IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 5, Issue 5, November 2025
Impact Factor: 7.67

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

Kartik Kapse[1], Rohit Chayal[2], Rohit Ghallal[3], Prof. S. S. Medhe[4]
Students, Department of Computer Engineering[1 2 3]
Professor, Department of Computer Engineering[4]
Adsul Technical Campus, Chas, Ahilyanagar, Maharashtra, India

Abstract: *The growing IoT landscape requires effective server deployment strategies to meet demands including real-time processing and energy efficiency. This is complicated by heterogeneous, dynamic applications and servers. To address these challenges, we propose ReinFog, a modular distributed software empowered with Deep Reinforcement Learning (DRL) for adaptive resource management across edge/fog and cloud environments. ReinFog enables the practical development/deployment of various centralized and distributed DRL techniques for resource management in edge/fog and cloud computing environments. It also supports integrating native and library-based DRL techniques for diverse IoT application scheduling objectives. Additionally, ReinFog allows for customizing deployment configurations for different DRL techniques, including the number and placement of DRL Learners and DRL Workers in large-scale distributed systems. Besides, we propose a novel Memetic Algorithm for DRL Component (e.g., DRL Learners and DRL Workers) Placement in ReinFog named MADCP, which combines the strengths of Genetic Algorithm, Firefly Algorithm, and Particle Swarm Optimization. Experiments reveal that the DRL mechanisms developed within ReinFog have significantly enhanced both centralized and distributed DRL techniques implementation. These advancements have resulted in notable improvements in IoT application performance, reducing response time by 45%, energy consumption by 39%, and weighted cost by 37%, while maintaining minimal scheduling overhead. Additionally, ReinFog exhibits remarkable scalability, with a rise in DRL Workers from 1 to 30 causing only a 0.3-second increase in startup time and around 2 MB more RAM per Worker. The proposed MADCP for DRL component placement further accelerates the convergence rate of DRL techniques by up to 38%.*

Keywords: Internet of Things Edge computing Fog computing Cloud computing Distributed software systems Deep Reinforcement Learning

I. INTRODUCTION

The evolution of computing paradigms has led to the emergence of edge/fog and cloud computing as complementary approaches to address the growing demands of modern applications (Jamil et al., 2022). Edge/fog computing refers to the deployment of computational resources closer to data sources, such as IoT devices, sensors, and gateways, to reduce latency and improve real-time processing capa-bilities (Meruje Ferreira et al., 2024). This distributed network of nodes performs data processing, storage, and decision-making near the data origin, minimizing the need to send large volumes of data to distant servers. By processing tasks locally or in proximity, edge/fog computing helps to reduce latency and enhance the responsiveness of time-sensitive applications. In contrast, cloud computing offers cen-tralized resources in

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 5, Issue 5, November 2025 Impact Factor: 7.67

remote data centers, providing vast storage and processing capabilities ideal for handling large-scale data and com-plex computations, albeit with potentially higher latency for real-time applications (Mansouri and Babar, 2021).

The synergy between edge/fog and cloud computing creates a pow-erful infrastructure capable of supporting various applications with varying requirements. This integrated approach allows critical applica-tions to be processed closer to the data source for faster response times, while less time-sensitive and resource-intensive applications can be offloaded to the cloud. Such a hybrid model is particularly crucial in the context of the Internet of Things (IoT), where applications are growing at an unprecedented rate (Jeyaraj et al., 2023). This has led to a signifi-cant increase in data generation and processing demands, necessitating efficient deployment strategies that can effectively leverage the compu-tational capabilities across edge/fog and cloud environments (Jin et al., 2023).

As IoT applications continue to evolve and expand, they present unique challenges for resource management and scheduling (Goudarzi et al., 2022). The heterogeneity of IoT devices and edge/fog and cloud environments, ranging from resource-constrained sensors to pow-erful cloud servers, is characterized by varying hardware capabili-ties (e.g., processing power, memory, and energy constraints), net-work conditions (e.g., latency, bandwidth, and reliability), and applica-tion requirements (e.g., real-time processing versus compute-intensive tasks) (Buyya et al., 2023). This diverse landscape makes rule-based resource scheduling strategies ineffective (Goudarzi et al., 2024). More-over, the dynamic nature of IoT workloads and network conditions requires adaptive resource management solutions capable of respond-ing to rapid and unpredictable changes (Sharif et al., 2024). The exponential growth in the number of IoT devices and applications further compounds these issues, demanding highly scalable manage-ment approaches. Additionally, IoT applications often have conflict-ing requirements, such as minimizing latency while maximizing en-ergy efficiency (Ali et al., 2023), necessitating complex multi-objective optimization strategies.

Traditional heuristic or rule-based approaches to resource manage-ment often rely on static optimization or pre-defined rules, which can be effective in predictable environments but fall short in adapting to the rapidly changing conditions of IoT ecosystems (Chen et al., 2021). These methods struggle to optimize multiple objectives such as minimizing response time and reducing energy consumption (Wang et al., 2024), particularly as the complexity of decision-making in-creases exponentially with the scale of the system (Huang et al., 2024). For instance, in real-time video processing applications deployed across edge/fog and cloud nodes, the system must continuously adapt to fluc-tuating bandwidth, latency, and processing power (Wang et al., 2022). Heuristic methods may struggle to balance these dynamic trade-offs, often resulting in suboptimal resource scheduling and increased system latency. In contrast, Deep Reinforcement Learning (DRL) techniques of-fer a more adaptive and scalable solution by continuously learning and optimizing resource management decisions based on real-time feedback from the environment (Zhou et al., 2022). This enables DRL-based approaches to dynamically schedule IoT applications, predict future workloads, and efficiently utilize available resources, outperforming traditional methods in complex and unpredictable IoT environments.

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

techniques, which is essential for efficient resource management across dynamic and stochastic edge/fog and cloud en-vironments. Second, existing frameworks cannot support both native DRL technique implementations and external DRL library integrations, limiting the flexibility and adaptability of DRL-based solutions in het-erogeneous computing environments. These limitations highlight the urgent need for a unified framework that can effectively leverage various DRL techniques for IoT application scheduling in edge/fog and cloud computing environments.

To address these challenges, we propose ReinFog, a novel frame-work that harnesses the power of DRL for adaptive resource man-agement in edge/fog and cloud computing environments. To the best of our knowledge, ReinFog is the first framework that comprehen-sively integrates mechanisms for the integration of both centralized and distributed DRL techniques for IoT application scheduling, while supporting both native DRL implementations and external DRL li-brary integrations through a modular and extensible design. To address the dynamic nature of IoT ecosystems, ReinFog incorporates mul-tiple DRL techniques, both centralized and distributed, to adapt torapidly changing

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 5, Issue 5, November 2025 Impact Factor: 7.67

workloads and network conditions, enabling real-time, intelligent IoT application scheduling decisions that optimize multiple objectives simultaneously. ReinFog supports native DRL tech-nique implementations, enabling researchers to design and develop centralized and distributed DRL techniques, specifically designed for edge/fog and cloud computing environments. Also, recognizing that each DRL library incorporates a set of specific DRL techniques, ReinFog offers mechanisms to seamlessly integrate external DRL libraries. This dual capability enhances the flexibility and adaptability of DRL-based solutions in heterogeneous computing environments, allowing users to leverage familiar tools and accelerate their development processes. Accordingly, to facilitate the implementation of both native and library-based DRL techniques, ReinFog adopts a modular design that supports easy extension and customization. This design separates DRL compo-nents into DRL Workers for environment interaction and DRL Learners for policy optimization. To accommodate the diverse requirements of large-scale distributed systems, ReinFog supports customizable de-ployment configurations for DRL techniques. This design allows for flexible configuration of DRL Learners and DRL Workers, enabling users to tailor deployments according to specific system architectures and performance needs. Such flexibility is crucial for effectively managing resources and ensuring optimal performance in complex IoT environ-ments with varying scales and topologies. Moreover, as the efficient execution of different DRL techniques requires interaction among mul-tiple DRL-related components, it is crucial to place these components on appropriate nodes. To optimize the DRL component placement, we propose a novel Memetic Algorithm for DRL Component Placement in ReinFog, named MADCP. MADCP combines the Genetic Algorithm (GA)'s robust exploration capabilities, Firefly Algorithm (FA)'s ability to fine-tune local search, and Particle Swarm Optimization (PSO)'s efficient global optimization to efficiently place DRL components across heterogeneous computing nodes. This algorithm enhances ReinFog's ability to quickly adapt to changing environmental conditions and optimize resource utilization before the start of DRL training processes.

The key contributions of our paper are as follows:

• We propose ReinFog, a containerized and modular framework for DRL-based resource management in edge/fog and cloud envi-ronments. It offers mechanisms to support both centralized and distributed DRL techniques. Also, it enables the integration of both native DRL techniques and external DRL libraries.

• We design customizable deployment configurations for DRL tech-niques in ReinFog, allowing flexible configuration of DRL Learn-ers and DRL Workers in large-scale distributed systems.

• We propose a novel Memetic Algorithm for DRL Component Placement in ReinFog, named MADCP, combining GA, FA, and PSO for efficient DRL component placement.

• We conduct extensive practical experiments evaluating ReinFog's performance across various aspects. It demonstrates that Rein-Fog is a lightweight and scalable framework capable of effec-tively scheduling IoT applications under diverse optimization objectives.

II. RELATED WORK

IoT application scheduling and resource management in edge, fog, and cloud environments have attracted significant research attention. Existing approaches can be broadly categorized into two groups: (i) algorithmic techniques that focus on optimizing scheduling decisions using heuristics, meta-heuristics, or DRL, and (ii) system-level soft-ware frameworks that aim to support the practical deployment and management of IoT applications. In this section, we first review algo-rithmic techniques, including both heuristic/meta-heuristic and DRL-based methods. Then, we summarize relevant software frameworks and highlight their limitations in handling dynamic, large-scale, and het-erogeneous environments. Finally, we provide a comparative analysis to clearly position the originality and technical contributions of our proposed ReinFog framework.

2.1. Algorithmic techniques for IoT scheduling

A wide range of algorithmic techniques have been proposed to address the challenges of scheduling and resource management in IoT-enabled edge, fog, and cloud environments. These techniques can be broadly classified into heuristic/meta-heuristic methods and ma-chine learning-based techniques. In terms of heuristic techniques, Wu et al. (2018) modeled IoT application scheduling in edge and fog environments as a Directed Acyclic Graph (DAG), using

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 5, Issue 5, November 2025 Impact Factor: 7.67

EDA and partitioning to queue IoT applications and assign servers. Ali et al. (2020) proposed an NSGA2-based technique for minimizing the to-tal computation time and system cost of IoT application scheduling in heterogeneous fog cloud computing environments.

Hoseiny et al. (2021) proposed a GA-based technique to minimize computation time and energy consumption in heterogeneous fog-cloud IoT application scheduling. While these heuristic methods perform well in specific scenarios, they often lack adaptability to dynamic environments. In re-cent years, machine learning techniques, particularly DRL, have gained significant attention for resource management in edge/fog and cloud computing environments, owing to their adaptability and capacity for continuous learning in dynamic scenarios. Huang et al. (2019) applied a Deep Q-Network (DQN)-based approach to address resource allocation problems within edge computing environments. Zheng et al. (2022) proposed a Soft Actor-Critic (SAC)-based technique to solve the opti-mization problem of computational offloading and resource allocation in collaborative vehicle networks. Siyadatzadeh et al. (2023) proposed ReLIEF, which employs Q-Learning to manage resources in fog-based IoT systems. Wang et al. (2024) proposed DRLIS, which leverages Proximal Policy Optimization (PPO) to optimize system load balancing and response time in edge and fog computing environments. Zou et al. (2020) proposed A3C-DO, which utilizes the Asynchronous Advantage Actor-Critic (A3C) technique to manage resources in edge computing environments. Liu et al. (2023) proposed an A3C-based approach for edge computing in the smart vehicles domain. Wang et al. (2025) pro-posed TF-DDRL, which is based on Importance Weighted Actor-Learner Architectures (IMPALA) to schedule IoT applications under three opti-mization objectives. Table 1 presents a qualitative analysis of existing techniques proposed for IoT application scheduling. While prior stud-ies focus on DRL-based scheduling, none provide a unified software framework that facilitates the implementation of both centralized and distributed DRL techniques, which is essential for experimental and practical deployment. These limitations are overcome by our proposed ReinFog software framework.

2.2. Frameworks for resource management

Building upon these techniques, researchers have developed several frameworks for resource management in edge/fog and cloud computing environments. Many of these frameworks employ heuristic or meta-heuristic techniques for making resource management decisions. For instance, Yigitoglu et al. (2017) developed Foggy, a container-enabled framework supporting policy and rule-based scheduling of container-ized IoT applications with dependent tasks. Similarly, Merlino et al. (2019) proposed a framework allowing policy-driven vertical and hor-izontal task offloading. Yousefpour et al. (2019) introduced FogPlan, which employs greedy algorithms to minimize IoT application response time, while Ghosh et al. (2019) developed Mobi-IoST, using a prob-abilistic approach for IoT application scheduling. Deng et al. (2021) created FogBus2, introducing a GA-based IoT application scheduling technique, and Pallewatta et al. (2024) proposed MicroFog, integrating multiple heuristic algorithms to enhance IoT application scheduling flexibility. In recent years, some frameworks have started to incorporate reinforcement learning techniques. N. Toosi et al. (2022) developed GreenFog, which combines linear programming optimization with the Multi-Armed Bandit approach for energy consumption reduction. Sim-ilarly, Nkenyereye et al. (2023) proposed CEIF, which adopted Deep Q-Learning for resource management in edge computing environments.

2.3. Summary and technical comparison with existing frameworks

Table 2 identifies the main properties of the related frameworks and compares them with ReinFog. Environment Support indicates the com-puting environments supported by each framework. DRL-Integrated Framework indicates whether the framework is comprehensively inte-grated with DRL capabilities, encompassing multiple DRL techniques and providing support for extensibility. The DRL Capabilities section is a specific contribution of ReinFog. These capabilities enable more adaptive and intelligent resource management by leveraging DRL to dynamically optimize IoT application scheduling in real time. It is further divided into three sub-categories. Mechanism indicates whether the framework supports the integration of native DRL techniques or importing external libraries. Architecture shows whether the frame-work supports centralized and distributed DRL techniques. Finally, DRL Component Placement shows whether the framework can auto-matically optimize the placement of DRL components. The Generic Capabilities section represents general requirements for frameworks in edge/fog or cloud computing environments.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 5, Issue 5, November 2025 Impact Factor: 7.67

These are commonly expected features that any robust framework should provide to ensure flexibility, adaptability, and ease of integration across various platforms and environments, especially in addressing the challenges posed by heterogeneous systems. It is further divided into five sub-categories. Multi-platform Support shows whether the framework can operate across diverse heterogeneous hardware and software platforms. Con-tainer Support refers to the ability to use containerization technologies. Scalability, Configurability, and Extensibility assess the framework's ability to be scaled, customized, and incorporate new features.

ReinFog offers significant advantages over related frameworks across multiple dimensions. Many existing frameworks rely on tradi-tional heuristic (Yigitoglu et al., 2017; Merlino et al., 2019; Yousefpour et al., 2019; Ghosh et al., 2019; Pallewatta et al., 2024) or meta-heuristic (Deng et al., 2021) techniques for IoT application scheduling, which often lack adaptability to manage dynamic and complex comput-ing environments. Although some recent frameworks have started in-corporating basic and single reinforcement learning techniques, such as Multi-Armed Bandit (N. Toosi et al., 2022) or Deep Q-Learning (Nkeny-ereye et al., 2023), these frameworks do not offer mechanisms for in-tegration/development of centralized and distributed DRL techniques. Accordingly, these frameworks struggle with the implementation of ef-ficient and scalable DRL techniques for large-scale deployments. To the best of our knowledge, ReinFog is the first resource management frame-work that enables the integration of both centralized and distributed DRL mechanisms for IoT application scheduling across edge/fog and cloud environments. These mechanisms enable the integration/devel-opment of a wide range of centralized and distributed techniques such as PPO (Schulman et al., 2017) and IMPALA (Espeholt et al., 2018). Besides, ReinFog offers interfaces for the integration of both native and library-based DRL techniques. Notably, ReinFog introduces DRL component placement, a novel feature to customize and optimize the placement of DRL components using multiple meta-heuristic algorithms and proposed MADCP. These comprehensive features and capabilities make ReinFog a versatile platform for researchers, enabling them to either utilize built-in DRL techniques or extend its mechanisms for various resource management scenarios in edge/fog and cloud computing environments. In terms of generic capabilities, ReinFog also offers multi-platform support, a feature lacking in several frameworks (e.g., Yousefpour et al. (2019), Ghosh et al. (2019) and N. Toosi et al. (2022)). Besides, ReinFog is designed with scalability, configurability, and extensibility in mind, addressing limitations found in many existing frameworks (Yigitoglu et al., 2017; Merlino et al., 2019; Yousefpour et al., 2019; Ghosh et al., 2019; N. Toosi et al., 2022; Nkenyereye et al., 2023).

III. REINFOG FRAMEWORK ARCHITECTURE

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

3.1. Hardware environment

ReinFog is designed to operate across a heterogeneous multi-layered hardware environment, as illustrated in Fig. 1. This environment en-compasses three primary layers: Cloud, Edge/Fog, and IoT, each with distinct characteristics and roles in the overall system.

3.1.1. Cloud layer

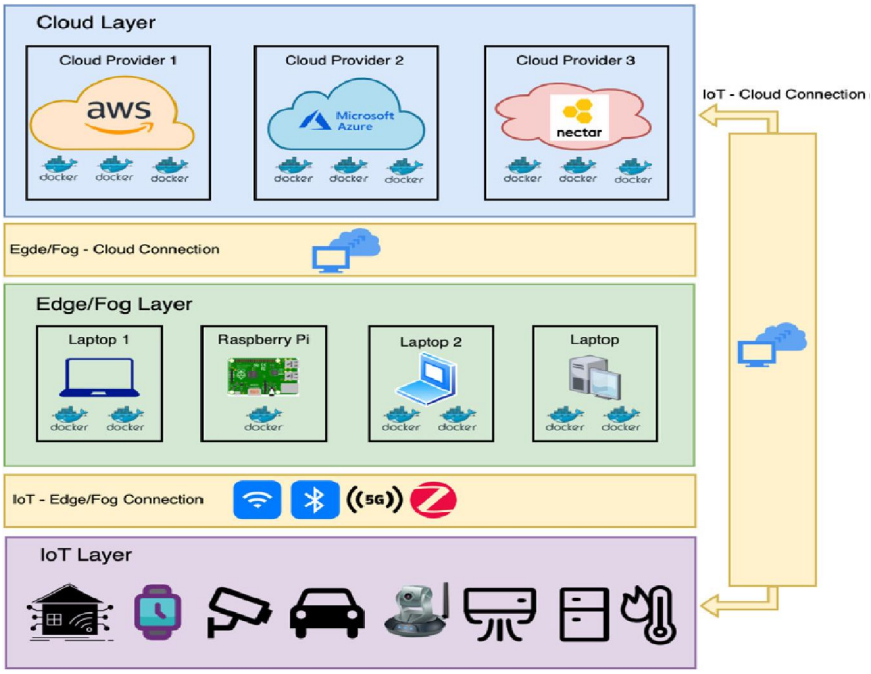
The Cloud Layer represents the highest tier of computing resources in the ReinFog hardware environment. It consists of high-performance servers provided by different cloud service providers such as Amazon Web Services (AWS), Microsoft Azure, and Nectar. These cloud en-vironments offer scalable computing power and storage capabilities, high reliability and availability, and advanced services for data ana-lytics and machine learning. The cloud layer supports containerization for consistent deployment of ReinFog components and typically han-dles computationally intensive tasks, large-scale data processing, and long-term data storage.

3.1.2. Edge/fog layer

The Edge/Fog Layer serves as an intermediate computing tier be-tween the cloud and IoT devices. This layer comprises various com-puting devices, including laptops, desktops with varying computational capabilities, and single-board

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 5, Issue 5, November 2025
Impact Factor: 7.67

computers (e.g., Raspberry Pi). These devices are strategically positioned closer to the data source, enabling reduced latency for time-sensitive applications, local data processing and filtering, and improved privacy and security by keeping sensitive data local. The Edge/Fog layer connects to the Cloud layer via Internet connections, allowing for seamless data transfer and task offloading when needed. It also supports containerization for flexible deployment of ReinFog software components, playing a crucial role in facilitating real-time applications and reducing the computational burden on both the cloud and IoT devices.



3.1.3. IoT layer

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

3.2. Software architecture

The software architecture of ReinFog is designed to efficiently man-age and schedule IoT tasks in heterogeneous edge/fog and cloud en-vironments. As illustrated in Fig. 2, the framework consists of two primary subsystems, along with user interfaces for IoT application submission.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 5, Issue 5, November 2025
Impact Factor: 7.67

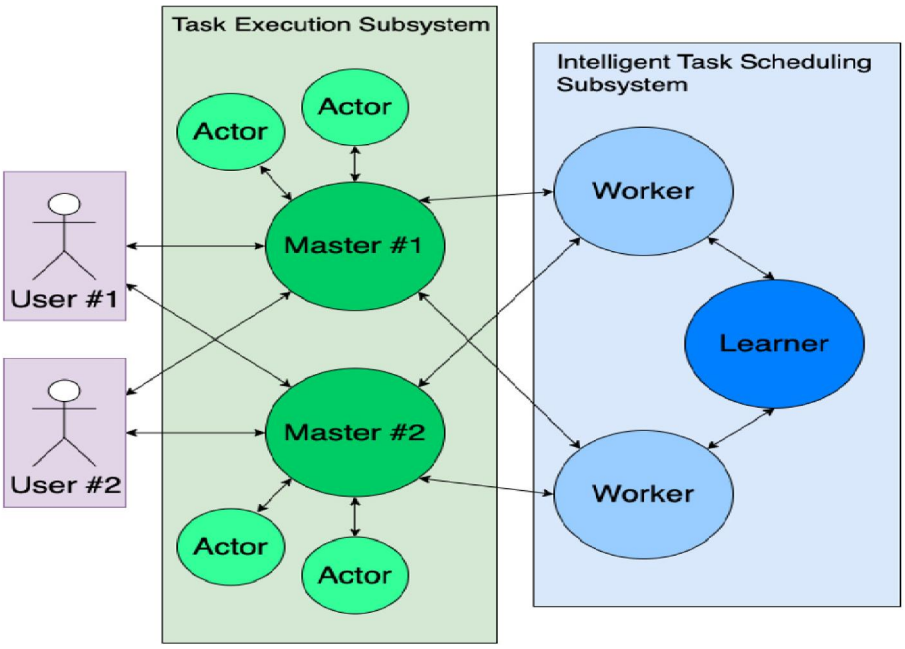


Fig. 2. High-level software architecture of ReinFog

3.2.1. Task execution subsystem

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

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

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

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

3.2.2. Intelligent task scheduling subsystem

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

Worker: Workers gather system state information from the Task Execution Subsystem and generate IoT task scheduling decisions based on the learned policy. They communicate these decisions back to the Task Execution Subsystem for implementation. Work-ers continuously refine their decision-making process, adapting to the system's dynamic nature through distributed learning.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 5, Issue 5, November 2025 Impact Factor: 7.67

Learner: Centrally located, the Learner aggregates experiences or gradients from Workers and optimizes the global DRL technique. The Learner periodically distributes the refined policy to all Work-ers, ensuring system-wide improvement in decision-making. This continuous optimization enables ReinFog to adapt to changing conditions and enhance overall IoT task scheduling efficiency.

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

3.2.3. ReinFog operation workflow

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

This architecture allows ReinFog to balance the load across avail-able resources and optimize overall system performance. By integrat-ing DRL-based decision-making with traditional task execution mech-anisms, ReinFog can effectively adapt to the complex and dynamic nature of modern IoT, edge/fog and cloud computing environments, providing a robust and efficient solution for resource management and IoT application scheduling.

IV. REINFOG DESIGN AND IMPLEMENTATION

To achieve adaptive resource management in edge/fog and cloud environments, we implement the ReinFog framework by extending the core components of the FogBus2 framework and implementing and integrating the new Intelligent Task Scheduling Subsystem. We chose FogBus2 as the foundation for ReinFog because it already imple-ments many of the basic functionalities required in our Task Execution Subsystem.

FogBus2 is a lightweight and distributed container-based framework for integrating heterogeneous IoT systems with edge/fog and cloud environments. It comprises five main components that align well with our Task Execution Subsystem requirements:

- User: handles environmental data collection and actuation con-trol, similar to our user interface for IoT application submission.
- Master: manages IoT applications and scheduling, which forms the basis of our Master component in the Task Execution Subsys-tem.
- Actor: performs host resource profiling, aligning with our Actor component's responsibilities.
- Task Executor: executes submitted IoT applications, fitting di-rectly into our execution model.
- Remote Logger: provides persistent log storage, supporting our system's monitoring and analysis needs.

By leveraging FogBus2, we are able to focus our efforts on de-veloping our DRL-based scheduling subsystem to handle complex, dependency-aware IoT applications. This approach allowed us to build upon a proven foundation while integrating our novel Intelligent Task Scheduling Subsystem with its DRL capabilities. The overall design of ReinFog, showing both the extended FogBus2 components and our new DRL components, is illustrated in Fig. 3.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 5, Issue 5, November 2025 Impact Factor: 7.67

4.1. ReinFog DRL components

In this section, we present the design and organization of our DRL components: the DRL Learner and the DRL Worker. Each compo-nent is modularly designed and encompasses several sub-components and modules, which interact through well-defined APIs and internal messages. The overall design is depicted in Fig. 4.

4.1.1. DRL learner

The DRL Learner component is responsible for training the DRL techniques and managing the learning process. It comprises the follow-ing sub-components:

Learner DRL core: The Learner DRL Core is central to policy learning and optimization. It includes the following modules:

1. Exploration Engine: This module implements and manages exploration strategies during the training process to balance exploration and exploitation. It incorporates established meth-ods such as ϵ-greedy and Ornstein–Uhlenbeck process noise, while also providing an extensible mechanism for integrating additional exploration algorithms. The Exploration Engine ex-poses a well-defined API, allowing other components to interact with it efficiently. Through this API, it dynamically interacts with the Network Architecture Hub and the DRL Technique Repository via the DRL Policy Constructor, enabling it to access and utilize current technique states and network architectures. This API-driven design enhances modularity and facilitates easy integration of new exploration strategies.

2. Network Architecture Hub: This module manages and main-tains base neural networks for various DRL techniques. It sup-ports a wide range of architectures including:

- Deep Neural Networks (DNNs)
- Recurrent Neural Networks (RNNs)
- Long Short-Term Memory Networks (LSTMs)
- Transformers

These diverse architectures enable the hub to accommodate different types of tasks. The module exposes APIs for querying and updating network architectures, facilitating seamless inte-gration and dynamic adaptability. Designed with extensibility in mind, it allows for easy incorporation of new neural network architectures and policy optimization methods in the future.

3. DRL Technique Repository: This module efficiently manages and maintains various DRL techniques. It is divided into Cen-tralized Techniques and Distributed Techniques sub-modules, accommodating different user requirements and computational architectures. The repository provides two integration mecha-nisms. The native integration mechanism enables direct imple-mentation of DRL techniques within the framework, offering optimal performance and full customization flexibility. ReinFog already includes native implementations of several representa-tive DRL techniques, such as DQN, PPO, A3C, and IMPALA. In parallel, the library-based integration mechanism enables seam-less incorporation of external DRL libraries through standardized interfaces. ReinFog has already integrated techniques including SAC and R2D2 via the Ray library. This dual-mechanism design allows developers to either build custom DRL techniques within the framework or directly leverage existing implementations, en-suring flexibility, maintainability, and performance consistency across heterogeneous computing environments.

4. DRL Policy Constructor: This module is central to building and optimizing DRL policies. It orchestrates interactions with multiple modules through well-defined APIs to create and refine effective DRL strategies. The Constructor leverages the Net-work Architecture Hub's API for base network construction and accesses the DRL Technique Repository via API for policy de-velopment. It interacts with the Exploration Engine through its API to balance exploration and exploitation during the learning process. For policy persistence, the Constructor integrates with the Policy Manager, enabling saving and loading of policies. To enhance learning efficiency, it collaborates with the Replay Buffer for experience sampling. The DRL Policy Constructor also coordinates parallel learning processes via the Worker Ses-sion Manager and ensures policy synchronization through the Synchronizer.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 5, Issue 5, November 2025 Impact Factor: 7.67

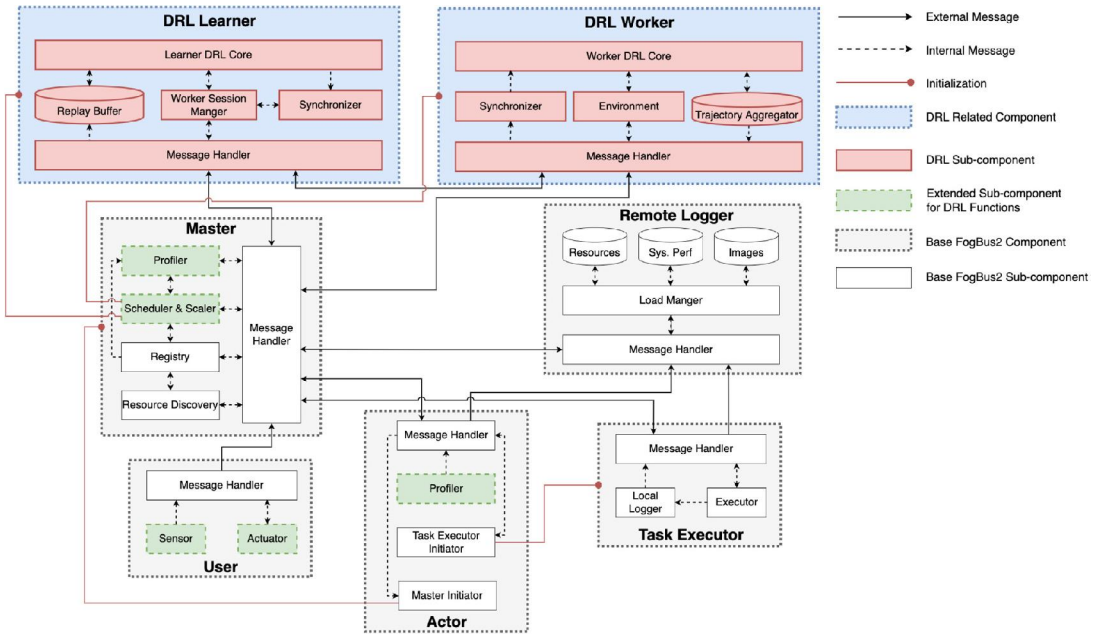


Fig. 3. ReinFog design overview.

5. Policy Manager: This module is responsible for comprehensive DRL policy administration. It consists of two key sub-modules: the Saver and the Loader. The Saver sub-module ensures version-controlled, persistent storage of optimized policies in the Trained Policy Repository. The Loader sub-module, on the other hand, retrieves policies from the Repository as needed for further refinement or evaluation. The Policy Manager interfaces directly with the DRL Policy Constructor, enabling seamless integra-tion of policy persistence operations within the overall learning process.

6. Trained Policy Repository: This module functions as a cen-tralized storage for optimized and validated DRL policies. It interfaces directly with the Policy Manager's Saver and Loader sub-modules, enabling efficient storage and retrieval of policies.

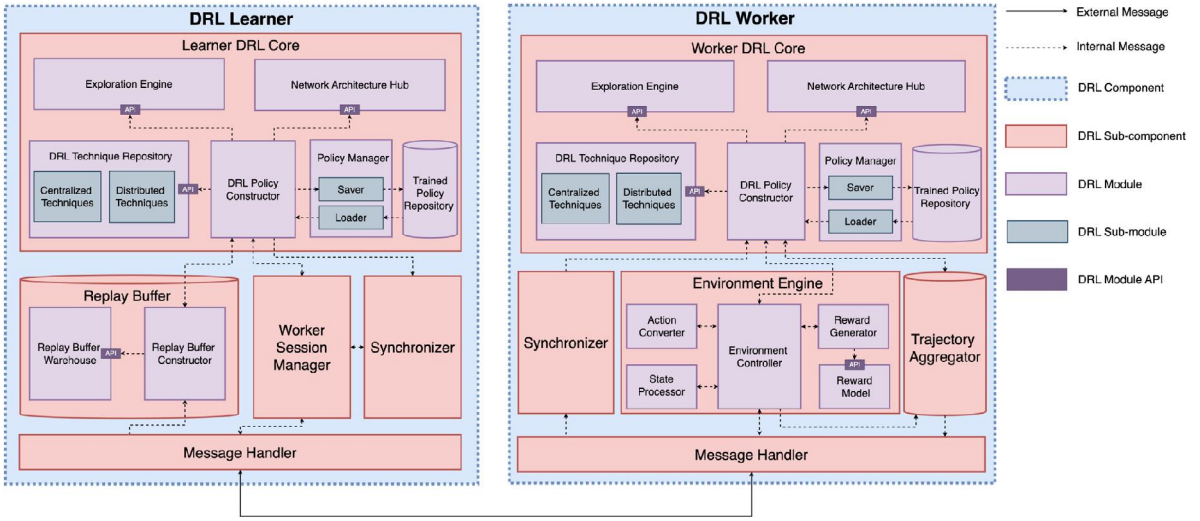


Fig. 4. Distributed DRL components design

The Repository maintains a structured archive of policies, sup-porting version control and rapid access for DRL experiments. It enhances the overall efficiency and effectiveness of policy management in the ReinFog framework.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 5, Issue 5, November 2025 Impact Factor: 7.67

Replay buffer: The Replay Buffer enhances learning efficiency and stability by storing and reusing experiences. It consists of two key modules:

1. Replay Buffer Warehouse: This module stores agent-environ-ment interaction experiences. It features both random and reser-voir sampling buffers. Random sampling helps break temporal correlations in the data, while reservoir sampling maintains a fixed-size buffer suitable for streaming or unbounded data. The Warehouse also supports flexible integration of additional sampling algorithms.

2. Replay Buffer Constructor: This module manages the creation, maintenance, and updating of buffers. It interfaces with the Re-play Buffer Warehouse through its API to facilitate the storage, retrieval, and sampling of experiences. It also interacts directly with the DRL Policy Constructor, supporting efficient experience utilization in the learning process.

Worker session manager: The Worker Session Manager is responsible for managing communication across multiple DRL Workers in a distributed environment. It handles thread creation, maintenance, and DRL Worker communication, ensuring fast and reliable transmission of data and instructions. This sub-component plays a crucial role in coordinat-ing parallel learning processes and maintaining efficient distributed operations within the ReinFog framework.

Synchronizer: The Synchronizer is tasked with synchronizing train-ing across the distributed learning environment. It coordinates pol-icy updates, gradients, and parameters between the DRL Learner and DRL Workers, maintaining a unified learning environment. Through close collaboration with the Worker Session Manager, the Synchro-nizer ensures consistency in the learning process, facilitating effective distributed learning in ReinFog.

Message handler: The Message Handler manages inter-component com-munications within the ReinFog framework. It receives and processes incoming messages, efficiently routing them to the appropriate internal sub-components. It serves as a central communication hub, facilitating effective information exchange between various components of the ReinFog framework. By ensuring smooth and organized message flow, the Message Handler maintains system coherence and optimizes overall operational efficiency.

4.1.2. DRL worker

The DRL Worker component is responsible for interacting with the environment, collecting and processing data to support policy learning and optimization within the ReinFog framework. It shares core functionalities with the DRL Learner, with the Worker DRL Core mirroring the architecture of the Learner DRL Core to ensure consis-tent functionality and coordination. Despite these similarities, the DRL Worker incorporates several unique sub-components: an Environment Engine for direct interaction with the learning environment, a Trajec-tory Aggregator for efficient collection and processing of experience data, and a Synchronizer adapted for DRL Worker-specific synchroniza-tion tasks, functioning differently from the Synchronizer in the DRL Learner. In centralized learning scenarios, the DRL Worker can assume the role of the DRL Learner, enabling direct policy updates for DRL techniques. This comprehensive design enables flexible deployment of the DRL Worker in both centralized and distributed learning con-texts, enhancing the overall adaptability and efficiency of the ReinFog framework.

Environment engine: The Environment Engine manages interactions be-tween the learning components and the environment through several key modules:

1. State Processor: This module transforms raw environmental data into a format suitable for decision-making and learning processes.

2. Action Converter: This module translates generated actions into environment-specific commands, ensuring proper execution of decisions.

3. Reward Generator: This module processes reward parameters and utilizes the Reward Model's API to calculate rewards.

4. Reward Model: This module defines reward calculation meth-ods, maintains consistency in reward generation, and supports the extension of additional reward functions.

5. Environment Controller: This module oversees the overall in-teraction process, managing the flow of actions and states, and coordinating the integration of other modules.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 5, Issue 5, November 2025 Impact Factor: 7.67

This modular design ensures a smooth and continuous interaction cycle, facilitating efficient learning and adaptation within the ReinFog framework.

Trajectory aggregator: The Trajectory Aggregator is responsible for col-lecting and processing trajectories generated from interactions with the environment. These trajectories, consisting of sequences of states, actions, and rewards, encapsulate experiential data over time. The ag-gregator maintains a well-structured and accessible repository of these experiences for training and evaluation purposes. In specific techniques such as A3C, where gradients are calculated within DRL Workers, the Trajectory Aggregator collects and transmits these gradients to the DRL Learner for updating the global policy. Moreover, in centralized learning scenarios where policies are optimized within the DRL Worker component, the Trajectory Aggregator can adapt to function as a Replay Buffer, storing and sampling trajectory data to support policy training and updates. This versatile design enables the Trajectory Aggregator to support various learning paradigms and techniques within the ReinFog framework, enhancing flexibility and efficiency in different operational contexts.

Synchronizer: The Synchronizer in the DRL Worker maintains consis-tency between local and global policies in distributed learning scenar-ios. It periodically obtains the latest policy parameters from the DRL Learner and updates the local policies accordingly. This mechanism facilitates efficient knowledge sharing, allowing DRL Workers to make decisions based on up-to-date global knowledge while contributing to the overall learning process. The Synchronizer is essential for bal-ancing local exploration and global exploitation within the ReinFog framework.

4.2. Extended FogBus2 sub-components

To enable FogBus2 to work seamlessly with DRL capabilities, we have extended several of its sub-components. This section introduces these extended sub-components, as illustrated in Fig. 3.

4.2.1. Extended scheduler & scaler

The Scheduler & Scaler sub-component has been extended to in-corporate DRL capabilities, with the primary focus on enhancing the Scheduler Module. This extension enables coordination with the newly introduced DRL components. Fig. 5 illustrates the detailed structure of the Extended Scheduler Module within the Scheduler & Scaler sub-component.

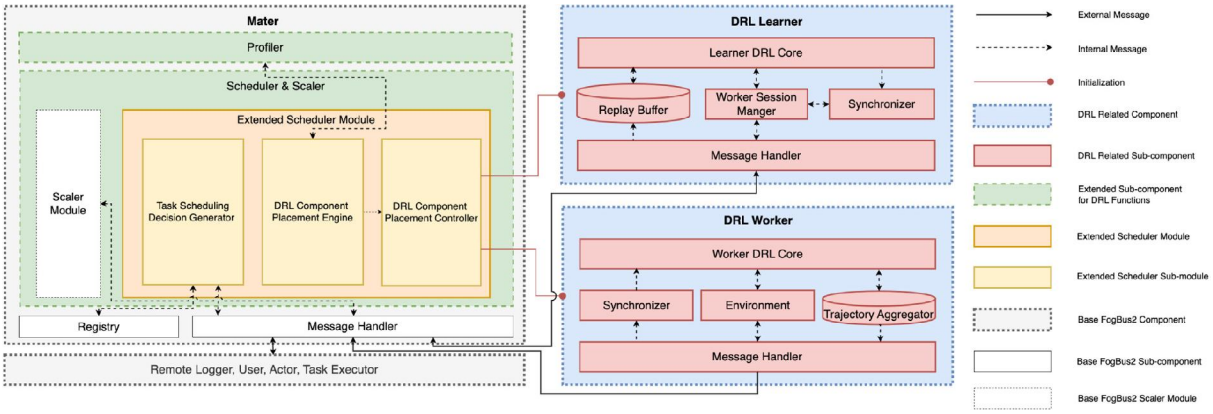


Fig. 5. Extended FogBus2 scheduler module.

DRL component placement engine: The DRL Component Placement En-gine generates strategies for placing DRL Learners and DRL Workers. It incorporates several algorithms:

- GA: Inspired by natural selection, GA evolves a population of potential solutions over generations.
- FA: Based on the flashing behavior of fireflies, FA uses attraction and movement towards brighter solutions.
- PSO: Mimicking the social behavior of bird flocking, PSO updates solutions based on personal and global best experiences.

DOI: 10.48175/IJARSCT-30062

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 5, Issue 5, November 2025 Impact Factor: 7.67

• MADCP: Our proposed algorithm that combines GA, FA, and PSO to leverage their respective strengths for more efficient DRL component placement.

In addition, the Engine supports the integration of additional algo-rithms to address diverse DRL component placement requirements and optimization objectives.

DRL component placement controller: The DRL Component Placement Controller retrieves placement strategies from the DRL Component Placement Engine and applies them to place DRL Learners and DRL Workers. This sub-module ensures that the DRL components are prop-erly set up according to the generated strategies, facilitating the effi-cient start of the learning process.

Task scheduling decision generator: The Task Scheduling Decision Gen-erator communicates with DRL components to obtain IoT task schedul-ing decisions. It then manages the deployment of these tasks based on the received decisions. This sub-module bridges the gap between DRL components and practical task execution within the ReinFog frame-work.

4.2.2. Extended profiler

The Profiler has been extended in both the Master and Actor com-ponents to provide more comprehensive system monitoring. In the Master, the extended Profiler now collects and analyzes system-wide performance metrics, providing a holistic view of the entire distributed environment. This includes monitoring resource utilization patterns, network latencies, and overall system throughput. In the Actor, the Profiler has been augmented to gather more detailed node-specific information, such as CPU and memory usage, task execution times, and local network conditions. These enhancements enable more accurate modeling of the system state, which is crucial for the DRL components to make informed IoT task scheduling decisions.

4.2.3. Extended user's sensor and actuator

The User component's Sensor and Actuator have been extended to enhance data collection and performance monitoring capabilities, supporting the integration of DRL in ReinFog. These extensions focus on gathering more detailed information about IoT device operations and application performance. Key enhancements include the ability to mon-itor detailed energy consumption patterns of IoT devices, fine-grained operational timing such as task execution and completion times, as well as device-specific characteristics like processing capabilities and storage capacity. Additionally, the extensions provide real-time data on network conditions and connectivity status of IoT devices.

4.3. Centralized and distributed deployment

ReinFog supports both centralized and distributed deployment for DRL techniques. This section details how ReinFog components are utilized in these two deployment patterns, with specific examples to illustrate the implementation of each deployment.

4.3.1. Centralized DRL techniques deployment

Centralized DRL techniques deployment is essential in ReinFog for scenarios where the environment complexity is moderate and the sys-tem can benefit from simplified architecture and reduced communica-tion overhead. In centralized deployment, a single DRL Worker handles both learning and decision-making processes, suitable for scenarios with relatively stable workload patterns or when system resources are limited.

Centralized DRL deployment scheme in ReinFog: In centralized deploy-ment, the learning process is conducted directly within the DRL Worker, eliminating the need for a separate DRL Learner. The Environment Engine collects environmental data from the ReinFog Master and processes them for training. The Trajectory Aggregator will assume the role of the Replay Buffer, which stores and samples experiences for policy optimization. The Worker DRL Core contains all the necessary modules for both learning and decision-making. This design centralizes all environmental data collection and policy updating within the DRL Worker, creating a self-contained unit that operates independently.

Sample illustration: To demonstrate the centralized deployment, we utilize DQN (Mnih et al., 2015) as an example. DQN is a widely applied DRL technique that combines Q-learning with DNNs, employing a primary network for action

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 5, Issue 5, November 2025 Impact Factor: 7.67

selection and a separate target network for value estimation to stabilize training and improve learning effi-ciency in sequential decision-making problems (Arulkumaran et al., 2017). In our implementation, the DRL Worker's Environment Engine receives environmental data from the ReinFog Master as a scheduling request and processes them into suitable formats for DQN utilization.

The Worker DRL Core maintains two neural networks: the primary network for action selection and the target network for stable Q-value estimation. Actions are selected based on the current Q-values and sent to the Master for IoT task scheduling. The scheduling results and rewards are processed by the Environment Engine and stored in the Trajectory Aggregator. The Worker DRL Core then samples mini-batches of experiences directly from the Trajectory Aggregator and performs Q-learning updates.

4.3.2. Distributed DRL techniques deployment

Distributed DRL techniques deployment is essential in ReinFog for addressing complex, large-scale environments where centralized ap-proaches may be inefficient. This deployment pattern can harness the computational capabilities of heterogeneous edge/fog and cloud resources through multiple DRL Workers collaborating with a single or multiple DRL Learners, enabling parallel training and distributed learning to deal with dynamic workloads that demand rapid adaptation.

Distributed DRL deployment scheme in ReinFog: In distributed deploy-ment, ReinFog employs multiple DRL Workers collaborating with a single or multiple DRL Learners to enable parallel training and decision-making. Each DRL Worker operates independently, with its Environ-ment Engine collecting and processing environmental data from the Master and its DRL Core generating local decisions based on the cur-rent policy. The Trajectory Aggregator in each DRL Worker collects and processes local experiences for transmission. The Worker Session Manager coordinates the communication flow, enabling DRL Workers to transmit their processed experiences to the central DRL Learner. The DRL Learner stores these experiences in its Replay Buffer. The Learner DRL Core then performs global policy optimization using experiences stored and sampled from the Replay Buffer. After policy updates, the Synchronizer ensures policy consistency by distributing the latest global policy to all DRL Workers. This distributed architecture enables scalable learning across multiple nodes while maintaining policy coherence.

Sample illustration: To illustrate the distributed deployment, we present IMPALA as an example. IMPALA is a highly scalable distributed DRL technique that efficiently handles large-scale learning by employing multiple actors running in parallel with a centralized learner, using importance sampling and V-trace off-policy correction to maintain stability in the learning process. In our implementation of IMPALA, multiple DRL Workers operate in parallel, each with its Environment Engine processing environmental data received from the ReinFog Mas-ter. The Worker DRL Core maintains local copies of the actor and critic networks for policy representation. Based on these local networks, the Worker DRL Core generates actions for IoT task scheduling. The scheduling results and rewards are processed locally by the Environ-ment Engine, and each DRL Worker's Trajectory Aggregator collects these experiences. The collected experiences are then transmitted to the DRL Learner through worker sessions managed by the Worker Session Manager, and stored in the Replay Buffer. The Learner DRL Core maintains the global actor and critic networks for policy optimization. It samples experiences from the Replay Buffer, computes importance sampling ratios to address the discrepancy between behavior and tar-get policies, and employs V-trace for off-policy correction to update these global networks. The updated global policy parameters are then distributed to all DRL Workers through the Synchronizer, ensuring consistency across the distributed system.

4.4. Native and library-based integration

ReinFog supports two primary mechanisms for integrating DRL techniques: native integration and library-based integration. The native integration mechanism enables direct implementation of DRL tech-niques within the framework, offering optimal performance and cus-tomization capabilities. The library-based integration mechanism al-lows seamless incorporation of external DRL libraries, providing access to established implementations while maintaining framework com-patibility. This dual integration approach ensures that ReinFog can accommodate diverse

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 5, Issue 5, November 2025 Impact Factor: 7.67

research needs while leveraging existing DRL solutions. In this section, we detail both integration mechanisms and provide examples of their implementation within ReinFog.

4.4.1. Native DRL techniques integration

The native integration mechanism is essential to ReinFog as it offers substantial advantages in performance, maintainability, and flexibil-ity. Directly implementing DRL techniques within the framework can optimize performance, particularly in dynamic edge/fog and cloud en-vironments where real-time responsiveness is crucial. This mechanism also minimizes dependencies on external libraries, enhancing stability and compatibility across various deployment scenarios. Furthermore, native integration provides researchers with the flexibility to imple-ment and integrate their own custom DRL techniques, allowing for tailored solutions to specific research problems.

Native integration mechanism: To support native integration of DRL techniques, ReinFog provides a comprehensive framework through well-defined interfaces and modular design, to offer maximum flex-ibility while maintaining consistency and efficiency. The Network Architecture Hub is responsible for neural network architecture def-inition and management. It maintains a collection of base network architectures (e.g., DNNs, RNNs, LSTMs, Transformers) and allows researchers to define custom architectures through a consistent inter-face. Building upon these network architectures, the DRL Technique Repository is responsible for managing DRL techniques, providing an extensible interface through a base class. This base class defines es-sential abstract methods including policy network initialization, action selection, loss computation, and parameter updates. These abstrac-tions enable researchers to implement new DRL techniques. ReinFog's Exploration Engine supports native integration through a pluggable exploration strategy interface, offering built-in configurability for com-mon strategies (e.g., ϵ-greedy and Ornstein–Uhlenbeck process noise) while enabling custom strategy implementation. The Replay Buffer enables custom buffer implementation through a standard interface. It provides several built-in buffer types (e.g., random sampling and reser-voir sampling) while allowing researchers to define flexible sampling strategies and integrate specialized buffer types.

Sample illustration: To demonstrate the native integration mechanism, we describe how to implement a customized version of IMPALA that leverages ReinFog's advanced features. The Network Architecture Hub is utilized to construct two Transformer-based neural networks. The ac-tor network processes the input states and outputs action probabilities for IoT task scheduling decisions, while the critic network evaluates these states to guide the learning process. IMPALA's core algorithm is implemented by extending the base class in the DRL Technique Repository, where the V-trace off-policy correction and importance sampling calculations are defined. The Exploration Engine is configured to use Ornstein–Uhlenbeck noise for action exploration. The Replay Buffer is set up with reservoir sampling to efficiently store and man-age experiences. Through these configurations and implementations, IMPALA operates as a fully functional distributed DRL technique within the ReinFog framework.

4.4.2. Library-based DRL techniques integration

The library-based integration mechanism serves as another cru-cial feature of ReinFog, offering development efficiency, algorithmic diversity, and research flexibility. By importing well-established DRL libraries, researchers can leverage proven algorithms without reimple-mentation, significantly reducing development time while ensuring re-liable performance. This mechanism also enriches ReinFog with diverse DRL techniques, enabling it to handle various scheduling scenarios in heterogeneous edge/fog and cloud environments. Moreover, it empow-ers researchers to integrate and experiment with different DRL libraries based on their specific research requirements.

Library-based integration mechanism: To support external DRL libraries integration, ReinFog provides a standardized interface through the DRL Technique Repository. This interface defines essential methods for bridging external libraries with ReinFog's environment. Specifically, the interface allows researchers to implement state preprocessing to convert ReinFog's system state representations into formats compatible with external libraries, action translation to map library-generated actions back to ReinFog's scheduling decisions, and reward signal adaptation to ensure proper learning feedback. The DRL Technique Repository manages these transformations, enabling external DRL tech-niques to operate seamlessly within ReinFog's resource management framework while maintaining their original implementations.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 5, Issue 5, November 2025 Impact Factor: 7.67

Sample illustration: To demonstrate the library-based integration mech-anism, we provide an example of integrating Recurrent Replay Dis-tributed DQN (R2D2) (Kapturowski et al., 2018) from the Ray1 library. Ray library is chosen for its high-performance distributed computing features and comprehensive DRL techniques suite. R2D2 is a distributed DRL technique that extends DQN by incorporating RNNs and a replay system to handle partial observability and temporal dependencies in sequential decision-making problems. Through the interface provided in the DRL Technique Repository, we implement state preprocessing to convert ReinFog's scheduling environment states (e.g., node resources, IoT task characteristics, and network conditions) into R2D2's required tensor format.

The action translation mechanism transforms R2D2's output probabilities into concrete scheduling decisions within ReinFog, effectively guiding the scheduling of IoT tasks across available nodes. For reward signal adaptation, the implementation processes ReinFog's performance metrics (e.g., scheduling result, response time, energy consumption) into a scalar reward value suitable for R2D2's learning process. These implementations enable R2D2 to effectively learn and make scheduling decisions within ReinFog while preserving its original recurrent replay-based learning mechanism.

V. MADCP: A MEMETIC ALGORITHM FOR DRL COMPONENT PLACEMENT

In ReinFog, effective DRL-based IoT application scheduling requires careful placement of various components (e.g., DRL Learners and DRL Workers) across different nodes in the heterogeneous computing en-vironment. Poor DRL component placement decisions can lead to in-creased communication overhead, inefficient resource utilization, and degraded learning performance. To solve this challenge, ReinFog offers a mechanism for DRL component placement. While traditional meta-heuristic algorithms could be applied to this placement problem, they show specific limitations. GA provides robust exploration capabilities but may converge slowly in complex solution spaces (Renders and Flasse, 1996). FA excels at local search refinement but can be trapped in local optima (Wu et al., 2020). PSO offers efficient global search but may lack fine-tuning abilities in local regions (Moradi and Gholampour, 2016). To address this critical placement challenge while overcoming these algorithmic limitations, we propose MADCP, a Memetic Algo-rithm for DRL Component Placement that combines the strengths of GA, FA, and PSO. In ReinFog, MADCP is integrated into the DRL Component Placement Engine of the extended Scheduler module (see Fig. 5). It is invoked before DRL training to determine efficient placement of DRL Learners and Workers across nodes.

In this section, we first define the optimization problem we are addressing along with the MADCP. We also explain how our proposed MADCP combines the strengths of three methods: GA, FA, and PSO. In addition, we present a comprehensive analysis of the computa-tional complexity of MADCP, examining its initialization and iterative optimization phases

5.1. Optimization problem definition

The MADCP is designed to address a complex optimization problem in distributed computing environments. The primary goal is to effi-ciently place DRL components across heterogeneous computing nodes to optimize overall system performance.

5.1.1. Problem formulation

Consider a set of DRL components $=\{C1,C2,\cdots,Cm\}$ that need to be assigned to a set of heterogeneous computing nodes $=\{N1,N2,\cdots,Nn\}$. Each node has different computational capabilities, memory sizes, and energy consumption rates.

Each component Ci has a computational requirement Ui (e.g., CPU cycles), a memory requirement Mi, and a deadline or time constraint Di. Similarly, each node Nj is characterized by a computational capac-ityPj, and available memory Aj.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 5, Issue 5, November 2025 Impact Factor: 7.67

5.1.2. Objective function

The objective is to find an optimal assignment of DRL components to computing nodes that minimizes the total operation time and en-ergy consumption while meeting all time and resource constraints. We define the objective function as:Minimize$F=m\Sigma i=1n\Sigma j=1xij(\omega 1\cdot O(Ci,Nj)+\omega 2\cdot E(Ci,Nj))$,(1)

Here, xij is a binary variable indicating whether component Ci is assigned to node Nj: $xij=\{1$,if component Ci is assigned to node Nj,0,otherwise.(2)$O(Ci,Nj)$ is the operation time of component Ci on node Nj, $E(Ci,Nj)$is the energy consumed by node Nj during the operation time $O(Ci,Nj)$, and $\omega 1$ and $\omega 2$ are weighting factors balancing the importance of operation time and energy consumption.

5.1.3. Constraints

The optimization problem is subject to the following constraints:

Resource constraints: For each node Nj, the total computational and memory requirements of the assigned components should not exceed its capacity: $m\Sigma i=1xijUi\leqslant Pj,\forall Nj\in$,(3)$m\Sigma i=1xijMi\leqslant Aj,\forall Nj\in$.(4)

Deadline constraints: Each component must complete the operation within its deadline:$O(Ci,Nj)\leqslant Di,\forall Ci\in,\forall Nj\in$ where $xij=1$.(5)

Assignment constraints: Each component is assigned to exactly one node:$xij\in\{0,1\},\forall Ci\in,\forall Nj\in$,(6)$n\Sigma j=1xij=1,\forall Ci\in$.

VI. CONCLUSIONS AND FUTURE WORK

This paper proposed ReinFog, a novel framework leveraging DRL mechanisms and techniques for adaptive resource management in edge/fog and cloud computing environments. ReinFog addresses the challenge of efficiently scheduling heterogeneous IoT applications across diverse computing resources through its modular and extensi-ble DRL components. It offers capabilities to support centralized and distributed DRL techniques and allows integration of both native and library-based DRL techniques. It features customizable deployment configurations, allowing users to flexibly configure DRL Learners and Workers based on system requirements. Additionally, it incorporates MADCP, an efficient DRL component placement algorithm that dy-namically optimizes the allocation of DRL Learners and Workers, enhancing DRL-based scheduling techniques performance in distributed environments. Our extensive experiments demonstrate that ReinFog is a lightweight and scalable framework capable of effectively scheduling IoT applications under diverse optimization objectives.

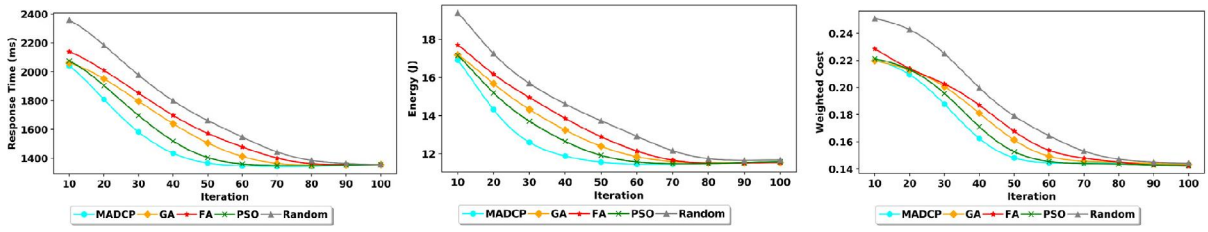


Fig. 15. Impact of different DRL component placement algorithms on IMPALA's convergence performance.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 5, Issue 5, November 2025
Impact Factor: 7.67

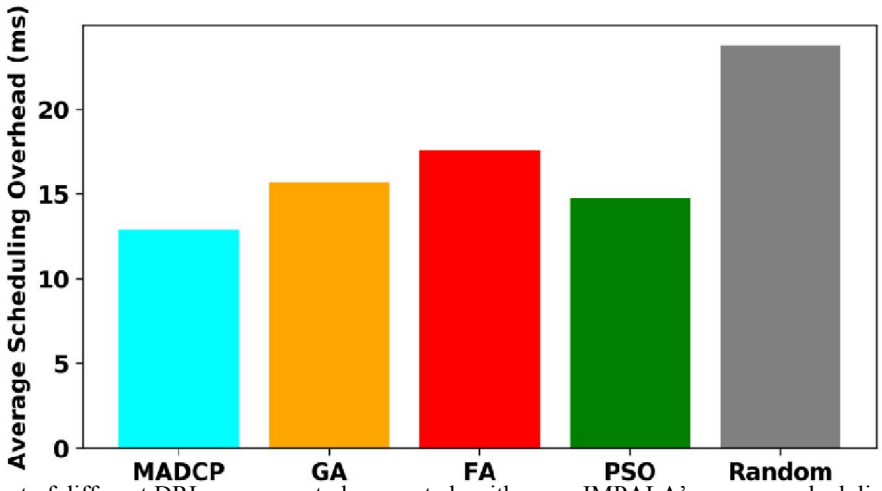


Fig. 16. Impact of different DRL component placement algorithms on IMPALA's average scheduling overhead.

As part of our future work, we will extend ReinFog by integrating security and privacy mechanisms to enable secure DRL parameter-sharing and experience-sharing approaches. The framework's modular design allows seamless integration of security protocols at all levels, ensuring that security features can be embedded for secure communication and data handling without affecting core functionality or system efficiency. Additionally, we will integrate more recent DRL techniques into ReinFog, which will allow the framework to remain at the fore-front of cutting-edge research. This integration will enable researchers to leverage advanced algorithms, providing a flexible platform for experimenting with and refining novel DRL techniques for diverse research. Furthermore, we plan to investigate techniques to improve system resilience against hardware and software failures, including fault-tolerant scheduling and adaptive component redistribution across heterogeneous nodes.

REFERENCES

[1]. Ali, E.S., Saeed, R.A., Eltahir, I.K., Khalifa, O.O., 2023. A systematic review on energy efficiency in the internet of underwater things (IoUT): Recent approaches and research gaps. J. Netw. Comput. Appl. 213, 103594.

[2]. Ali, I.M., Sallam, K.M., Moustafa, N., Chakraborty, R., Ryan, M., Choo, K.-K.R., 2020. An automated task scheduling model using non-dominated sorting genetic algorithm II for fog-cloud systems. IEEE Trans. Cloud Comput. 10 (4), 2294–2308.

[3]. Arulkumaran, K., Deisenroth, M.P., Brundage, M., Bharath, A.A., 2017. Deep reinforcement learning: A brief survey. IEEE Signal Process. Mag. 34 (6), 26–38.

[4]. Van den Bergh, F., Engelbrecht, A.P., 2006. A study of particle swarm optimization particle trajectories. Inform. Sci. 176 (8), 937–971.

[5]. Buyya, R., Srirama, S.N., Mahmud, R., Goudarzi, M., Ismail, L., Kostakos, V., 2023. Quality of service (QoS)-driven edge computing and smart hospitals: a vision, architectural elements, and future directions. In: International Conference on Communication, Electronics and Digital Technology. Springer, pp. 1–23.

[6]. Chen, W., Qiu, X., Cai, T., Dai, H.-N., Zheng, Z., Zhang, Y., 2021. Deep reinforcement learning for internet of things: A comprehensive survey. IEEE Commun. Surv. Tutor. 23 (3), 1659–1692.

[7]. Deb, K., Jain, H., 2013. An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part I: solving problems with box constraints. IEEE Trans. Evol. Comput. 18 (4), 577–601.

[8]. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T., 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans. Evol. Comput. 6 (2), 182–197.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 5, Issue 5, November 2025
Impact Factor: 7.67

[9]. Deng, Q., Goudarzi, M., Buyya, R., 2021. Fogbus2: a lightweight and distributed container-based framework for integration of iot-enabled systems with edge and cloud computing. In: Proceedings of the International Workshop on Big Data in Emergent Distributed Environments. pp. 1–8.

[10]. Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., Doron, Y., Firoiu, V., Harley, T., Dunning, I., et al., 2018. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In: Proceedings of the International Conference on Machine Learning. PMLR, pp. 1407–1416.

[11]. Ghosh, S., Mukherjee, A., Ghosh, S.K., Buyya, R., 2019. Mobi-iost: mobility-aware cloud-fog-edge-iot collaborative framework for time-critical applications. IEEE Trans. Netw. Sci. Eng. 7 (4), 2271–2285.