. 11 345–11 359, 2023.

[380] J. Zhang, X. Wang, P. Yuan, H. Dong, P. Zhang, and Z. Tari, "Dependency-aware task offloading based on application hit ratio," *IEEE Transactions on Services Computing*, vol. 17, no. 6, pp. 3373–3386, 2024.

[381] J. Li, Y. Gao, Y. Yang, Y. Bai, X. Zhou, Y. Li, H. Sun, Y. Liu, X. Si, Y. Ye, Y. Wu, Y. Lin, B. Xu, B. Ren, C. Feng, and H. Huang, "Fundamental capabilities and applications of large language models: A survey," *ACM Computing Surveys*, vol. 58, no. 2, Sep. 2025.

[382] Z. Deng, H. Tian, X. Zheng, and D. D. Zeng, "Deep causal learning: representation, discovery and inference," *ACM Computing Surveys*, vol. 58, no. 2, pp. 1–36, 2025.

[383] J. Zheng, S. Qiu, C. Shi, and Q. Ma, "Towards lifelong learning of large language models: A survey," *ACM Computing Surveys*, vol. 57, no. 8, pp. 1–35, 2025.

[384] Z. Zhou, X. Chen, M. Ying, Z. Yang, C. Huang, Y. Cai, and Z. Zhang, "Unified design of space-air-ground-sea integrated maritime communications," *IEEE Transactions on Communications*, vol. 73, no. 12, pp. 13 441–13 455, 2025.

[385] T. M. Getu, G. Kaddoum, and M. Bennis, "Semantic communication: A survey on research landscape, challenges, and future directions," *Proceedings of the IEEE*, vol. 112, no. 11, pp. 1649–1685, 2025.

[386] M. Beliaev and R. Pedarsani, "Inverse reinforcement learning by estimating expertise of demonstrators," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 39, no. 15, 2025, pp. 15 532–15 540.

[387] Y. Jiang, B. Ma, X. Wang, G. Yu, P. Yu, Z. Wang, W. Ni, and R. P. Liu, "Blockchained federated learning for internet of things: A comprehensive survey," *ACM Computing Surveys*, vol. 56, no. 10, pp. 1–37, 2024.

[388] K. Sedghighadikolaei and A. A. Yavuz, "A survey of threshold signatures: Nist standards, post-quantum cryptography, exotic techniques, and real-world applications," *ACM computing surveys*, vol. 58, no. 6, Dec. 2025.

[389] N. Rathi, I. Chakraborty, A. Kosta, A. Sengupta, A. Ankit, P. Panda, and K. Roy, "Exploring neuromorphic computing based on spiking neural networks: Algorithms to hardware," *ACM Computing Surveys*, vol. 55, no. 12, pp. 1–49, 2023.

[390] M. M. S. Ahmad and S. H. Klidbary, "Hardware-software co-design low-complexity fuzzy algorithm for high-dimensional feature space with in-situ learning," *IEEE Transactions on Emerging Topics in Computing*, vol. 13, no. 3, pp. 1156–1169, 2025.

[391] O. Danushi, S. Forti, and J. Soldani, "Carbon-efficient software design and development: A systematic literature review," *ACM computing surveys*, vol. 57, no. 10, pp. 1–35, 2025.