A Knowledge Distillation-empowered Adaptive Federated Reinforcement Learning Framework for Multi-Domain IoT Applications Scheduling

Zhiyu Wang, Mohammad Goudarzi, Mingming Gong, and Rajkumar Buyya

arXiv:2508.21328v1 [cs.DC] 29 Aug 2025

***Abstract*—The rapid proliferation of Internet of Things (IoT) applications across heterogeneous Cloud-Edge-IoT environments presents significant challenges in distributed scheduling optimization. Existing approaches face issues, including fixed neural network architectures that are incompatible with computational heterogeneity, non-Independent and Identically Distributed (non-IID) data distributions across IoT scheduling domains, and insufficient cross-domain collaboration mechanisms. This paper proposes KD-AFRL, a Knowledge Distillation-empowered Adaptive Federated Reinforcement Learning framework that addresses multi-domain IoT application scheduling through three core innovations. First, we develop a resource-aware hybrid architecture generation mechanism that creates dual-zone neural networks enabling heterogeneous devices to participate in collaborative learning while maintaining optimal resource utilization. Second, we propose a privacy-preserving environment-clustered federated learning approach that utilizes differential privacy and K-means clustering to address non-IID challenges and facilitate effective collaboration among compatible domains. Third, we introduce an environment-oriented cross-architecture knowledge distillation mechanism that enables efficient knowledge transfer between heterogeneous models through temperature-regulated soft targets. Comprehensive experiments with real Cloud-Edge-IoT infrastructure demonstrate KD-AFRL's effectiveness using diverse IoT applications. Results show significant improvements over the best baseline, with 21% faster convergence and 15.7%, 10.8%, and 13.9% performance gains in completion time, energy consumption, and weighted cost, respectively. Scalability experiments reveal that KD-AFRL achieves 3-5 times better performance retention compared to existing solutions as the number of domains increases.**

***Index Terms*—Internet of Things, Edge/Cloud Computing, Deep Reinforcement Learning, Federated Learning, Knowledge Distillation**

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 [1]. Modern IoT deployments span multiple autonomous domains—from smart cities and industrial automation to healthcare monitoring and autonomous vehicles—each exhibiting distinct computational capabilities, workload characteristics, and operational constraints [2], [3]. 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 [4]. 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 [5]. 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 [6]. 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 distributing 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 [7]. 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 [8]. However, applying federated learning to multi-domain IoT scheduling introduces several fundamental challenges. First, the heterogeneity in computational capabilities

Zhiyu Wang and Rajkumar Buyya are with The Quantum Cloud Computing and Distributed Systems (qCLOUDS) Laboratory, School of Computing and Information Systems, The University of Melbourne, Australia (e-mail: zhiywang1@student.unimelb.edu.au, rbuyya@unimelb.edu.au).

Mingming Gong is with School of Mathematics and Statistics, The University of Melbourne, Australia (email: mingming.gong@unimelb.edu.au)

Mohammad Goudarzi is with The Faculty of Information Technology, Monash University, Australia (email: mohammad.goudarzi@monash.edu)

across domains necessitates adaptive model architectures that can scale appropriately to device constraints while maintaining learning effectiveness [9]. 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 [10]. 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 [11].

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 work 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 paper is organized as follows. Section 2 reviews related work. Section 3 presents the system model and problem formulation. Section 4 details the KD-AFRL framework. Section 5 presents experimental evaluation. Section 6 concludes the paper.

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.

2.1 Centralized DRL for IoT Scheduling

Tang et al. [12] 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. [13] 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. [14] 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. [15] 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. [16] 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. [17] 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. [18] 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. [19] 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.

2.2 Distributed DRL for IoT Scheduling

Wu et al. [20] 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. [21] 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. [6] 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. [22] 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. [23] 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. [24] 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. [25] 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. [26] 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.

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 I, evaluating them across four critical dimensions: application properties, system properties, technique properties, and evaluation methodology.

2.3.1 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.

2.3.2 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 heterogeneity 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.

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.

3.1 System Model

Fig. 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.

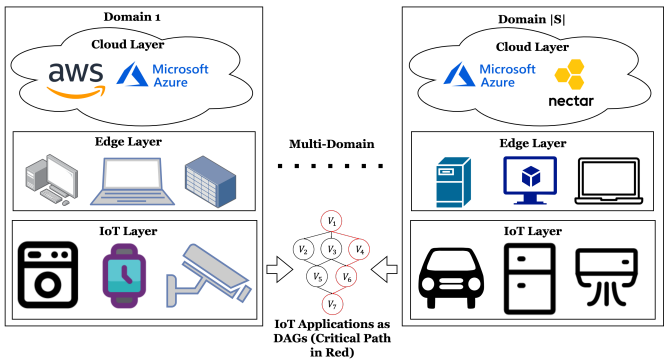


Fig. 1: The hierarchical Cloud-Edge-IoT computing architecture with distributed scheduling domains.

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

TABLE I: 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. [12]	Multiple	Independent	◐	Homogeneous	Edge	Heterogeneous	×	Centralized	DQN	✓	✓	✓	×	Simulation
Zhu et al. [13]	Single	Independent	○	Homogeneous	Edge	Homogeneous	×		PPO	×	✓	✓	×	Simulation
Fan et al. [14]	Single	Independent	◐	Homogeneous	Edge	Heterogeneous	×		DDPG	✓	✓	✓	×	Simulation
Chen et al. [15]	Multiple	Dependent	◐	Heterogeneous	Edge and Cloud	Heterogeneous	×		DQN	✓	×	×	×	Simulation
Wang et al. [16]	Multiple	Dependent	●	Heterogeneous	Edge and Cloud	Heterogeneous	×		PPO	✓	×	✓	×	Practical
Hsieh et al. [17]	Single	Independent	◐	Heterogeneous	Edge and Cloud	Heterogeneous	×		Actor-Critic	✓	×	×	×	Simulation
Zhao et al. [18]	Multiple	Dependent	◐	Heterogeneous	Edge	Heterogeneous	×		DDPG	✓	✓	✓	×	Simulation
Chi et al. [19]	Single	Independent	○	Homogeneous	Edge	Homogeneous	×		DQN	✓	✓	✓	×	Simulation
Wu et al. [20]	Single	Independent	◐	Homogeneous	Edge and Cloud	Homogeneous	×	Distributed	DQN	✓	×	✓	×	Simulation
Zhao et al. [21]	Single	Independent	○	Homogeneous	Edge and Cloud	Homogeneous	×		A3C	✓	×	×	×	Simulation
Wang et al. [6]	Multiple	Dependent	●	Heterogeneous	Edge and Cloud	Heterogeneous	×		IMPALA	✓	✓	✓	×	Practical
Zhou et al. [22]	Single	Independent	○	Homogeneous	Edge and Cloud	Homogeneous	×		A3C	✓	×	✓	×	Simulation
Zhang et al. [23]	Single	Independent	◐	Heterogeneous	Edge and Cloud	Heterogeneous	×		A3C	✓	×	×	×	Simulation
Ju et al. [24]	Single	Independent	◐	Heterogeneous	Edge	Heterogeneous	×		A3C	×	✓	×	×	Simulation
Liu et al. [25]	Single	Independent	◐	Homogeneous	Edge	Homogeneous	×		A3C	✓	×	×	×	Simulation
Shen et al. [26]	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

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. 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.

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:

$$\mathcal{C}_i^j = (\mathcal{N}_k, \mathcal{S}_m), \quad k \in \{1, \ldots, |\mathcal{N}|\}, \quad m \in \{1, \ldots, |\mathcal{S}|\}, \tag{1}$$

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:

$$\mathcal{C}_i = \{\mathcal{C}_i^j | 1 \leq j \leq |\mathcal{A}_i|\}, \tag{2}$$

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 $\mathcal{A}_i$.

3.2.1 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 $T^{pd}(\mathcal{C}_i^j)$:

$$TCT(\mathcal{C}_i^j) = T^{cl}(\mathcal{C}_i^j) + T^{pd}(\mathcal{C}_i^j). \tag{3}$$

The communication latency model $T^{cl}(\mathcal{C}_i^j)$ represents the maximum time required for data dependencies to be satisfied before task execution:

$$T^{cl}(\mathcal{C}_i^j) = \max_{\mathcal{A}_i^k \in PR(\mathcal{A}_i^j)} T^{cl}_{\mathcal{C}_i^k, \mathcal{C}_i^j}, \tag{4}$$

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:

$$T^{cl}_{\mathcal{C}_i^k, \mathcal{C}_i^j} = \begin{cases} T^{dt}_{\mathcal{C}_i^k, \mathcal{C}_i^j} + \mathcal{P}_{\mathcal{N}_l, \mathcal{N}_k} & \text{if servers differ,} \\ 0 & \text{if same server,} \end{cases} \tag{5}$$

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:

$$T^{dt}_{\mathcal{C}_i^k, \mathcal{C}_i^j} = \frac{DV_{\mathcal{C}_i^k, \mathcal{C}_i^j}(\mathcal{A}_i^j)}{\mathcal{B}_{\mathcal{N}_l, \mathcal{N}_k}}, \tag{6}$$

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:

$$T^{pd}(\mathcal{C}_i^j) = \frac{CC(\mathcal{A}_i^j)}{Freq(\mathcal{N}_k)}, \tag{7}$$

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:

$$CT(\mathcal{C}_i) = \sum_{j=1}^{|\mathcal{A}_i|}(TCT(\mathcal{C}_i^j) \times CP(\mathcal{A}_i^j)), \tag{8}$$

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:

$$\min_{\mathcal{C}} \sum_{m=1}^{|\mathcal{S}|} \sum_{\mathcal{A}_i \in \mathcal{A}_{\mathcal{S}_m}} CT(\mathcal{C}_i), \tag{9}$$

where $\mathcal{C}$ represents the collective scheduling decisions across all domains.

3.2.2 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:

$$E(\mathcal{C}_i^j) = E^{pd}(\mathcal{C}_i^j) + (E^{cm}(\mathcal{C}_i^j) \times ED(\mathcal{A}_i^j)), \tag{10}$$

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:

$$E^{pd}(\mathcal{C}_i^j) = T^{pd}(\mathcal{C}_i^j) \times P^{pd}(\mathcal{N}_k), \tag{11}$$

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:

$$E^{cm}(\mathcal{C}_i^j) = \sum_{\mathcal{A}_i^l \in SU(\mathcal{A}_i^j)} \frac{DV_{\mathcal{C}_i^j, \mathcal{C}_i^l}(\mathcal{A}_i^j)}{\mathcal{B}_{\mathcal{N}_k, \mathcal{N}_q}} \times P^{cm}(\mathcal{N}_k) \times \delta(\mathcal{N}_k, \mathcal{N}_q), \tag{12}$$

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:

$$E(\mathcal{C}_i) = \sum_{j=1}^{|\mathcal{A}_i|} E(\mathcal{C}_i^j). \tag{13}$$

For each scheduler $\mathcal{S}_m$, the global energy optimization objective is to minimize the sum of application energy consumption across all scheduling domains:

$$\min_{\mathcal{C}} \sum_{m=1}^{|\mathcal{S}|} \sum_{\mathcal{A}_i \in \mathcal{A}_{\mathcal{S}_m}} E(\mathcal{C}_i), \tag{14}$$

where $\mathcal{C}$ represents the collective scheduling decisions across all domains.

3.2.3 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:

$$J(\mathcal{C}_i) = \alpha_{cost} \times \frac{CT(\mathcal{C}_i) - CT^{min}}{CT^{max} - CT^{min}} + (1 - \alpha_{cost}) \times \frac{E(\mathcal{C}_i) - E^{min}}{E^{max} - E^{min}}, \tag{15}$$

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:

$$\min_{\mathcal{C}} \sum_{m=1}^{|\mathcal{S}|} \sum_{\mathcal{A}_i \in \mathcal{A}_{\mathcal{S}_m}} J(\mathcal{C}_i). \tag{16}$$

This optimization problem is subject to the following constraints:

$$\text{s.t.} \quad C1: \ |\{\mathcal{N}_k | (\mathcal{N}_k, \mathcal{S}_m) = \mathcal{C}_i^j\}| = 1, \ \forall \mathcal{C}_i^j \in \mathcal{C}_i \tag{17}$$

$$C2: \ DV_{\mathcal{C}_i^k, \mathcal{C}_i^j}(\mathcal{A}_i^j), \mathcal{B}_{\mathcal{N}_l, \mathcal{N}_k} > 0, \ \forall \mathcal{N}_l, \mathcal{N}_k \in \mathcal{N}, \ \forall \mathcal{A}_i^j \in \mathcal{A}_i \tag{18}$$

$$C3: \ Freq(\mathcal{N}_k), Ram(\mathcal{N}_k) > 0, \ \forall \mathcal{N}_k \in \mathcal{N} \tag{19}$$

$$C4: \sum_{\mathcal{A}_i \in \mathcal{A}_{\mathcal{S}_m}} \sum_{\mathcal{A}_i^j \in \mathcal{A}_i} Ram(\mathcal{A}_i^j) \times SO(\mathcal{A}_i^j, \mathcal{N}_k) < Ram(\mathcal{N}_k), \ \forall \mathcal{N}_k \in \mathcal{N}_m, \forall \mathcal{S}_m \in \mathcal{S} \tag{20}$$

$$C5: \ TCT(\mathcal{A}_i^k) \geq TCT(\mathcal{A}_i^j) + T^{cl}_{\mathcal{C}_i^j, \mathcal{C}_i^k}, \forall \mathcal{A}_i^j \in PR(\mathcal{A}_i^k) \tag{21}$$

$$C6: \ 0 \leq \alpha_{cost} \leq 1 \tag{22}$$

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 [27]. 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.

3.3 Deep Reinforcement Learning Formulation

To solve the complex multi-domain scheduling optimization problem, we formulate it as a DRL problem where each scheduler $\mathcal{S}_m$ learns an optimal scheduling policy through interactions 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.

3.3.1 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:

$$s_t^m = \{SR_t^m, AT_t^m, QT_t^m\}, \tag{23}$$

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.

3.3.2 Action Space

For each scheduler $\mathcal{S}_m$, the action at time step t is defined as:

$$a_t^m = \{n_k | n_k \in \mathcal{N}_m\}. \tag{24}$$

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.

3.3.3 Reward Function

The reward function directly connects to our optimization objective, providing feedback on scheduling decision quality. For scheduler $\mathcal{S}_m$, the reward at time step t is:

$$r_t^m = \begin{cases} -J(\mathcal{C}_i) & \text{if task execution succeeds,} \\ \text{penalty} & \text{if task execution fails.} \end{cases} \tag{25}$$

Here, $J(\mathcal{C}_i)$ is the weighted cost model from Eq. 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.

3.3.4 Policy and Objective

The agent's policy function defines the probability of selecting action a when observing state s:

$$\pi(a|s) = Pr[A_t = a|S_t = s]. \tag{26}$$

The ultimate goal of each scheduler $\mathcal{S}_m$ is to learn a policy π_m that maximizes the expected cumulative discounted reward:

$$\pi_m^* = \arg\max_{\pi_m} \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t^m | \pi_m\right]. \tag{27}$$

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.

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.

These mechanisms operate synergistically within an iterative learning paradigm, blending distributed experience collection with coordinated federated optimization.

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. 2.

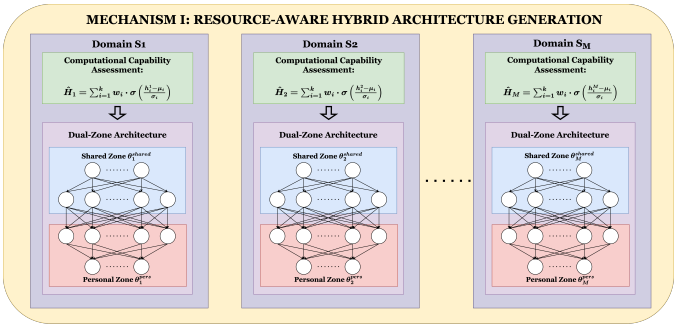


Fig. 2: Resource-aware hybrid architecture generation mechanism showing computational capability assessment and dual-zone architecture design across heterogeneous domains.

4.1.1 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 $H_m \in \mathbb{R}^k$:

$$H_m = [h_1^m, h_2^m, ..., h_k^m]^T, \tag{28}$$

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:

$$h_i^m = \alpha_i \cdot h_i^{m,\text{static}} \cdot (1 - \beta_i \cdot \text{util}_i^m). \tag{29}$$

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 non-linear aggregation:

$$\hat{H}_m = \sum_{i=1}^{k} w_i \cdot \sigma\left(\frac{h_i^m - \mu_i}{\sigma_i}\right), \tag{30}$$

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.

4.1.2 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):

$$\theta_m = \{\theta_m^{shared}, \theta_m^{pers}\}. \tag{31}$$

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.

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:

$$\text{Type}_m = \min\left(\lceil \hat{H}_m \cdot K_{arch} \rceil, K_{arch}\right). \tag{32}$$

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:

$$\theta_m^{depth,pers} = \lceil d_{\min} + \alpha_{arch} \cdot \hat{H}_m \cdot (d_{\max} - d_{\min}) \rceil, \tag{33}$$

$$\theta_m^{width,pers} = \lceil w_{\min} + \alpha_{arch} \cdot \hat{H}_m \cdot (w_{\max} - w_{\min}) \rceil, \tag{34}$$

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 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.

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-

Algorithm 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*

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. 3.

4.2.1 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:

$$\tilde{\mathcal{F}}_m = \mathcal{F}_m + \xi_m, \tag{35}$$

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).

4.2.2 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 similar scheduling environments. The similarity between domains is measured using Euclidean distance:

$$d_{ij} = ||\tilde{\mathcal{F}}_i - \tilde{\mathcal{F}}_j||_2. \tag{36}$$

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.

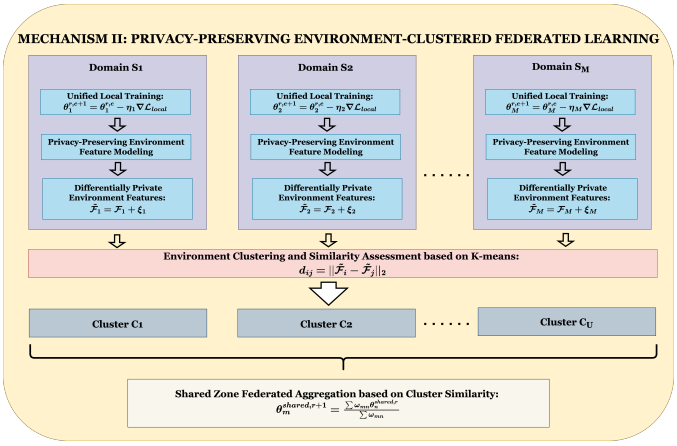


Fig. 3: Privacy-preserving environment-clustered federated learning mechanism illustrating unified local training, differential privacy protection, environment clustering, and shared zone aggregation.

To handle the dynamic nature of scheduling environments, we implement a drift-based adaptation mechanism. We periodically compute the drift for each domain:

$$\text{Drift}_m^{(r)} = ||\mathcal{F}_m^{(r)} - \mathcal{F}_m^{(r-\Delta r)}||_2. \tag{37}$$

When the maximum drift across all domains exceeds the threshold:

$$\max_m \text{Drift}_m^{(r)} > \tau_{drift}, \tag{38}$$

we trigger re-clustering to ensure that domains with substantially changed environments are appropriately reassigned to better-matching clusters.

4.2.3 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. This ensures that both shared and personalized zones are optimized jointly while maintaining model coherence:

$$\theta_m^{r,e+1} = \theta_m^{r,e} - \eta_m \nabla_{\theta_m} \mathcal{L}_{local}(\theta_m^{r,e}), \tag{39}$$

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:

$$\omega_{mn} = \begin{cases} \exp\left(-\frac{d_{mn}^2}{2\sigma_j^2}\right) & \text{if Type}_n = \text{Type}_m \text{ and } n \in C_j, \\ \beta_{fed} \cdot \exp\left(-\frac{d_{mn}^2}{2\sigma_{global}^2}\right) & \text{if Type}_n = \text{Type}_m \text{ and } n \notin C_j, \\ 0 & \text{if Type}_n \neq \text{Type}_m, \end{cases} \tag{40}$$

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:

$$\theta_m^{shared,r+1} = \frac{\sum_{n=1}^{M} \omega_{mn} \cdot \theta_n^{shared,r}}{\sum_{n=1}^{M} \omega_{mn}}. \tag{41}$$

The detailed procedure for the privacy-preserving environment-clustered federated learning is outlined in Algorithm 2. The algorithm first performs environment clustering with differential privacy protection, then alternates between local training (where 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.

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. 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.

4.3.1 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.

Algorithm 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* $\{\theta_m^{shared}, \theta_m^{pers}\}_{m=1}^M$

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:

$$\mathcal{T}_m = \{n \mid \text{Type}_n > \text{Type}_m, \text{EnvComp}(m, n) > \tau_{env}\}, \tag{42}$$

where $\text{EnvComp}(m, n)$ denotes the environmental compatibility between domains m and n:

$$\text{EnvComp}(m, n) = \exp\left(-\frac{d_{mn}^2}{2\sigma_{global}^2}\right), \tag{43}$$

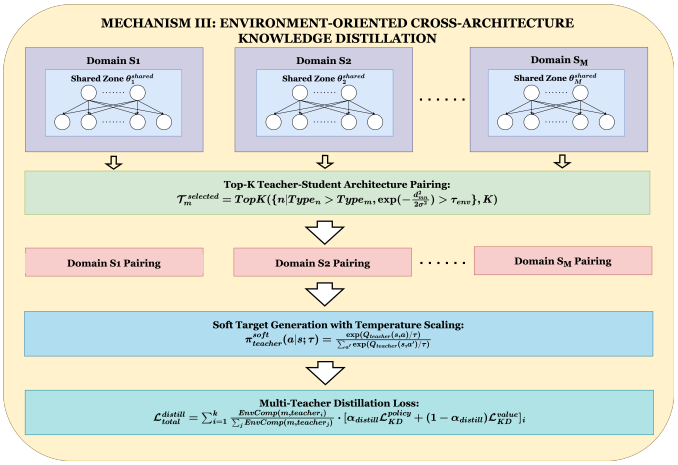


Fig. 4: Environment-oriented cross-architecture knowledge distillation mechanism demonstrating Top-K teacher-student pairing and multi-teacher distillation process.

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:

$$\mathcal{T}_m^{selected} = \text{TopK}\left(\mathcal{T}_m, K\right). \tag{44}$$

4.3.2 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:

$$\pi_{teacher}^{soft}(a|s_t;\tau_{temp}) = \frac{\exp(Q_{teacher}(s_t,a)/\tau_{temp})}{\sum_{a'\in\mathcal{A}}\exp(Q_{teacher}(s_t,a')/\tau_{temp})}. \tag{45}$$

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:

$$\mathcal{L}_{KD}^{policy} = \mathbb{E}_{s_t\sim\mathcal{D}_{student}}\Big[D_{KL}\big(\pi_{teacher}^{soft}(a|s_t;\tau_{temp}) \\ \| \ \pi_{student}(a|s_t;\tau_{temp})\big)\Big] \times \tau_{temp}^2, \tag{46}$$

where the τ_{temp}^2 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:

$$\mathcal{L}_{KD}^{value} = \mathbb{E}_{s_t\sim\mathcal{D}_{student}}\left[\left(V_{teacher}(s_t) - V_{student}(s_t)\right)^2\right]. \tag{47}$$

Adaptive Distillation Loss Balancing: To balance the contributions of policy distillation and value function distillation, the complete distillation loss function for student domains is:

$$\mathcal{L}_{distill} = \alpha_{distill}\cdot\mathcal{L}_{KD}^{policy} + (1-\alpha_{distill})\cdot\mathcal{L}_{KD}^{value}, \tag{48}$$

where $\alpha_{distill} \in [0,1]$ is a balancing parameter.

4.3.3 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:

$$w_i = \frac{\text{EnvComp}(m, teacher_i)}{\sum_{j=1}^{k}\text{EnvComp}(m, teacher_j)}. \tag{49}$$

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:

$$E_i = E_{base}\cdot\left(1+\beta_{distill}\cdot\text{EnvComp}(m, teacher_i)\right), \tag{50}$$

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:

$$\mathcal{L}_{total}^{distill} = \sum_{i=1}^{k} w_i\cdot\mathcal{L}_{distill}^{i}, \tag{51}$$

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 3. The algorithm first establishes teacher-student pairings based on architecture complexity and environmental compatibility, then performs multi-teacher dis-

tillation 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.

Algorithm 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* $\{\theta_m\}_{m=1}^{M}$

5 Performance Evaluation

This section introduces the experimental setup, hyperparameter tuning configurations, and comprehensive experiments to evaluate KD-AFRL performance.

5.1 Experiment Setup

This subsection describes the distributed multi-domain experimental environment, IoT application workloads, and baseline techniques.

5.1.1 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 independent scheduling domains across different geographical locations and infrastructure providers, as shown in Fig. 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.

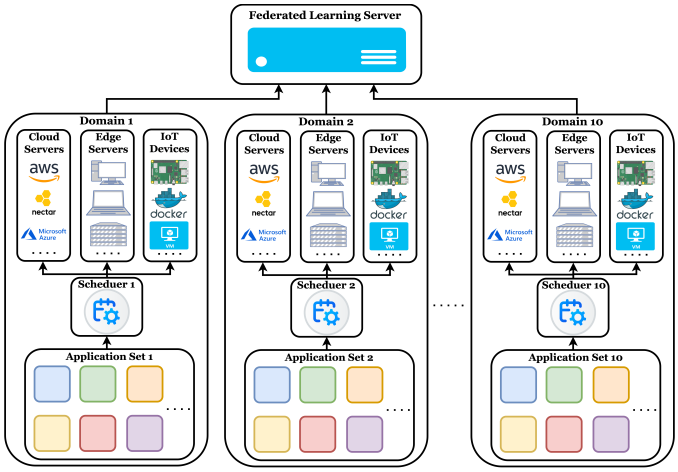


Fig. 5: KD-AFRL experimental environment

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` [28] to implement accurate real-time power measurement. In Equation 15, we set the weight parameter α_{cost} to 0.5 to balance response time and energy consumption optimization objectives.

5.1.2 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` [29]; workload scaled by analysis-window length.
- *TextSentimentAnalysis*: sentiment inference on text using

`TextBlob` [30] and `NLTK` [31]; workload scaled by text-block size.
- *SpeechRecognition*: speech-to-text inference combining `torchaudio` [32] and a lightweight Transformer decoder; workload scaled by audio-chunk length.
- *DataCompressionService*: lossless compression with `zlib` [33]/`gzip` [34]; workload scaled by file size and compression level.
- *ImageProcessor*: batch image filtering and resizing using `PIL` [35] and `OpenCV` [36]; workload scaled by batch size.
- *HealthTracker*: activity recognition via `TensorFlow Lite` [37], [38] pose estimation; workload scaled by sampling rate and model capacity.
- *FaceDetection*: face detection on streaming video with `OpenCV` [36] and `MediaPipe` [39]; workload scaled by frame resolution and detection frequency.
- *ColorTracking*: HSV-based colour tracking for AR overlays using `OpenCV` [36]; workload scaled by frame rate and tracking-window size.
- *FaceAndEyeDetection*: cascaded face-and-eye detection with landmark refinement using `OpenCV` [36] and `dlib` [40]; workload scaled by input resolution and cascade depth.
- *VideoOCR*: video text spotting (detection + recognition) with `EasyOCR` [41] and `OpenCV` [36]; 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 footprints—covering I/O-bound, CPU-bound, memory-intensive, GPU-accelerated, and network-constrained characteristics—thereby exercising KD-AFRL across realistic deployment scenarios.

5.1.3 Baseline Techniques

We implement KD-AFRL and all baseline techniques on the ReinFog platform [42] 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 [26]. 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 [6]. 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 [21]. 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 ([22], [23], [24], [25]), ADTO provides a representative benchmark for evaluating A3C-based solutions.
- **DRLIS**: The extended version of the technique proposed in [16]. 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 [43], and given that PG algorithms are commonly used in current literature ([13], [14], [17], [18]), DRLIS provides a representative benchmark for evaluating PG-based solutions.

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 II summarizes the key hyperparameter settings. Baseline methods use identical tuning procedures.

TABLE II: 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

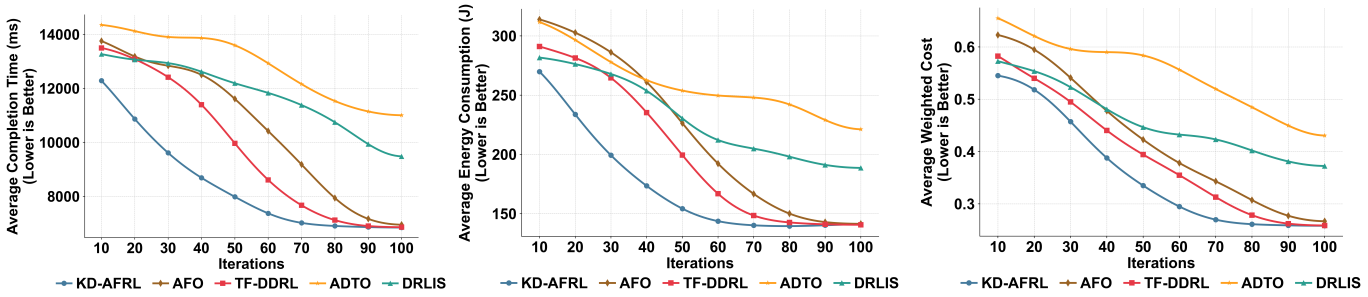
5.3 Experimental Results and Analysis

The following subsections systematically analyze convergence performance, federated learning and knowledge distillation contributions, adaptive architecture effectiveness, and framework scalability.

5.3.1 Convergence Performance Analysis

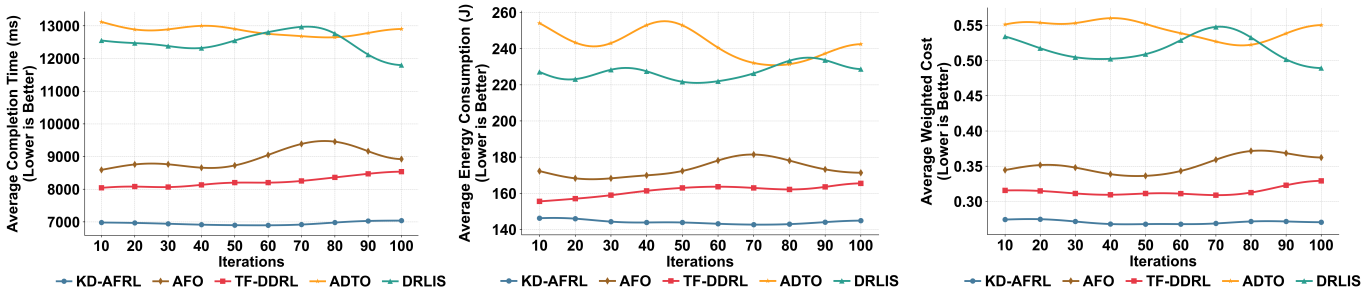
To evaluate the learning efficiency and convergence characteristics of KD-AFRL, we design the experiment comprising training and evaluation phases. The training phase employs a dedicated set of training applications for policy learning and parameter updates, while the evaluation phase freezes the learning 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 the metrics 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), 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 iterations. In the evaluation phase (Fig. 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



(a) Average completion time (b) Average energy consumption (c) Average weighted cost

Fig. 6: Convergence performance analysis during training phase



(a) Average completion time (b) Average energy consumption (c) Average weighted cost

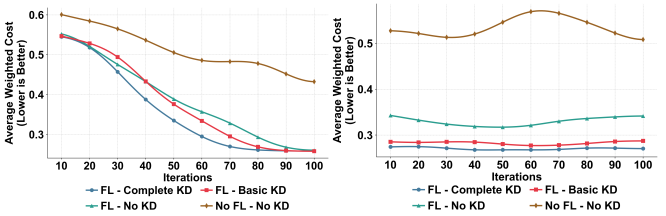
Fig. 7: Convergence performance analysis during evaluation phase

domains share knowledge, and cross-architecture distillation that enables resource-constrained devices to learn from more capable ones.

5.3.2 Federated Learning and Knowledge Distillation Analysis

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. 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 - Complete KD strategy. As illustrated in Fig. 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



(a) Training phase (b) Evaluation phase

Fig. 8: Performance comparison of different learning strategies during training and evaluation phases

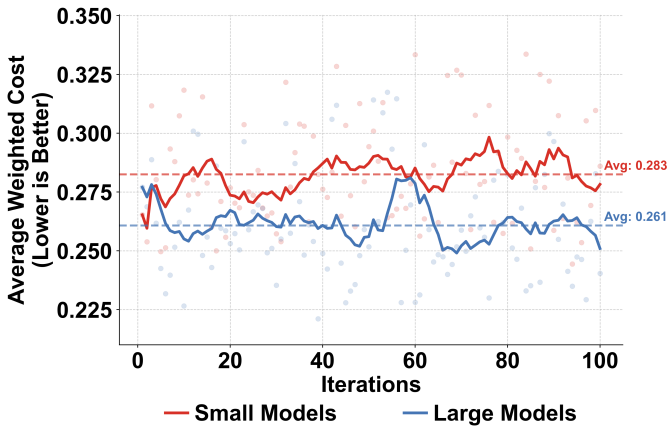
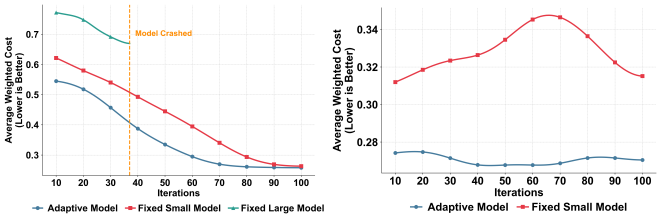


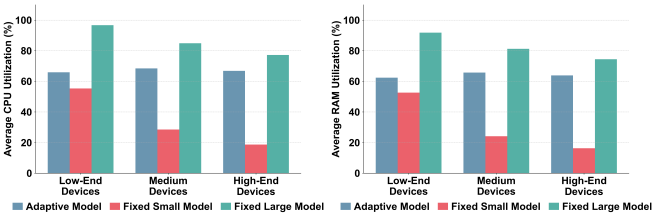
Fig. 9: Cross-architecture knowledge distillation performance between small and large models

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.



(a) Training phase (b) Evaluation phase

Fig. 10: Performance comparison of adaptive, fixed small, and fixed large models during training and evaluation phases



(a) CPU resource utilization (b) RAM resource utilization

Fig. 11: Resource utilization comparison across heterogeneous devices

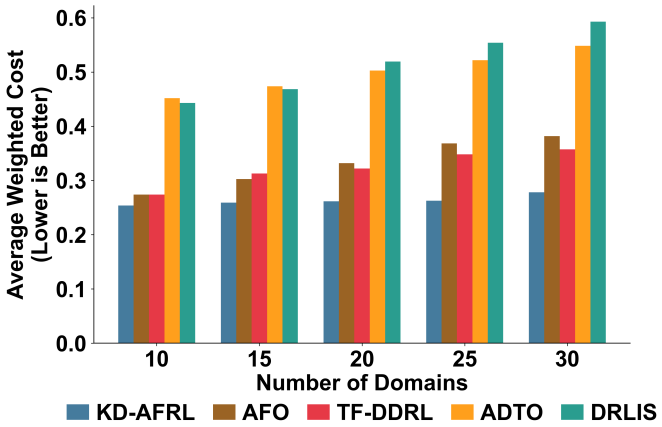


Fig. 12: Scalability performance comparison across varying number of domains

5.3.3 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.

The training and evaluation phase results demonstrate the superior stability and convergence characteristics of the adaptive model. As shown in Fig. 10, 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 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. 11 shows the adaptive model achieves balanced resource 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.

5.3.4 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. 12, 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 federated 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 Conclusions and Future Work

In this paper, we propose KD-AFRL, a Knowledge Distillation-empowered Adaptive Federated Reinforcement Learning framework for multi-domain IoT application scheduling. The framework addresses existing limitations through three core innovations: resource-aware hybrid architecture generation, privacy-preserving environment-clustered federated learning, and environment-oriented cross-architecture knowledge distillation. Experimental evaluation demonstrates KD-AFRL achieves 21% faster convergence and performance improvements of 15.7%, 10.8%, and 13.9% in completion time, energy consumption, and weighted cost respectively, compared to the best baseline. Scalability experiments demonstrate that KD-AFRL achieves 3-5 times better performance retention compared to existing solutions as domains scale.

Future research directions include developing intelligent temperature regulation mechanisms for dynamic knowledge distillation optimization, exploring advanced privacy protection mechanisms such as homomorphic encryption and blockchain-based trust systems, and investigating automated neural architecture search techniques for dual-zone design optimization.

References

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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*, 2025.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] C. Wu, Z. Xu, X. He, Q. Lou, Y. Xia, and S. Huang, "Proactive caching with distributed deep reinforcement learning in 6 g cloud-edge collaboration computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 35, no. 8, pp. 1387–1399, 2024.

[21] 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.

[22] 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.

[23] Z. Zhang, F. Zhang, Z. Xiong, K. Zhang, and D. Chen, "Lsia3cs: Deep reinforcement learning-based cloud-edge collaborative task scheduling in large-scale iiot," *IEEE Internet of Things Journal*, vol. 11, no. 13, pp. 23 917–23 930, 2024.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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*, vol. 106, no. Suppl 1. Springer, 2022, pp. S118–S128.

[29] 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.

[30] S. Loria *et al.*, "textblob documentation," *Release 0.15*, vol. 2, no. 8, p. 269, 2018.

[31] S. Bird, "Nltk: the natural language toolkit," in *Proceedings of the COLING/ACL 2006 interactive presentation sessions*, 2006, pp. 69–72.

[32] 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.

[33] J.-l. Gailly and M. Adler, "zlib: A massively spiffy yet delicately unobtrusive compression library," https://www.zlib.net/, 1995.

[34] P. Deutsch, "Gzip file format specification version 4.3," https://tools.ietf.org/html/rfc1952, RFC 1952, Internet Engineering Task Force, Tech. Rep., 1996.

[35] A. Clark and Contributors, "Pillow (pil fork) documentation," https://pillow.readthedocs.io/, 2015.

[36] G. Bradski, "The opencv library," *Dr. Dobb's Journal of Software Tools*, 2000.

[37] 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.

[38] TensorFlow Team, "Tensorflow lite: On-device machine learning framework," https://www.tensorflow.org/lite, 2017.

[39] 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.

[40] D. E. King, "Dlib-ml: A machine learning toolkit," http://dlib.net/, pp. 1755–1758, 2009.

[41] JaidedAI, "Easyocr: Ready-to-use ocr with 80+ supported languages," https://github.com/JaidedAI/EasyOCR, 2020.

[42] 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, pp. 1–20, 2025.

[43] 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.