Reinforcement Learning Based Resource Allocation in Fog Computing

by

Masoud Mokhtari

A Dissertation Submitted in Partial Fulfillment of the
Requirements for the Degree of

DOCTOR OF PHILOSOPHY

in the Department of Computer Science

We acknowledge and respect the Lək ʷəŋən (Songhees and Xʷsepsəm/Esquimalt) Peoples on whose territory the university stands, and the Lək ʷəŋən and W̱SÁNEĆ Peoples whose historical relationships with the land continue to this day.

Reinforcement Learning Based Resource Allocation in Fog Computing

by

Masoud Mokhtari

Supervisory Committee

Dr. Sudhakar Ganti, Supervisor
(Department of Computer Science)

Dr. Alex Thomo, Internal Member
(Department of Computer Science)

Dr. Lin Cai, External Member
(Department of Electrical and Computer Engineering)

ABSTRACT

The Internet of Things (IoT) has revolutionized connectivity by enabling seamless data exchange among diverse devices, fostering intelligent services and informed decision-making. However, the rapid surge in data traffic has exposed the limitations of traditional cloud-based solutions, particularly in meeting the quality-of-service (QoS) demands of latency-sensitive applications. Fog computing has emerged as a transformative paradigm, extending computational resources closer to end-users and bridging the gap between centralized cloud systems and edge devices. This approach addresses QoS challenges by providing critical services and resources at the network's edge.

Despite its advantages, fog computing faces resource limitations at the node level, necessitating efficient resource allocation to optimize performance and meet application-specific QoS requirements. Deciding whether to process data at the fog or cloud level involves navigating complex trade-offs dictated by resource availability, offloading criteria, and diverse application scenarios.

This thesis addresses these challenges through a comprehensive approach to resource allocation in fog and cloud computing environments. First, a reinforcement learning-based method is introduced to optimize resource allocation for a single fog node. By formulating the problem as a Markov Decision Process (MDP), the approach maximizes fog resource utilization while considering the number of resource blocks and delay tolerance for each request. Experimental evaluations demonstrate the superiority of the E-SARSA algorithm in terms of speed, utilization, and adaptability compared to Q-learning, SARSA, and a Fixed-Threshold approach.

The study then extends to multi-fog/cloud systems, introducing a two-phase process. In the first phase, the optimal fog node for resource allocation is identified. In the second phase, reinforcement learning is applied to determine whether tasks should be processed locally or offloaded to the cloud. This method ensures efficient resource utilization, with experimental results highlighting the superior performance of the Selection-2 approach compared to Genetic Algorithms (GA), Round Robin (RR), and Random strategies, particularly in speed, utilization, and load balancing.

Finally, the framework is further enhanced with a hybrid approach combining Genetic Algorithms and Reinforcement Learning (GA/RL) for dynamic resource allocation in integer-based multi-fog/cloud systems. This method applies the two-phase process, achieving significant improvements in speed, utilization, and load balancing compared to existing methods.

By dynamically allocating fog resources and optimizing offloading strategies, this work addresses the limitations of traditional cloud computing systems and ensures seamless performance for latency-sensitive IoT applications. The proposed approaches advance resource allocation strategies in fog and cloud computing, offering scalable, efficient, and adaptive solutions for future IoT ecosystems.

Contents

List of Tables

List of Figures

ACKNOWLEDGEMENTS

This thesis is the culmination of a journey that would not have been possible without the incredible support, guidance, and encouragement of several remarkable individuals. First and foremost, I extend my deepest gratitude to my supervisor, whose unconditional belief in my potential and tireless dedication to my success have been the cornerstone of this achievement. Your expertise, patience, and encouragement have guided me through the most challenging moments, and I truly could not have reached this point without your mentorship. Thank you for being not just an academic guide but a source of inspiration and strength. To my parents, whose love and sacrifices have shaped the person I am today—thank you for always believing in me, even when I doubted myself. Your steadfast support, both emotional and practical, has been my foundation throughout this journey. To my sister, my constant inspiration and confidant, your faith in my dreams has kept me going, even during moments of uncertainty. This work stands as a testament to the collective strength and belief of all of you. Thank you for walking this path with me.

DEDICATION

To my family.

Chapter 1

Introduction

The rapid expansion of the Internet of Things (IoT) and the increasing demand for real-time data processing have created significant challenges in the management of computational resources. Traditional cloud computing, while offering robust computational power, often fails to meet the stringent requirements of latency-sensitive and resource-constrained applications. To overcome these limitations, fog computing has emerged as a promising paradigm. Fog computing extends cloud capabilities to the edge of the network, bringing computational resources closer to end-users and devices. This decentralized approach significantly reduces latency, optimizes bandwidth usage, and enhances the performance of applications across various domains, including smart cities, healthcare, autonomous vehicles, industrial automation, and next-generation (B5G/6G) edge intelligence.

In fog/edge computing, *task scheduling* and *resource management* are critical components that determine how computational tasks are allocated across edge devices, fog nodes, and cloud servers. This allocation must account for multiple objectives, including minimizing task execution time and end-to-end latency, reducing energy consumption, ensuring high-quality service (QoS) and SLA satisfaction, maintaining reliability, and controlling operational costs. The complexity of scheduling in fog environments arises from several intertwined factors: (i) heterogeneity across compute, storage, and network resources; (ii) dynamics of workload arrivals and mobility; (iii) partial observability and uncertainty in network conditions; (iv) strict real-time constraints and deadline sensitivity; and increasingly, (v) *security* and *privacy* requirements for distributed intelligence and data-driven decision-making. As a result, developing efficient, scalable, and trustworthy scheduling algorithms remains a central research challenge.

1.1 The Strategic Importance of RL-Driven Orchestration

The problem of resource allocation in fog computing is fundamentally an *NP-hard* optimization problem complicated by highly stochastic environments. As IoT networks scale to billions of devices, static policies and traditional heuristics are increasingly incapable of managing the explosive volatility of workload demands. In this context, the application of Reinforcement Learning (RL) has transitioned from a theoretical novelty to a strategic imperative.

RL-driven resource allocation is pivotal for three primary reasons. First, it enables autonomous adaptability; unlike ILP or static heuristics which require complete prior knowledge of the system model, RL agents learn optimal policies through interaction, allowing the system to adapt to unknown traffic patterns and node failures in real-time. Second, it addresses the multi-objective dilemma inherent in modern infrastructure—balancing the conflicting goals of minimizing latency for critical tasks while maximizing energy efficiency to prolong battery life in edge devices. Third, in the context of safety-critical applications such as autonomous driving and telesurgery, the capacity of RL to make millisecond-level decisions based on learned long-term rewards is not merely an efficiency metric but a safety requirement. Consequently, the development of robust RL and Hybrid-RL frameworks is widely regarded as the cornerstone for enabling the autonomy required in 6G and beyond.

1.2 Task Scheduling Algorithms in Fog Computing

To address these challenges, the literature proposes a wide spectrum of methods that can be broadly classified into: **(i) mathematical optimisation (e.g., ILP/MILP)**, **(ii) heuristic and meta-heuristic algorithms**, **(iii) hybrid optimisation frameworks**, **(iv) reinforcement learning (RL) and deep reinforcement learning (DRL)**, **(v) fuzzy/knowledge-driven reasoning**, and more recently, **(vi) federated (and privacy-preserving) learning for distributed control** and **(vii) security-aware offloading and trustworthy scheduling**. Each family offers a different trade-off among optimality, scalability, adaptability, and implementation complexity.

1.2.1 Integer Linear Programming (ILP) Algorithms

ILP and related exact optimisation methods are known for providing optimal solutions to task scheduling problems, particularly in small-scale or static scenarios. These methods

formulate scheduling as a constrained optimisation problem, typically aiming to minimise latency and/or energy while satisfying capacity and deadline constraints. However, as the scale increases and the environment becomes dynamic, the computational complexity grows rapidly, limiting direct applicability in large-scale fog systems. Representative studies (e.g., Tsai et al. [18] and Fonseca et al. [17]) demonstrate the use of ILP-style formulations to balance resource allocation across fog and cloud infrastructures, often serving as (i) optimality benchmarks, (ii) guides for heuristic design, or (iii) components in hybrid pipelines.

1.2.2 Heuristic and Meta-Heuristic-Based Solutions

Heuristic and meta-heuristic approaches provide approximate solutions with lower computational overhead than ILP, making them suitable for real-time and large-scale settings. Common methods include Genetic Algorithms (GAs), Particle Swarm Optimisation (PSO), Ant Colony Optimisation (ACO), and Cuckoo Search. For example, Canali et al. [31] and Aburukba et al. [30] employed GA-based optimisation to reduce latency in IoT-driven scheduling. While these approaches are more scalable than exact solvers, they typically cannot guarantee global optimality and may exhibit sensitivity to parameter tuning, stopping rules, and population diversity management.

While Reinforcement Learning provides adaptability, Genetic Algorithms (GA) have remained a popular choice for static and complex multi-objective optimization in Fog environments, as highlighted in a comprehensive survey by Guerrero et al. [114]. Recent works such as PGA [110] demonstrate GA's continued effectiveness in heterogeneous environments, while Saad et al. [111] utilized a hybrid GA-PSO approach to enhance multi-objective scheduling for big data applications. Addressing energy and cost constraints specifically, Sahoo et al. [112] proposed the EMCS algorithm, utilizing evolutionary strategies to optimize makespan and power consumption simultaneously. Other hybrid approaches have also emerged to address convergence issues; for instance, Kumar and Singh [113] proposed HGD-GBCO, which fuses Gradient Descent with Genetic Bee Colony Optimization to dynamically switch between exploration and exploitation, preventing the stagnation often seen in pure meta-heuristics. However, purely evolutionary methods often struggle with real-time adaptability to rapidly changing network conditions, a gap this thesis addresses by integrating GA with the dynamic learning capabilities of RL.

1.2.3 Hybrid Algorithms

Hybrid algorithms combine complementary optimisation paradigms to improve robustness and solution quality. Wang et al. [23] integrated PSO and ACO to jointly consider completion time, energy consumption, and reliability, while Hosseinioun et al. [33] proposed an IWO-CA hybrid to enable energy-aware scheduling. In modern fog systems, hybridisation often appears in two practical forms: (i) *two-phase scheduling* (a fast heuristic filtering/selection stage followed by a refined optimiser), and (ii) *learning–optimisation coupling*, where ML/RL guides the search space and meta-heuristics improve exploration or accelerate convergence.

Recent advancements have further refined this hybridization. For instance, [GA-PSO 2024] combines GA with Particle Swarm Optimization to improve convergence speed, while others have merged Gradient Descent with Genetic Bee Colony [GD-GBCO 2025] to show the efficacy of mixing local search with global search. More closely related to the objectives of this thesis is the specific integration of evolutionary techniques with reinforcement learning; notably, [Hybrid GA-RL 2025] proposes a resource allocation framework in multi-fog/cloud systems that leverages the strengths of both paradigms.

Furthermore, Choppara and Mangalampalli [99] developed a hybrid technique merging Fuzzy Logic with Deep Reinforcement Learning. By using Fuzzy Logic to handle task prioritization and input uncertainty before the learning phase, they demonstrated improved decision-making reliability. Similarly, Nucci and Papadia [106] explored the intersection of meta-heuristics and machine learning in IoT-enabled healthcare maintenance. Their framework couples a Genetic Algorithm (GA) with a Proximal Policy Optimization (PPO) agent, using the GA to "seed" the DRL agent with high-quality initial policies. This illustrates how evolutionary search can complement deep policies under complex constraints—a design pattern directly relevant to fog/cloud scheduling, where hybrid GA–RL strategies can exploit GA-based exploration while preserving RL's long-horizon adaptivity (as developed later in Chapter 4).

1.2.4 Reinforcement Learning (RL)-Based Algorithms

RL-based scheduling has gained significant attention due to its ability to learn adaptive policies in uncertain and time-varying environments. By interacting with the system and observing feedback, RL agents can discover scheduling policies that optimise long-term objectives such as delay, energy, congestion, and SLA satisfaction. Studies such as Guo et al. [8] and Peng et al. [25] demonstrate RL-driven delay reduction and security/robustness

improvements in vehicular edge computing (VEC) and IoT settings.

1.2.5 Deep Reinforcement Learning (DRL) Architectures

DRL extends RL by incorporating function approximation (typically deep neural networks) to handle high-dimensional states and complex policy classes. This is particularly effective in heterogeneous fog environments, where the state may include multi-dimensional resource vectors, network conditions, and task-level features. For example, Zheng et al. [5] addressed task migration via DRL by learning optimal time/location for service relocation.

Recent scholarship has focused on creating modular and specialized DRL frameworks to address specific architectural challenges. Wang et al. [108] proposed *ReinFog*, a modular framework designed to bridge the gap between theoretical DRL models and practical deployment. Their work emphasizes the use of Actor-Critic (A2C) agents to manage resources across the Edge-Fog-Cloud continuum. Focusing on workload granularity, Kallela et al. [102] introduced *DRL4HFC*, utilizing Deep Q-Networks (DQN) specifically for container-based microservice scheduling, highlighting the necessity of treating tasks as interconnected components of larger business processes.

Furthermore, Multi-agent DRL (MADRL) has become crucial for scalable, multi-fog systems. Ling et al. [100] advanced this field in the context of Industrial IoT (IIoT) by incorporating *Attention Mechanisms* into MADRL. Their approach filters state features to prioritize critical information, addressing the interference and noise inherent in industrial environments. This effectively allows multiple decision-makers to coordinate under local observations while focusing policy updates on the most relevant nodes and interactions.

1.2.6 Privacy-Preserving and Federated Learning Frameworks

As fog computing is increasingly deployed in sensitive environments—such as healthcare, autonomous driving, and private industrial networks—data privacy has become a primary constraint. This has led to the adoption of Federated Learning (FL), where edge nodes collaboratively train models without exchanging raw data.

Andong and Min [104] proposed a Federated Multi-Agent Reinforcement Learning (Fed-MARL) framework specifically for 6G edge networks. Their work addresses the "cross-layer" challenge, orchestrating resources at both the MAC layer (spectrum access) and the Application layer (task offloading) while mathematically guaranteeing data privacy through secure aggregation protocols. Similarly, Mali et al. [105] applied Federated Reinforcement

Learning to the problem of *dynamic resource allocation*, allowing agents to learn general traffic patterns from neighboring nodes without breaching privacy boundaries.

Na et al. [103] further refined this paradigm by proposing a *Hybrid Federated and Centralized Learning* architecture. They argued that a purely federated approach might suffer from slow convergence, while a purely centralized one risks privacy. Their hybrid model offloads less sensitive data for centralized training while keeping critical data local, striking an optimal balance between prediction accuracy and energy consumption in aerial (UAV) networks.

1.2.7 Emerging Paradigms: Digital Twins, Intent, and Security

Beyond standard allocation algorithms, several novel architectural paradigms have emerged to handle physical complexities and high-level management.

In high-mobility scenarios, such as Vehicular Edge Computing (VEC), Wang et al. [109] introduced a *Digital Twin-Assisted* approach. By maintaining a synchronized digital replica of the physical network, their system allows the scheduling algorithm to train on the "Twin" rather than the live network, enabling proactive offloading decisions that anticipate connectivity breaks.

Moving up the abstraction ladder, Konidaris et al. [107] proposed *Intent-Based Resource Allocation*. This paradigm shifts the focus from low-level technical metrics to high-level business goals ("Intents"), utilizing Reinforcement Learning to automatically translate a network administrator's intent into concrete allocation policies. Finally, addressing the often-overlooked aspect of trust, Roshan et al. [101] presented a *Secure Task-Offloading Framework*. Their work emphasizes that in cooperative fog environments, resource allocation must account for the reputation and trustworthiness of neighboring nodes, ensuring tasks are offloaded only to secure entities to mitigate data tampering risks.

1.3 Reinforcement Learning

Reinforcement Learning (RL) is a core subfield of machine learning focused on learning optimal behaviors through interaction with an environment. Unlike supervised learning, which relies on labeled data, RL involves learning from experience and interaction. Agents learn to make sequences of decisions by taking actions in an environment and receiving feedback in the form of scalar rewards [11, 12]. These rewards inform the agent whether its previous actions have brought it closer to or farther from achieving its objective. This form

of learning is deeply inspired by behavioral psychology, where organisms learn through rewards and punishments.

What makes RL particularly powerful is its ability to learn optimal behavior in dynamic and uncertain environments without explicit supervision [1, 6]. Applications of RL range from training robots to perform complex physical tasks, to optimizing treatments in healthcare, to powering game-playing systems that can beat human champions [8, 25, 26]. The rise of deep learning has enabled RL to scale to large and complex environments, giving birth to a subfield called Deep Reinforcement Learning (DRL) [4,5,22]. In this chapter, we delve into the mathematical foundations, algorithmic strategies, and practical considerations necessary to understand and implement RL systems effectively.

1.3.1 The Reinforcement Learning Framework

Agent-Environment Interaction

Reinforcement learning problems are modeled as a sequential interaction between an agent and its environment. The agent is the decision-maker, while the environment includes everything it interacts with. At each discrete time step t, the agent observes a state $s_t \in S$, selects an action $a_t \in A(s_t)$, and in return, the environment provides a scalar reward $r_{t+1} \in R$ and transitions to a new state $s_{t+1} \in S$ [38, 40]. The agent's goal is to learn a policy π that maps states to actions to maximize the cumulative future reward.

This setup abstracts a wide variety of problems into a common structure, making RL a general-purpose tool for learning and control. The repeated feedback loop of interaction is fundamental: the agent's actions influence the future states and rewards it encounters, creating a closed-loop system that distinguishes RL from other types of learning [21, 22].

Markov Decision Process (MDP)

The formal framework for RL is the Markov Decision Process (MDP). An MDP provides a mathematical description of an environment in terms of states, actions, transitions, rewards, and a discount factor. Specifically, an MDP is defined as a 5-tuple (S, A, P, R, γ), where S denotes the set of all possible states, A is the set of actions, $P(s'|s,a)$ is the transition probability, $R(s,a)$ is the expected immediate reward, and $\gamma \in [0,1]$ is the discount factor [14, 23].

The Markov property is a crucial aspect of MDPs. It asserts that the transition probabilities depend solely on the current state and action, not on any prior history. This property

enables a mathematically tractable formulation and recursive structure used in RL algorithms [25, 40].

1.3.2 Objective of Reinforcement Learning

The principal aim of reinforcement learning is to identify an optimal policy that maximizes the expected cumulative reward over time [8, 21]. A policy $\pi(a|s)$ defines the behavior of the agent by specifying the probability of taking action a in state s.

Return

The return at time step t, denoted G_t, is defined as the discounted cumulative reward from that point forward:

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}, \tag{1.1}$$

where $\gamma \in (0,1]$ is the discount factor and k indexes the look-ahead horizon. Here $k = 0$ refers to the immediate reward r_{t+1}, $k = 1$ to the reward one step further r_{t+2}, and so on, with each subsequent reward attenuated by the factor γ^k.

This formulation provides the mathematical foundation for long-term sequential decision-making [38, 42].

Value Functions

To evaluate states and actions under a given policy, RL introduces value functions. The state-value function $V^{\pi}(s)$ quantifies the expected return starting from state s. The action-value function $Q^{\pi}(s,a)$ measures the expected return from state s after taking action a [5,14]. These functions are central because they allow comparison of alternatives and improvement of policies through iterative updates.

1.3.3 Dynamic Programming

Dynamic Programming (DP) methods solve RL problems when the complete model of the environment is available. That is, the agent knows transition dynamics and rewards [10, 11].

Table 1.1: Core notation for Markov decision processes and value functions

Symbol	Meaning	Description
$s \in S$	State	Configuration of the environment at a discrete time step.
$a \in A(s)$	Action	Control decision available to the agent when it is in state s.
$\pi(a \mid s)$	Policy	Probability that the agent selects action a in state s; fully characterises the agent's behaviour.
$P(s' \mid s,a)$	Transition probability	Probability that the environment transitions to successor state s' after the agent executes action a in state s.
$R(s,a)$	Expected reward	Average immediate reward obtained when action a is taken in state s. A richer model $R(s,a,s')$ can include dependence on the realised successor state.
$\gamma \in [0,1]$	Discount factor	Exponential weight applied to future rewards; $\gamma = 0$ yields myopic behaviour, while $\gamma \to 1$ values long-term returns almost equally to immediate rewards.
$V^{\pi}(s)$	State-value function	Expected discounted return when starting in state s and following policy π: $V^{\pi}(s) = \mathbb{E}_{\pi}\Big[\sum_{t=0}^{\infty} \gamma^t r_{t+1} \;\Big|\; s_0 = s\Big].$
$V^{*}(s)$	Optimal value	Maximum achievable expected return from state s over all admissible policies: $V^{*}(s) = \max_{\pi} V^{\pi}(s).$

Bellman Equations

Bellman expectation equation For a fixed policy π, the value function satisfies,

$$V^{\pi}(s) \;=\; \sum_{a \in A(s)} \pi(a \mid s) \sum_{s' \in S} P(s' \mid s,a)\big[R(s,a) + \gamma V^{\pi}(s')\big], \tag{1.2}$$

The outer sum averages over the agent's stochastic choice of action dictated by π; the inner sum averages over stochastic environment transitions. The bracketed term combines the one-step expected reward with the discounted value of the successor state.

Bellman optimality equation The optimal value function obeys,

$$V^*(s) \;=\; \max_{a\in A(s)} \sum_{s'\in S} P(s' \mid s,a)\,\big[R(s,a)+\gamma V^*(s')\big], \tag{1.3}$$

Here the maximisation over a replaces the policy average, reflecting that an optimal agent always selects the action that delivers the highest expected long-term return. This nonlinear fixed-point equation underpins dynamic-programming solution methods such as value iteration and policy iteration.

These recursive definitions form the theoretical foundation of RL [12, 24].

Table 1.2: Symbols introduced in the Bellman-equation subsection

Symbol	Meaning	Description
S	State space	Set of all possible environment states.
$s \in S$	State	Specific configuration of the environment at a given time step.
$A(s)$	Action set	Feasible actions when the agent is in state s.
$a \in A(s)$	Action	Control decision executed in state s.
$\pi(a \mid s)$	Policy	Probability of selecting action a in state s; defines the agent's behaviour.
$P(s' \mid s,a)$	Transition model	Probability that the environment moves to successor state s' after action a is taken in s.
$R(s,a)$	Reward model	Expected immediate reward obtained by executing a in s (can be extended to $R(s,a,s')$ if it depends on the outcome state).
$\gamma \in [0,1]$	Discount factor	Exponential weight applied to future rewards; $\gamma = 0$ makes the agent myopic, $\gamma \to 1$ emphasises long-term returns.
$V^\pi(s)$	State-value function	Expected discounted return starting from s and following policy π: $V^\pi(s) = \mathbb{E}_\pi\Big[\sum_{t=0}^{\infty} \gamma^t r_{t+1} \;\Big|\; s_0 = s\Big]$.
$V^*(s)$	Optimal value	Maximum attainable expected return from s over all policies: $V^*(s) = \max_\pi V^\pi(s)$.

Policy Iteration and Value Iteration

Policy iteration alternates between policy evaluation and improvement, while value iteration merges them using the Bellman optimality equation [21, 22]. These algorithms guarantee convergence to optimal policies for finite MDPs.

1.3.4 Model-Free Methods

In most real-world RL problems, transition dynamics and rewards are unknown. Model-free methods learn directly from interaction [4, 5]. Two families exist:

- **Monte Carlo methods**: rely on complete episodes to estimate returns.
- **Temporal-Difference (TD) methods**: update estimates from partial experience.

1.3.5 Control Algorithms

Control algorithms combine value estimation with policy improvement to learn optimal behavior [14, 15].

Q-Learning

Q-learning is an off-policy TD method. Update rule:

$$Q_{t+1}(s_t, a_t) \;=\; Q_t(s_t, a_t) \;+\; \alpha\Big[r_{t+1} \;+\; \gamma \max_{a' \in A(s_{t+1})} Q_t(s_{t+1}, a') \;-\; Q_t(s_t, a_t)\Big], \tag{1.4}$$

Equation (1.4) is the core *Q-learning* update: it shifts the current estimate $Q_t(s_t, a_t)$ toward the sum of the immediate reward and the discounted best-case value of the successor state, with step size α. It converges to the optimal value function under sufficient exploration [14, 31].

SARSA

SARSA is an on-policy TD algorithm:

$$Q_{t+1}(s_t, a_t) \;=\; Q_t(s_t, a_t) \;+\; \alpha\Big[r_{t+1} + \gamma Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t)\Big], \tag{1.5}$$

Table 1.3: Notation used in the Q-learning update

Symbol	Meaning	Description
s_t	Current state	Environment state observed at discrete time step t.
a_t	Current action	Action selected in s_t (e.g., by an ε-greedy exploration policy).
r_{t+1}	Immediate reward	Scalar reinforcement signal obtained right after executing a_t.
s_{t+1}	Next state	Successor state produced by the environment's transition dynamics after a_t.
$A(s)$	Action set	All admissible actions when the agent is in state s. In the TD target, $\max_{a' \in A(s_{t+1})} Q_t(s_{t+1}, a')$ is the *greedy target*: the highest estimated action value available in s_{t+1}.
$Q_t(s,a)$	Action–value estimate	Learner's prediction (at time t) of expected return for taking action a in state s and following the current policy thereafter.
$\alpha \in (0,1]$	Learning rate	Step-size parameter that controls how strongly the new TD target $r_{t+1} + \gamma \max_{a'} Q_t(s_{t+1}, a')$ overrides the existing estimate $Q_t(s_t, a_t)$. Small α smoother learning; large α faster but higher-variance updates.
$\gamma \in [0,1]$	Discount factor	Exponential weight applied to future rewards. $\gamma \approx 0$ makes the agent myopic; $\gamma \to 1$ values long-term returns almost as highly as immediate rewards.

It evaluates and improves the same policy used for behavior, yielding more stable learning in stochastic environments [2, 7].

Expected SARSA

Expected SARSA reduces variance by computing the expectation over all possible next actions:

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha \Big[r_{t+1} + \gamma \sum_{a' \in A(s_{t+1})} \pi(a' \mid s_{t+1})\, Q_t(s_{t+1}, a') - Q_t(s_t, a_t) \Big], \quad (1.6)$$

This method improves stability and smoothness compared to SARSA [8, 25].

1.3.6 Exploration vs. Exploitation

Balancing exploration and exploitation is fundamental [12, 34]. Strategies include:

Table 1.4: Notation used in the on-policy SARSA update

Symbol	Meaning	Description
s_t	Current state	Environment state observed at discrete time step t.
a_t	Current action	Action selected in state s_t according to the behaviour policy (e.g., ε-greedy).
r_{t+1}	Immediate reward	Scalar reward produced by the environment immediately after executing a_t.
s_{t+1}	Next state	Successor state generated by the environment's transition dynamics.
a_{t+1}	Next action	Action chosen by the *same* policy in state s_{t+1}; this makes SARSA an on-policy method.
$Q_t(s,a)$	Action–value estimate	Agent's estimate, at time t, of the expected discounted return for taking action a in state s and thereafter following its policy.
$Q_{t+1}(s_t,a_t)$	Updated estimate	New value assigned to the pair (s_t,a_t) after observing $(s_t,a_t,r_{t+1},s_{t+1},a_{t+1})$.
$\alpha \in (0,1]$	Learning rate	Step-size parameter controlling how strongly the new TD target overwrites the old estimate. Smaller α gives smoother learning; larger α adapts faster but with higher variance.
$\gamma \in [0,1]$	Discount factor	Exponential weight applied to future rewards; $\gamma \approx 0$ makes the agent myopic, whereas $\gamma \to 1$ values long-term returns nearly as much as immediate ones.

- ε-greedy exploration
- Softmax (Boltzmann) exploration
- Upper Confidence Bound (UCB) methods

Exploration strategies are critical in high-dimensional or safety-sensitive domains [40, 41].

1.3.7 Function Approximation

In large or continuous state spaces, function approximation is essential [24, 29]. Instead of tabular values, parameterized functions (linear models or neural networks) generalize across states.

Deep Q-Networks (DQN) are a prime example, stabilizing learning using experience replay and target networks [4, 5]. Function approximation is key to DRL scalability but

Table 1.5: Notation used in the Expected-SARSA update

Symbol	Meaning	Description
s_t	Current state	Environment state observed at discrete time step t.
a_t	Current action	Action selected in state s_t by the agent's behaviour policy π.
r_{t+1}	Immediate reward	Scalar reward received immediately after executing a_t.
s_{t+1}	Next state	Successor state produced by the environment's transition dynamics.
$A(s)$	Action set	Set of admissible actions when the agent is in state s.
$\pi(a' \mid s_{t+1})$	Policy probability	Probability of selecting action a' in the successor state s_{t+1}; taking the expectation over a' leads to the *Expected-SARSA* target.
$Q_t(s,a)$	Action–value estimate	Agent's estimate (at time t) of the expected return for taking action a in state s and thereafter following π.
$\alpha \in (0,1]$	Learning rate	Step-size parameter that controls how strongly the new TD target overwrites the previous estimate.
$\gamma \in [0,1]$	Discount factor	Exponential weight applied to future rewards; lower values emphasise immediacy, higher values emphasise long-term returns.

introduces stability and bias challenges [22, 43].

1.3.8 Policy Gradient Methods

Policy gradient methods directly optimize the policy [42, 44]. The REINFORCE algorithm uses:

$$\nabla_\theta J(\theta) \;=\; \mathbb{E}_{(s,a)\sim\pi_\theta}\big[\nabla_\theta \log \pi_\theta(a \mid s)\, G_t\big], \tag{1.7}$$

Actor-Critic frameworks combine policy gradients with value estimation for variance reduction. Advanced algorithms include PPO and TRPO, which improve stability in large-scale tasks [45, 46].

Applications and Extensions

RL is widely applied in robotics, healthcare, finance, and networking [7, 8, 25]. Beyond single-agent RL:

Table 1.6: Notation used in the policy-gradient formulation

Symbol	Meaning	Description
θ	Policy parameters	Vector of trainable weights for the stochastic policy network or other function approximator.
$J(\theta)$	Objective function	Expected return under policy π_θ: $J(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{\infty} \gamma^t r_{t+1}\right]$.
$\pi_\theta(a \mid s)$	Stochastic policy	Probability of selecting action a in state s when the policy is parameterised by θ.
$\nabla_\theta \log \pi_\theta(a \mid s)$	Score function	Gradient of the log-likelihood of the chosen action (likelihood-ratio term) used in REINFORCE-style updates.
G_t	Return	Discounted cumulative reward from time step t: $G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$.
$\mathbb{E}_{(s,a)\sim\pi_\theta}$	Policy expectation	Expectation with respect to the state–action distribution generated by running policy π_θ.
$\nabla_\theta J(\theta)$	Policy-gradient	Gradient of the expected return with respect to the parameters; direction for gradient-ascent updates in policy-gradient methods.

- **Multi-agent RL (MARL)**: for coordination and competition [2].
- **Hierarchical RL**: temporal abstraction for long-term planning [93].
- **Inverse RL**: infers reward functions from expert demonstrations [94].
- **Offline RL**: learns from fixed datasets without live interaction [95].

1.4 Challenges and Future Directions

Key challenges include sample inefficiency, sparse rewards, instability, and poor generalization [38, 40]. Current research addresses:

- Model-based RL for better sample efficiency.
- Representation learning for generalization.
- Human-in-the-loop RL for safe exploration.
- Hierarchical/meta-RL for faster adaptation.

1.4.1 Summary of RL

Reinforcement Learning provides a comprehensive framework for sequential decision-making through interaction. From MDP foundations to modern DRL and policy optimization, RL has grown into a robust paradigm with real-world applications. While challenges remain in stability, efficiency, and generalization, ongoing advancements promise scalable and safe RL deployments across diverse domains [11, 12, 21, 22, 38, 40, 46].

1.5 Genetic Algorithm

Genetic Algorithms (GAs) constitute a celebrated family of population-based, stochastic search and optimisation procedures whose intellectual roots lie in the twin disciplines of population genetics and Charles Darwin's theory of natural selection [50–53]. Over the past five decades the GA paradigm has matured into a remarkably versatile optimisation framework—one that thrives whenever the decision landscape is high-dimensional, highly nonlinear, discontinuous, multimodal, or simply too poorly understood to surrender to classical calculus-driven techniques [54, 55]. Because a GA dispenses with derivatives, Hessians, and convexity assumptions, it can nimbly explore rugged or noisy spaces where gradients are unavailable or uninformative—and it can do so while imposing only minimal mathematical structure on the objective function [56].

The conceptual skeleton of a GA is unmistakably Darwinian. A population of *chromosomes*—abstract encodings of candidate solutions—is allowed to *evolve* over many generations according to the principles of selection, inheritance, recombination, and random mutation [57]. At every generation the algorithm evaluates the *fitness* of each individual, selects those of superior merit, and deploys biologically-inspired operators to breed a new cohort of offspring [58]. Iterated indefinitely, this cycle nudges the population, on average, toward regions of the search space that harbour ever-improving solutions.

From a methodological standpoint, GAs belong to the broader class of *metaheuristics*: pragmatic, problem-agnostic search strategies that trade mathematical guarantees for the ability to tackle formidable spaces with aplomb [55]. Their inherently parallel, many-points-in-flight search style not only lends itself to modern multi-core or GPU hardware [60] but also mitigates the danger of being seduced by the first decent local optimum that comes into view [59].

Since the seminal work of Holland in the mid-1970s [50], genetic algorithms have become a mainstay across an astonishing array of disciplines, including—though by no

means limited to—engineering optimisation [61], production scheduling and logistics [62], protein-structure prediction [63], autonomous-robot control [64], computational economics [65], and machine creativity under the banner of Genetic Programming [66]. The GA paradigm has itself evolved into myriad variants—Neuroevolution for evolving both the weights *and* the topologies of neural networks [67]; memetic or hybrid GA–local-search schemes that graft powerful neighbourhood heuristics onto the stochastic backbone [68]; and island models that run several sub-populations in parallel with controlled migration to preserve diversity.

The remainder of this chapter offers a thorough, self-contained tour of genetic algorithms (GAs). We retrace their biological inspiration and formalise the canonical workflow; examine representation/encoding choices, selection schemes, crossover and mutation operators, and replacement policies; discuss convergence intuition and runtime considerations; and dissect common pitfalls and limitations alongside practical diagnostics. We then highlight current research frontiers and design patterns relevant to large-scale, latency-sensitive systems. Throughout, the narrative emphasises both conceptual insight *and* implementation detail so that readers emerge with the confidence—and the cautionary wisdom—needed to harness GAs in their own optimisation endeavours. Appendix A presents a detailed, end-to-end example of a GA application.

1.5.1 Biological Inspiration

At its philosophical core, a GA is an abstraction of evolutionary biology [57]. In the natural world, information about an organism's traits is packaged into chromosomes—long DNA molecules segmented into discrete units called genes. Through the processes of sexual reproduction and mutation, chromosomes are copied, shuffled, and occasionally altered, producing offspring whose phenotypes are stochastic mosaics of parental traits [69, 70]. Those offspring that are better adapted to the environment are more likely to survive and reproduce, thereby biasing the genetic pool in favour of advantageous alleles.

A GA transposes this grand, slow drama onto the digital stage: chromosomes become bit-strings, real-valued vectors, trees, or permutations; *fitness* supplants ecological survivability; parental selection is rendered by probabilistic sampling biased toward higher fitness [51]; crossover mirrors meiotic recombination [71]; and mutation introduces the random sparks of novelty that keep the population from genetic monoculture [76]. Because the algorithm harvests many partially independent search trajectories simultaneously, it can sift through multiple basins of attraction in parallel—a feat that eludes most greedy local heuristics.

Table 1.7: Notation introduced in the Genetic-Algorithm framework section

Symbol	Meaning	Description
N	Population size	Number of chromosomes maintained at each generation.
p_c	Crossover probability	Chance that a selected parent pair undergoes recombination instead of being copied unchanged. Typical range $0.6-0.9$.
p_m	Mutation probability	Per-gene probability of applying the mutation operator; usually one order of magnitude lower than p_c.
f_i	Fitness of individual i	Objective value used to rank or select chromosomes. Higher (or lower, for minimisation) means better.
$P(i)$	Selection probability	In roulette-wheel selection, $P(i) = f_i \big/ \sum_{j=1}^{N} f_j$. Individuals with larger fitness occupy larger "wheel" segments.
k	Cut point	Index of the crossover breakpoint in single-point recombination for binary strings of length n.
$\alpha \in [0,1]$	Blend coefficient	Weighting factor in arithmetic (or BLX-α) crossover for real vectors; controls how far offspring move toward one parent.
β_q	SBX spread factor	Scaling coefficient drawn from the distribution in Eq. (5); governs offspring distance from parents in Simulated Binary Crossover.
q	Distribution index	Shape parameter of SBX; larger q offspring closer to parents (exploitation), smaller q broader spread (exploration).
$u \sim \mathscr{U}(0,1)$	Uniform variate	Random number used to sample β_q in SBX.
x_i	Gene value	Allele at position i before mutation.
x_i'	Mutated gene	Result of bit-flip (Eq. 1.12) or Gaussian perturbation (Eq. 1.13).
σ	Mutation step size	Standard deviation of the Gaussian noise added to real-valued genes; may be static or self-adaptive.
$f(x)$	Raw objective	Original cost or utility function to be optimised.

Continued on next page

Table 1.7 – continued

Symbol	Meaning	Description
$g_c(x)$	Constraint violation	Amount by which solution x violates constraint c; positive values indicate infeasibility.
λ	Penalty weight	Multiplier in the penalised fitness $F(x) = f(x) + \lambda \sum_c \max\{0, g_c(x)\}$; balances optimisation and feasibility.
$F(x)$	Penalised fitness	Fitness value after adding constraint penalties; guides selection in constrained problems.

1.5.2 The Genetic Algorithm Framework

The canonical GA proceeds through the loop summarised in Table 1.8. Although deceptively simple, each step encapsulates numerous design choices whose interplay largely determines performance.

Table 1.8: Canonical GA workflow.

Step	Action	Key references
Initialise	Randomly generate N chromosomes	[56]
Evaluate	Compute fitness for every individual	[72]
Select	Biased sampling of parents	[73, 74]
Crossover	Recombine parents (p_c)	[75]
Mutate	Stochastically alter genes (p_m)	[76]
Replace	Form next generation	[77]
Terminate	Check stopping criteria	[78]

Selection: For fitness-proportionate (roulette-wheel) sampling, an individual i with fitness f_i is chosen with probability

$$P(i) = \frac{f_i}{\sum_{j=1}^{N} f_j} \tag{1.8}$$

but alternative schemes—rank, tournament, SUS—modulate selection pressure in more controlled fashion [73, 74].

Crossover: On binary strings, single-point crossover concatenates prefixes and suffixes of two parents:

$$\begin{aligned} \text{Offspring}_1 &= [x_1, \dots, x_k, y_{k+1}, \dots, y_n], \\ \text{Offspring}_2 &= [y_1, \dots, y_k, x_{k+1}, \dots, x_n]. \end{aligned} \tag{1.9}$$

where the cut point is k [50]. Arithmetic crossover for real vectors blends parental genes:

$$\begin{aligned} \text{Offspring}_1 &= \alpha\, x + (1-\alpha)\, y, \\ \text{Offspring}_2 &= (1-\alpha)\, x + \alpha\, y \end{aligned} \tag{1.10}$$

with $\alpha \in [0,1]$ [75]. Simulated Binary Crossover (SBX) generalises this via a spread factor β_q drawn from

$$\beta_q = \begin{cases} (2u)^{\frac{1}{q+1}}, & u \le \tfrac{1}{2}, \\ \big[2(1-u)\big]^{-\frac{1}{q+1}}, & u > \tfrac{1}{2}, \end{cases} \qquad u \sim \mathcal{U}(0,1). \tag{1.11}$$

where q controls parent–offspring similarity [83].

Mutation: The classical bit-flip mutation

$$x_i' = \begin{cases} 1 - x_i, & \text{with probability } p_m, \\ x_i, & \text{otherwise.} \end{cases} \tag{1.12}$$

is replaced in real-coded GAs by additive Gaussian noise

$$x_i' = x_i + \varepsilon, \qquad \varepsilon \sim \mathcal{N}\big(0, \sigma^2\big) \tag{1.13}$$

where σ may itself adapt over time [92]. Permutation encodings rely on swap, insertion, or inversion operators [84].

Replacement and elitism: Generational schemes discard all parents; steady-state variants overwrite only a few individuals [77]. Elitism guarantees that the best-so-far chromosome survives intact, accelerating progress but imperilling diversity [53, 72].

Termination: Stop after a fixed generation limit, upon fitness stagnation, or when a target fitness is attained; combining criteria is prudent [78].

1.5.3 Encoding Strategies (Representation)

A GA's power is mediated by how it encodes solutions. Binary strings are concise and operator-friendly but coarse for real variables [50]. Real-valued vectors provide precision and natural semantics for continuous domains [82]. Permutation encodings elegantly represent orderings such as routes or schedules while demanding specialised operators [84]. Tree encodings, emblematic of Genetic Programming, evolve symbolic expressions or program parse trees [66]. When constraints loom large, hybrid or repair-based encodings are indispensable [85]. Good representations minimise *epistasis*—undesirable gene-interaction entanglements—thereby allowing crossover to recombine "building blocks" effectively [80].

1.5.4 Fitness Evaluation

Fitness is the compass of evolution. In constrained problems, a penalised fitness

$$F(x) = f(x) + \lambda \sum_{c} \max\{0,\ g_c(x)\} \tag{1.14}$$

discourages infeasible solutions by a penalty weight λ [88]. Multi-objective GAs employ Pareto dominance and diversity preservation (e.g., NSGA-II) to approximate the Pareto front [86]. Whenever a single fitness call is expensive—say, a fluid-dynamics simulation—surrogate models (Gaussian processes, neural nets) slash run-time by predicting fitness and reserving true evaluations for the most promising candidates [87, 91]. Careful scaling or ranking prevents runaway selection pressure or stagnation [74].

1.5.5 Selection Mechanisms

An effective selection mechanism reconciles two antagonistic imperatives: *exploitation*, which accelerates convergence by rewarding the fittest; and *exploration*, which guards against genetic monoculture. Roulette-wheel sampling (Equation (1.8)) is intuitive but sensitive to raw-fitness scaling; rank, tournament, and SUS offer steadier control [73, 74]. Elitism, while boosting speed, must be throttled to avert premature convergence [53, 72].

1.5.6 Crossover, Mutation, and Building-Block Theory

The building-block hypothesis suggests that short, low-order schemata of above-average fitness proliferate exponentially under selection and are recombined by crossover into higher-order structures [51, 52]. Crossover therefore sits at the heart of GA potency, while mutation serves as an insurance policy against stagnation. Parameter studies typically set $p_c \in [0.6, 0.9]$ for exploration and p_m an order of magnitude lower for cautious novelty [76].

1.5.7 Convergence and Termination

Under steady selective pressure, population variance shrinks and mean fitness rises. If diversity collapses too fast, the GA freezes on a local optimum; adaptive mutation, niching, crowding, and restarts inject new genetic material [60, 89]. Termination rules—generation caps, fitness plateaus, population homogeneity thresholds, or external resource budgets—balance thoroughness against pragmatism [78]. Because convergence offers no guarantee of global optimality, practitioners typically run multiple independent GA instances and compare best outcomes.

1.5.8 Challenges and Limitations

Notwithstanding their appeal, GAs can be finicky. They are parameter-sensitive, sometimes compute-hungry, and theoretically unruly in finite populations [55]. Complex constraints are difficult to encode, and high-dimensional spaces erode building-block integrity [80]. Hybrid metaheuristics, surrogate-assisted modelling, island models, and adaptive parameter control remain active countermeasures [68, 87].

1.5.9 Summary of GA

In summary, Genetic Algorithms stand as one of the great workhorses of modern optimisation. Their marriage of simple operators and population-level search yields an algorithm that is at once conceptually elegant, embarrassingly parallel, and surprisingly powerful on problems that defy conventional techniques. When armed with an insightful encoding, judicious parameter settings, diversity-preserving safeguards, and—where appropriate—synergistic hybrid enhancements, a GA can discover solutions of striking quality in landscapes that have little else to recommend them but their challenge. Ongoing research in adaptive control, machine-learning-guided search, and hardware-accelerated implementations ensures that GAs will remain an indispensable tool in the optimisation arsenal for many years to come.

1.6 Result-Focused Contributions

Over the course of our research, we progressively developed and refined a series of approaches to tackle the dynamic resource allocation problem in fog and cloud computing environments. Our initial work began with formulating the resource allocation task as a Markov Decision Process (MDP) and applying fundamental reinforcement learning (RL) algorithms to optimize task assignment between fog and cloud layers. Building upon this foundation, we extended the model to multi-fog scenarios, introducing fog node selection mechanisms alongside RL-based scheduling to improve load balancing and responsiveness. In the final stage, we enhanced the RL-based framework by

integrating a Genetic Algorithm (GA), aiming to accelerate convergence and further improve the system's adaptability and performance. Each step of this evolution was validated through simulations and empirical evaluation, providing a clear performance trajectory from basic RL methods to advanced hybrid solutions.

1: ***Fog Node Resource Allocation in Fog-Cloud Systems Using Reinforcement Learning***

In our recent publication titled Fog Node Resource Allocation in Fog-Cloud Systems Using Reinforcement Learning, presented at the IEEE Consumer Communications and Networking Conference (CCNC 2025) [96], we proposed a reinforcement learning-based framework for dynamic resource allocation in fog computing environments. Among the three evaluated algorithms—Q-learning, SARSA, and E-SARSA—the E-SARSA variant demonstrated superior performance. It achieved the fastest convergence, highest fog resource utilization, and best adaptability to dynamically changing request patterns. Notably, E-SARSA significantly reduced average task delay and minimized cloud offloading by autonomously learning optimal allocation strategies without relying on manually defined threshold parameters. The simulation results further confirmed the model's ability to prioritize time-sensitive requests, adapt to variable workloads, and effectively manage multi-dimensional, integer-valued resource constraints, making it well-suited for real-time fog-cloud environmentss.

2: ***Resource Allocation in Multi-Fog/Cloud Systems Using Reinforcement Learning***

In the work submitted to the IEEE International Conference on Big Data and Cloud Computing (BDCloud 2025) [98], we introduced a two-phase task scheduling framework for fog-cloud environments, combining rule-based fog node selection strategies with reinforcement learning. The system was evaluated under various task allocation strategies, including Selection-1, Selection-2, Round Robin, Random allocation, and a Genetic Algorithm. Among these, the Selection-2 + RL combination consistently outperformed all other methods. It achieved the lowest average task completion time and exhibited significantly better load balancing, as measured by reduced variance in fog node utilization. Furthermore, the approach effectively minimized cloud offloading, maximizing local fog resource usage and reducing overall system latency. The framework's reward model featured tunable parameters, ζ and θ, allowing the scheduler to dynamically prioritize between delay minimization and resource efficiency based on real-time system requirements, thereby enhancing the flexibility and responsiveness of the resource allocation process.

3: ***Resource Allocation in Multi-Fog/Cloud Systems Using a Hybrid Genetic Algorithm-Reinforcement Learning Approach***

In our third contribution, presented in the paper Resource Allocation in Multi-Fog/Cloud Systems Using a Hybrid Genetic Algorithm–Reinforcement Learning Approach, accepted at the IEEE International Conference on Cloud Computing (CLOUD 2025) [97], we extended the reinforcement

learning framework by incorporating a Genetic Algorithm (GA) to form a hybrid GA-RL model. This integration resulted in marked improvements across all evaluated performance metrics. Compared to standalone RL methods, the hybrid approach achieved faster convergence, lower average task completion time, and a significant reduction in cloud-offloaded requests. A key highlight of the hybrid model was its ability to deliver better load balancing, demonstrated by a lower standard deviation in fog node utilization, which reflects more equitable and efficient resource allocation across the fog layer. Extensive simulations involving workloads ranging from 100 to 1000 tasks validated the approach's scalability and robustness, showing consistent adaptability to fluctuating request rates and resource availability. These results underscore the hybrid GA-RL strategy as a powerful and adaptive solution for dynamic, latency-sensitive fog/cloud environments.

Table 1.9: Summary of Symbols and Variables Used in the thesis

Symbol	Meaning	Units
Sets, indices, and entities		
k	Request / task index	–
i	Resource type index, $i \in \{1, \ldots, m\}$	–
f	Fog node index (multi-fog chapters)	–
c	Cloud entity (logical destination)	–
$\mathscr{F}$	Set of fog nodes	–
$\mathscr{N}(f)$	Neighbor set of fog node f (if neighbor-forwarding is enabled)	–
$\mathscr{A}(t)$	Set of *active* requests at time t (resources still reserved, not expired)	–
$\mathscr{K}$	Set of all requests in an experiment/run	–
Resource model and feasibility		
m	Number of resource dimensions (e.g., CPU, RAM, BW)	–
C_i	Capacity of resource i on a fog node	resource units
$req_{k,i}$	Demand of request k on resource i (same meaning as $u_{k,i}$)	resource units

Continued on next page

Symbol	Meaning	Units
$u_{k,i}$	Demand of request k on resource i (alias of $req_{k,i}$; keep only one in final text)	resource units
$s_{t_k,i}$	Residual (available) amount of resource i at decision time t_k	resource units
$\mathbf{C}$	Capacity vector $\mathbf{C} = [C_1, \ldots, C_m]$	resource units
$\mathbf{req}_k$	Demand vector $\mathbf{req}_k = [req_{k,1}, \ldots, req_{k,m}]$	resource units
ρ_{stab}	Small positive constant for numerical stability (avoid division by zero)	–
$canAllocate$	Boolean: resource-feasibility check (all demanded resources available)	–
$\mathscr{A}_k^{\text{feas}}$	Feasible action set for request k (delay & resource constraints)	–
Timing, service, and workload		
t_k	Arrival/decision time for request k	s
τ_k	Service time (reservation holding time) of request k	s
T_k^{srv}	Service time of request k (alias of τ_k)	s
λ_{jobs}	Request arrival rate (if modeled as a stochastic process)	requests/s
N	Number of requests in a GA batch/window being optimized	–
H	Lookahead horizon (if used for batching/GA windowing)	s or steps
Delay / latency components		
D_k^{tot}	Total end-to-end delay experienced by request k	s
D_k^{net}	Network delay component (propagation + transmission, if separated)	s
D_k^{prop}	Propagation delay component	s
D_k^{tx}	Transmission delay component	s
D_k^{queue}	Queueing delay at the selected node	s

Continued on next page

Symbol	Meaning	Units
D_k^{proc}	Processing (compute) time at the selected node	s
$\widehat{D}_{i,k}$	Predicted total delay if request k is assigned to fog node i	s
$\widehat{D}_k(a)$	Predicted total delay for request k under action $a \in \{\text{Fog}, \text{Cloud}\}$	s
$d_k^{\max}$	Maximum tolerable delay (SLA deadline) for request k	s
$\bar{D}$	Average delay over a set/batch of requests (context-dependent)	s
$d_{\text{avg}}(k)$	Running/estimated average delay used inside reward shaping (if defined)	s
$d_{\min}$	Small reference delay to clamp denominators (stability)	s
Actions, decisions, and routing		
a_k	Action taken for request k (e.g., Fog/Neighbor/Cloud depending on chapter)	–
Fog	Action: process/assign to fog (local or selected fog node)	–
Cloud	Action: offload to cloud	–
$f^\star$	Selected fog node for a request after candidate filtering / scoring	–
$FogNodeList$	Candidate set/list of fog nodes that satisfy feasibility checks	–
q_k	Requeue count for request k (to prevent infinite deferral loops)	–
$q_{\max}$	Maximum allowed requeues before dropping a request	–
Reinforcement Learning (MDP / Q-learning family)		
s	RL state (feature vector capturing system status at decision time)	–
s_k	State observed when handling request k	–

Continued on next page

Symbol	Meaning	Units
a	RL action in a state (assignment decision)	–
r_k	Immediate reward after decision for request k	–
$Q(s,a)$	Action-value function	–
α	Learning rate	–
γ	Discount factor	–
ε	Exploration probability in ε-greedy selection	–
$\pi(a\|s)$	Policy (probability of choosing action a in state s)	–
δ	TD error (if explicitly written)	–
w	Weight vector for linear function approximation (if used)	–
Reward shaping parameters (as used in your chapters)		
ζ	Weight of the resource-headroom / utilization term in the reward	–
θ	Weight of the delay-consistency / SLA term in the reward	–
$\Phi(\cdot)$	Constraint-penalty term (generic; used in GA objective or reward penalties)	–
Genetic Algorithm (GA) / Hybrid GA–RL		
$\mathbf{x}$	Chromosome / candidate allocation plan (encoding depends on chapter)	–
P	Population of chromosomes	–
P_{size}	Population size	–
G	Number of generations	–
$Fit(\mathbf{x})$	Fitness value of chromosome $\mathbf{x}$	–
$\mathscr{J}(\mathbf{x})$	GA objective/cost function minimized by GA	–
ω_D	GA weight for delay term in $\mathscr{J}(\mathbf{x})$ (distinct from ζ, θ)	–
ω_C	GA weight for cloud-offload term in $\mathscr{J}(\mathbf{x})$	–

Continued on next page

Symbol	Meaning	Units
$C(\mathbf{x})$	Number (or fraction) of requests assigned to cloud under plan $\mathbf{x}$	count or ratio
$\overline{D}(\mathbf{x})$	Mean predicted delay under plan $\mathbf{x}$	s
λ	Penalty coefficient for constraint violations in GA objective	–
p_c	Crossover probability	–
p_m	Mutation probability	–
E	Elitism count (number of best chromosomes carried forward)	–
$Sel(\cdot)$	Selection operator (e.g., roulette, tournament)	–
$Xover(\cdot)$	Crossover operator	–
$Mut(\cdot)$	Mutation operator	–
Evaluation metrics (used across experiments)		
T_{comp}	Completion time / makespan for a set of requests (as defined in figures)	s
R_{fog}	Fraction of requests processed in fog (local or any fog)	ratio
R_{cloud}	Fraction of requests offloaded to cloud	ratio
R_{nbr}	Fraction forwarded to neighbor fog nodes (multi-fog only)	ratio
SLA	SLA satisfaction indicator or rate (deadline met)	ratio
L_{95}	95th percentile latency (if reported)	s
U_i	Utilization of resource i (e.g., used/total)	ratio

1.7 Dissertation Overview

The remainder of this dissertation is organized as follows: Chapter 2 presents a comprehensive study on resource allocation within a single fog node architecture in fog-cloud environments, utilizing reinforcement learning techniques such as Q-learning, SARSA, and E-SARSA. The chapter includes formal problem modeling, Markov Decision Process formulation, and detailed simulation-based evaluation against a Fixed-Threshold baseline. Chapter 3 extends this investigation to a multi-fog/cloud

architecture, introducing a distributed scheduling framework coordinated by a central distributor node. It explores advanced reinforcement learning strategies combined with heuristic fog node selection methods to achieve low-latency, load-balanced task scheduling. In Chapter 4, a hybrid approach is introduced that integrates Genetic Algorithms with reinforcement learning to further enhance exploration efficiency and system performance in complex multi-fog/cloud environments. This hybrid GA-RL framework is evaluated across multiple scheduling strategies and demonstrates significant improvements in convergence behavior, task completion time, and load balancing. Finally, Chapter 5 concludes the dissertation by summarizing the key findings, highlighting major contributions, and outlining future research directions.

Chapter 2

Resource Allocation in Fog-Cloud Systems Using Reinforcement Learning: A Single Fog Node Perspective

2.1 Introduction

Fog computing [12] provides a decentralized alternative to traditional cloud-based architectures by extending computation, storage, and networking resources to the edge of the network. This paradigm shift is particularly advantageous for latency-sensitive and resource-constrained applications, where proximity to end devices is crucial. However, the dynamic and heterogeneous nature of fog environments introduces significant challenges in resource management—especially when operating under real-time constraints and unpredictable workloads.

This chapter investigates intelligent resource allocation in fog-cloud systems, focusing on a *single fog node* that must make autonomous decisions regarding task processing or offloading. In contrast to static heuristics or threshold-based approaches, we propose a reinforcement learning (RL)-based framework where the fog node acts as an agent embedded in a Markov Decision Process (MDP). By interacting with its environment and learning from feedback, the agent progressively improves its decision-making policy to maximize long-term system efficiency.

Three prominent RL algorithms [93]—Q-Learning, SARSA, and Expected SARSA—are implemented and evaluated against a Fixed-Threshold baseline. Each algorithm exhibits unique learning dynamics and trade-offs in exploration and exploitation, allowing for a nuanced understanding of their behavior in resource-constrained environments.

Simulated experiments are conducted using synthetically generated datasets, as no suitable real-world benchmark is available. These simulations encompass diverse task request profiles—small,

average, and large—to test the framework's adaptability. Key evaluation metrics include average task completion time, fog utilization rate, cloud offloading ratio, and load balancing index.

In addition to convergence speed and reward evolution, we analyze the distribution of learned Q-values across different resource states to interpret the agent's strategy. The results demonstrate the advantages of learning-based scheduling in reducing delay, improving local resource utilization, and adapting to varied environmental conditions without manual intervention.

The remainder of this chapter is organized as follows: Section 2.4 presents the RL-based framework for resource allocation, including the MDP formulation, learning objectives, and the three RL algorithms used for task scheduling and decision-making. Section 2.5 provides an in-depth analysis of simulation results, comparing the RL algorithms against the Fixed-Threshold baseline and offering insights on convergence behavior, Q-value trends, and action preferences. Finally, Section 2.6 summarizes the key findings and motivates the multi-fog extension explored in Chapter 3.

2.2 Problem Statement

The emergence of fog computing has significantly reshaped the landscape of resource management by shifting computational power closer to the network edge. This proximity enables faster response times and better support for latency-sensitive applications. In such dynamic environments, where both workloads and resource availability fluctuate over time, effective resource allocation becomes critical to ensuring timely and efficient service delivery.

In the context of fog-cloud systems, user requests are modeled as tuples

$$(u_1, u_2, \ldots, u_m, d_{\max}, L). \tag{2.1}$$

where u_1 through u_m represent the quantities of m different resource types (e.g., CPU, memory, bandwidth), $d_{\max}$ denotes the maximum tolerable end-to-end delay for that request, and L is the service time (the duration for which allocated resources remain reserved in the fog node). This representation succinctly expresses both the quantitative resource needs and the quality-of-service (QoS) constraints associated with each request, while also capturing *service-time persistence*.

A key challenge in fog-based systems is managing the resource allocation process within a single fog node while making optimal decisions on whether to serve incoming tasks locally or offload them to the cloud. This decision-making process is inherently complex due to several interdependent factors:

- The high dimensionality of the resource state space arising from multiple resource types.
- The variability in task arrival rates and their respective delay sensitivities.

- The limited capacity of the fog node compared to the (assumed) effectively unlimited resources of the cloud.

Each potential resource allocation state is defined by the current availability levels of all resource types. Transitions between these states occur as tasks are served and resources are consumed or released. Determining whether a new request can be served within the fog node requires evaluating current resource availability and comparing it against the request's requirements. Simultaneously, the system must assess whether the request's delay tolerance can be satisfied if it is served locally.

When a request cannot be fulfilled locally—either due to insufficient resources or delay constraints—it may be offloaded to the cloud. Although cloud servers provide substantial processing capacity, offloading comes at the cost of additional transmission and end-to-end delay. Hence, the system must carefully balance the trade-off between leveraging local (fog) resources and deferring execution to the cloud, particularly when low latency is a priority.

To automate this decision-making process, early approaches employ static threshold-based strategies, where tasks are offloaded to the cloud once the fog node's utilization surpasses a predefined limit. While this offers a simple baseline mechanism, such approaches are not adaptive to dynamic conditions and often require manual tuning to remain effective.

Fig. 2.1 illustrates the simplified system architecture considered in this study. It includes a single fog node directly connected to a cloud server, with user requests arriving at the fog node for evaluation. This model serves as a foundational testbed for studying localized resource allocation strategies and for developing adaptive decision-making mechanisms that respond to evolving system conditions in real time.

To highlight the dynamic nature of this decision space, Fig. 2.2 depicts state transitions within the fog node. Each state corresponds to a specific configuration of available resources, and transitions are driven by incoming request attributes and allocation outcomes. This visual representation reinforces the need for an intelligent, adaptive mechanism capable of navigating a complex, time-evolving environment to optimize resource usage while preserving service quality.

2.3 Optimization Problem

To formalize the task scheduling challenge in a Fog–Cloud system, we model the decision-making process of a *single fog node* as an optimization problem. For each arriving request, the node must decide whether to execute the request locally (using its limited resources) or to forward it to the cloud. This decision must respect (i) *resource feasibility* across multiple heterogeneous resource types and (ii) *delay feasibility* with respect to the request's delay tolerance. The overarching objective is to maximize effective fog utilization—serving as many admissible requests locally as possible—while reducing dependence on cloud execution, which typically incurs higher end-to-end latency.

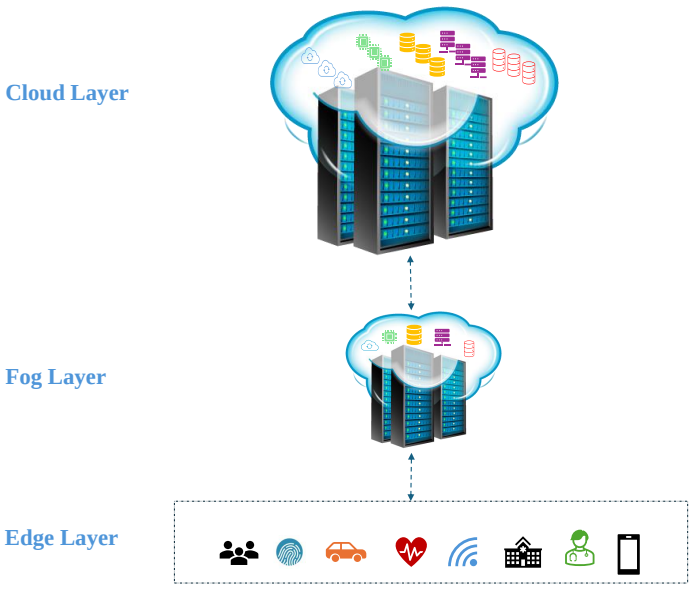

Figure 2.1: System under consideration (single fog node with cloud fallback).

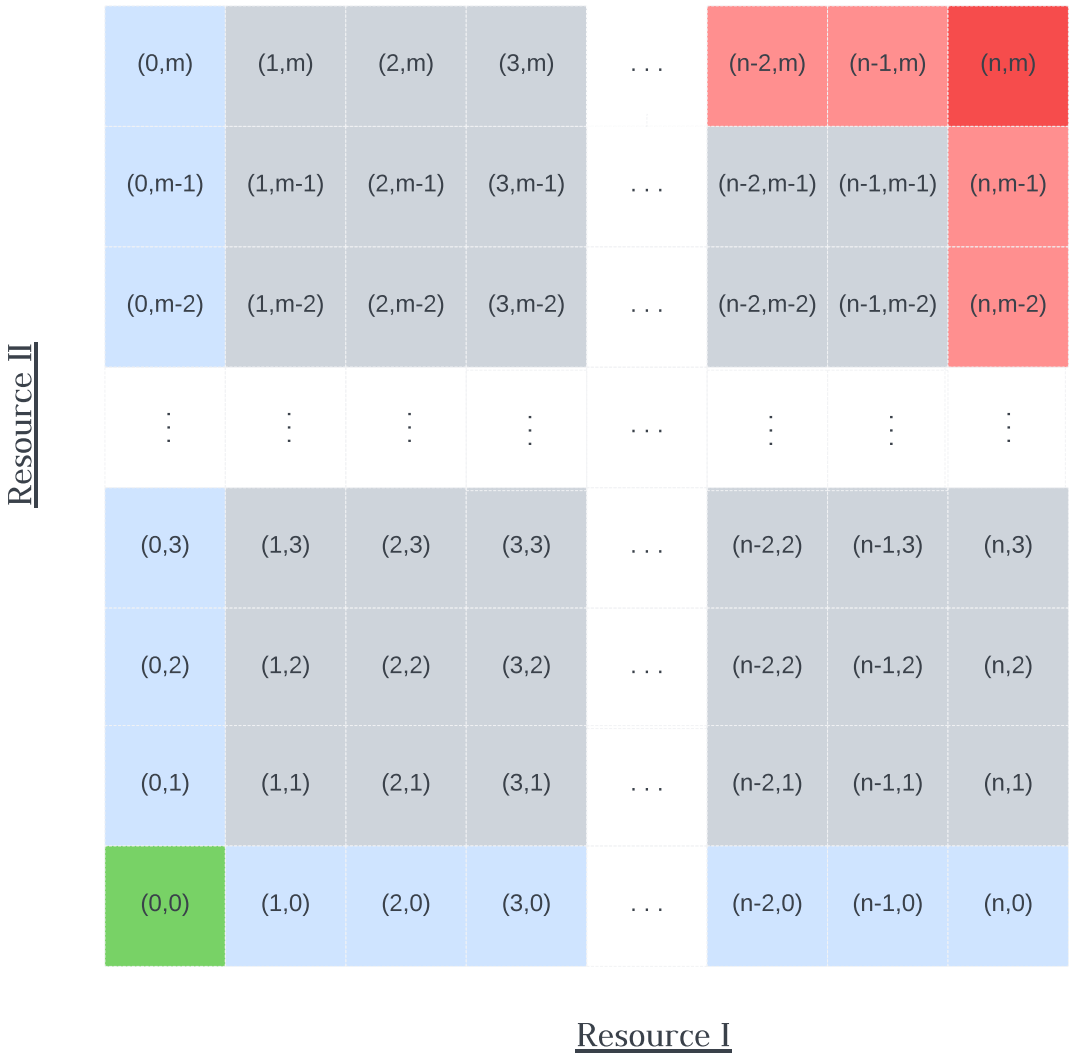

Figure 2.2: Resource-state transitions in the fog node.

Each request k is represented by a tuple

$$\mathbf{u}_k = \left(u_{k,1}, u_{k,2}, \ldots, u_{k,m}, d_k^{\max}, L_k\right). \tag{2.2}$$

where $u_{k,i}$ is the required amount of resource type $i \in \{1, \ldots, m\}$ (e.g., CPU, memory, bandwidth), $d_k^{\max}$ is the maximum end-to-end delay tolerated by the user (delay tolerance), and L_k is the *service time* (i.e., the duration for which the admitted request keeps its allocated resources reserved at the fog node). The inclusion of L_k is essential: because resources remain held during L_k, local admission decisions couple over time through persistent resource occupation and queueing.

System Model and Notation. We consider a fog–cloud system composed of a single fog node and a remote cloud server, as illustrated in Fig. 2.1. The fog node offers m resource types with capacity vector $\mathbf{C} = (C_1, \ldots, C_m)$, and its instantaneous resource availability is represented by the state $\mathbf{s}_t = (s_{t,1}, \ldots, s_{t,m})$, where $0 \leq s_{t,i} \leq C_i$ denotes the currently available amount of resource i at decision time t.

A request can be admitted and executed locally at the fog node if $\mathbf{u}_k \preceq \mathbf{s}_{t_k}$ (element-wise feasibility) and if its realized completion delay satisfies the QoS constraint $d_k \leq d_k^{\max}$. Otherwise, the request is offloaded to the cloud, which is assumed to have sufficient capacity but incurs additional communication delay. Hence, each scheduling decision must balance (i) local execution to reduce offloading and exploit proximity, and (ii) proactive offloading to avoid violating delay constraints and overloading the fog node.

We introduce a binary decision variable $Y_k \in \{0, 1\}$ indicating whether request k is served locally by the fog node ($Y_k = 1$) or offloaded to the cloud ($Y_k = 0$). Since each request is indivisible, a request can only be executed locally if *all* its resource demands are simultaneously feasible; therefore, feasibility is enforced through element-wise constraints.

To model persistence due to non-zero service times, let $\mathscr{A}(t)$ denote the set of *active* requests currently being served (i.e., requests whose service has started but not yet completed) at decision time t. Then the residual resource on dimension i at time t is

$$s_{t,i} \;=\; C_i \;-\; \sum_{j \in \mathscr{A}(t)} u_{j,i}. \tag{2.3}$$

This captures the fact that previously admitted requests continue to occupy resources until their service times elapse, which in turn influences the feasibility of admitting new requests.

To capture the delay context experienced by the fog node, we define a running average delay over previously *locally served* requests. Let $N_t^{\text{fog}} = \sum_{j=1}^{t-1} Y_j$ denote the number of locally served requests

prior to step t; then

$$d_{\text{avg}}(1) = 0, \qquad d_{\text{avg}}(t) = \begin{cases} \frac{1}{N_t^{\text{fog}}} \sum_{j=1}^{t-1} Y_j d_j, & N_t^{\text{fog}} > 0, \\ 0, & N_t^{\text{fog}} = 0, \end{cases} \quad t \geq 2. \tag{2.4}$$

where d_j denotes the realized end-to-end delay of request j when executed locally.

To avoid undefined ratios (e.g., division by zero) and to keep the objective numerically stable under very small denominators, we introduce a small positive constant $\rho_{\text{stab}} > 0$. In addition, we clamp the running-average delay used in ratio terms via

$$d_{\text{avg}}^{\text{eff}}(k) \;=\; \max\{d_{\text{avg}}(k), d_{\min}\}, \qquad d_{\min} > 0, \tag{2.5}$$

so that the delay-consistency component remains well-defined and bounded in early learning stages.

Under these definitions, the fog-side local scheduling objective can be written as:

$$\begin{aligned} \max_{\{Y_k\},\{r_k\}} \quad & \sum_{k=1}^{n} \left(\alpha_w Y_k r_k \;-\; \beta_w \frac{1 - Y_k}{r_k + \rho_{\text{stab}}} \right) \\ \text{Subject to:} \quad & Y_k u_{k,i} \;\leq\; s_{t_k,i}, \quad \forall k = 1,\ldots,n,\ \forall i = 1,\ldots,m \quad \textit{i.} \\ & Y_k d_k \;\leq\; Y_k d_k^{\max}, \quad \forall k = 1,\ldots,n \quad \textit{ii.} \\ & r_k = \zeta \sqrt{\prod_{i=1}^{m} \frac{s_{t_k,i}}{u_{k,i} + \rho_{\text{stab}}}} \;+\; \theta \sqrt{\frac{d_k^{\max} + \rho_{\text{stab}}}{d_{\text{avg}}^{\text{eff}}(k) + \rho_{\text{stab}}}}, \quad \forall k = 1,\ldots,n \quad \textit{iii.} \\ & \alpha_w + \beta_w = 1, \qquad \zeta + \theta = 1 \quad \textit{iv.} \\ & \alpha_w, \beta_w, \zeta, \theta \in [0,1], \quad Y_k \in \{0,1\}, \quad \rho_{\text{stab}} > 0. \quad \textit{v.} \end{aligned} \tag{2.6}$$

Interpretation. The objective rewards feasible local service through the term $\alpha_w Y_k r_k$ and penalizes non-local service (cloud offloading) through the term $\beta_w(1 - Y_k)/(r_k + \rho_{\text{stab}})$. Constraint (2.6i) enforces multi-resource feasibility at decision time t_k using the instantaneous residual state $s_{t_k,i}$, which already accounts for resources held by active requests over their service times. Constraint (2.6ii) enforces per-request delay feasibility whenever the request is served locally. The reward definition in (2.6iii) combines (a) a resource-efficiency term that increases when residual resources are large relative to the request's demand and (b) a delay-consistency term that is larger when the system's running delay context remains *low* relative to the request's tolerance, with ζ and θ controlling the trade-off. Finally, constraints (2.6iv)–(2.6v) define admissible weight ranges and ensure numerical stability.

Overall, this formulation captures the fog node's central decision trade-off: admit a request locally

only when its multi-resource demands can be satisfied *given currently held resources* (service-time persistence) and when its delay tolerance can be met, while discouraging excessive reliance on cloud execution via an explicit penalty structure.

2.4 Reinforcement Learning-Based Solution

The optimization model in Section 2.3 provides a principled description of the fog node's admission decision under multi-resource feasibility and delay tolerance. However, in realistic fog environments, task arrivals, service times, and residual-resource evolution are inherently dynamic and partially uncertain. This motivates a model-free reinforcement learning (RL) formulation in which the fog node learns an allocation policy directly from interaction with the environment, without assuming prior knowledge of workload statistics or transition dynamics.

We model the fog node as an RL agent operating in a Markov Decision Process (MDP). At each decision step t, the agent observes the current residual-resource state, receives an arriving request, chooses an action (serve locally or offload), and obtains an immediate reward. The agent's goal is to learn an optimal policy π^* that maximizes the expected long-term return, thereby adapting admission/offloading behavior as the environment evolves.

MDP Formulation

The MDP is defined as a tuple (S, A, P, R, γ):

- **State space (S):** The state summarizes the fog node's residual capacity across m resource types at decision step t:

$$s_t = \big(s_{t,1}, s_{t,2}, \ldots, s_{t,m}\big), \qquad 0 \le s_{t,i} \le C_i. \tag{2.7}$$

  In our discrete implementation, each $s_{t,i}$ is quantized to integer levels, leading to

$$S = \{0, 1, \ldots, C_1\} \times \cdots \times \{0, 1, \ldots, C_m\}. \tag{2.8}$$

  Remark: In general, one may augment the state with request features (demands/tolerance). In this chapter, to keep the tabular state tractable and to support clear interpretation of Q-value trends, we evaluate policies under controlled request-size regimes (small/average/big), and interpret Q-values per regime as shown in Section 2.5.

- **Action space (A):** The agent selects one of two actions:

$$A = \{\text{Serve in Fog},\ \text{Offload to Cloud}\}. \tag{2.9}$$

Local service is admissible only if the request is resource-feasible (element-wise) and delay-consistent, whereas offloading is used when local execution is infeasible or predicted to violate the QoS constraint.

- **Reward function ($R(s,a)$):** After choosing action a_t for request k in state s_t, the agent receives reward r_k. Consistent with Eq. (2.6), we employ a composite reward that reflects (i) residual-resource efficiency and (ii) delay consistency. To prevent excessively large ratios in the early stages when $d_{\text{avg}}(k)$ may be close to zero, we clamp the denominator:

$$d_{\text{avg}}^{\text{eff}}(k) = \max\{d_{\text{avg}}(k), d_{\min}\},$$

and use $d_{\text{avg}}^{\text{eff}}(k)$ in Eq. (2.6) and Eq. (2.10), where $d_{\min} > 0$ is a small reference delay.

$$r_k = \zeta \sqrt{\prod_{i=1}^{m} \frac{s_{t_k,i}}{u_{k,i} + \rho_{\text{stab}}}} + \theta \sqrt{\frac{d_k^{\max} + \rho_{\text{stab}}}{d_{\text{avg}}^{\text{eff}}(k) + \rho_{\text{stab}}}}, \qquad \zeta + \theta = 1, \tag{2.10}$$

where $s_{t_k,i}$ is the residual amount of resource i at the decision time for request k, $u_{k,i}$ is the requested demand, $d_k^{\max}$ is the delay tolerance, $d_{\text{avg}}(k)$ is the running average delay over previously *locally served* requests, $d_{\min} > 0$ is a small reference delay, and $\rho_{\text{stab}} > 0$ is a stability constant. This reward provides a dense learning signal that encourages local service when it is feasible and the system remains in a low-delay regime.

- **Discount factor (γ):** A scalar $\gamma \in [0,1]$ controlling the trade-off between immediate and future rewards.

Value Functions and Bellman Optimality

Given a policy π, the state-value function and action-value function are

$$\begin{aligned} V^{\pi}(s) &= \mathbb{E}_{\pi}[\, r_t + \gamma V^{\pi}(s_{t+1}) \mid s_t = s \,], \\ Q^{\pi}(s,a) &= \mathbb{E}_{\pi}[\, r_t + \gamma Q^{\pi}(s_{t+1}, a_{t+1}) \mid s_t = s,\ a_t = a \,]. \end{aligned} \tag{2.11}$$

The optimal action-value function $Q^*(s,a)$ satisfies the Bellman optimality equation:

$$Q^*(s,a) = \mathbb{E}\left[r_t + \gamma \max_{a' \in A} Q^*(s_{t+1}, a') \mid s_t = s,\ a_t = a \right], \tag{2.12}$$

and the optimal policy is

$$\pi^*(s) = \arg\max_{a \in A} Q^*(s,a). \tag{2.13}$$

Learning Algorithms and Exploration

To solve the MDP, we evaluate three standard temporal-difference RL algorithms:

- **Q-Learning (off-policy):** Updates using the greedy estimate of the next state's best action.
- **SARSA (on-policy):** Updates using the value of the next action actually selected by the current policy.
- **Expected SARSA:** Updates using the expectation of next-state action values under the current policy's action distribution.

All methods employ an ε-greedy exploration strategy:

$$\pi(a \mid s) = \begin{cases} \text{random action} & \text{with probability } \varepsilon, \\ \arg\max_a Q(s,a) & \text{with probability } 1-\varepsilon. \end{cases}$$

Algorithm 1 Q-Learning for Fog Resource Allocation

Initialize: $Q(s,a)$ arbitrarily; learning rate α, discount factor γ, exploration rate ε
for each episode **do**
    Observe initial state s
    for each request k in the episode **do**
        Choose a using ε-greedy from $Q(s,\cdot)$
        Execute a, observe reward r and next state s'
        $Q(s,a) \leftarrow Q(s,a) + \alpha\,[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$
        $s \leftarrow s'$
    end for
end for=0

Algorithm 2 SARSA for Fog Resource Allocation

Initialize: $Q(s,a)$ arbitrarily; learning rate α, discount factor γ, exploration rate ε
for each episode **do**
    Observe initial state s; choose a using ε-greedy from $Q(s,\cdot)$
    for each request k in the episode **do**
        Execute a, observe reward r and next state s'
        Choose a' using ε-greedy from $Q(s',\cdot)$
        $Q(s,a) \leftarrow Q(s,a) + \alpha\,[r + \gamma Q(s',a') - Q(s,a)]$
        $s \leftarrow s'$, $a \leftarrow a'$
    end for
end for=0

Algorithm 3 Expected SARSA for Fog Resource Allocation

Initialize: $Q(s,a)$ arbitrarily; learning rate α, discount factor γ, exploration rate ε
for each episode **do**
    Observe initial state s
    for each request k in the episode **do**
        Choose a using ε-greedy from $Q(s,\cdot)$
        Execute a, observe reward r and next state s'
        $\mathbb{E}[Q(s',\cdot)] \leftarrow \sum_{a'} \pi(a'|s')\, Q(s',a')$
        $Q(s,a) \leftarrow Q(s,a) + \alpha\,[r + \gamma \mathbb{E}[Q(s',\cdot)] - Q(s,a)]$
        $s \leftarrow s'$
    end for
end for=0

Summary

This RL formulation equips the fog node with an adaptive admission/offloading policy that evolves as workload and residual capacity change. The next section validates the approach empirically and compares Q-Learning, SARSA, and Expected SARSA against a fixed-threshold baseline under controlled synthetic workloads and repeatable experimental settings.

2.5 Analysis and Results

This section analyzes the performance of reinforcement learning for dynamic fog resource allocation under evolving workload conditions. We compare three RL algorithms—Q-Learning, SARSA, and Expected SARSA (E-SARSA)—against a Fixed-Threshold (FT) baseline. The evaluation focuses on two core objectives: (i) maximizing local fog utilization for admissible tasks and (ii) reducing reliance on cloud offloading, especially for latency-sensitive requests. Since no publicly available dataset captures the required granularity of multi-resource fog admission decisions, we generated a synthetic dataset that supports fair, repeatable experimentation across request-size regimes and workload scales.

Experimental Setup and Objectives

We generated 10^6 synthetic task records with diverse demand characteristics. Requests were grouped into three interpretable size regimes relative to fog capacity:

- **Big-size requests**: near the upper quarter of fog capacity.
- **Small-size requests**: near the lower quarter.
- **Average-size requests**: around the middle of the capacity.

Synthetic dataset generation and demand distribution. To enable repeatable evaluation in the absence of public benchmarks, we generated 10^6 synthetic task records. Each task is represented by a resource-demand vector over m resource dimensions, scaled relative to the fog capacity $\mathbf{C} = (C_1, \ldots, C_m)$. We define three demand regimes per resource dimension:

$$\textbf{Big-size:} \quad u_{k,i} \in \left[\tfrac{3}{4}C_i,\, C_i\right], \tag{2.14}$$

$$\textbf{Small-size:} \quad u_{k,i} \in \left[0,\, \tfrac{1}{4}C_i\right], \tag{2.15}$$

$$\textbf{Average-size:} \quad u_{k,i} \in \left[\tfrac{1}{4}C_i,\, \tfrac{3}{4}C_i\right], \tag{2.16}$$

and we generate tasks using an explicit mixture distribution over these regimes. Let $S \in \{\text{small}, \text{avg}, \text{big}\}$ denote the size class. We sample

$$S \sim \text{Categorical}(\pi_{\text{small}}, \pi_{\text{avg}}, \pi_{\text{big}}), \qquad \pi_{\text{small}} + \pi_{\text{avg}} + \pi_{\text{big}} = 1, \tag{2.17}$$

then draw each demand component within the interval associated with S. In the assumption-light setting, we use uniform sampling:

$$X \sim \text{Uniform}(0,1), \qquad u_{k,i} = \begin{cases} C_i\,(0.25X), & S = \text{small}, \\ C_i\,(0.25 + 0.50X), & S = \text{avg}, \\ C_i\,(0.75 + 0.25X), & S = \text{big}, \end{cases} \qquad i = 1, \ldots, m. \tag{2.18}$$

This construction ensures that "near the lower quarter," "around the middle," and "near the upper quarter" are realized precisely and continuously. Unless otherwise stated, demands are sampled independently across resource dimensions; correlation can be introduced if desired by sharing a common latent factor across dimensions. Finally, all experiments are reproducible given the random seed used for the mixture draw and within-class sampling.

The FT baseline accepts a request if the fog is sufficiently under-utilized. Concretely, we admit locally if

$$\min_{i \in \{1,\ldots,m\}} \frac{s_{t,i}}{C_i} \;\geq\; 0.70. \tag{2.19}$$

and otherwise offload to the cloud. In contrast, RL policies adapt online by maximizing the reward in Eq. (2.10), which explicitly embeds residual-resource efficiency and delay consistency.

Workload scaling, task-length distribution, and trial averaging. We evaluated the scheduling policies under increasing workload sizes by forming episodes of length $N \in \{100, 200, \ldots, 1000\}$, sampled from the synthetic pool. For each episode, task service times were generated from a uniform distribution:

$$L \sim \text{Uniform}(500,\, 3000), \tag{2.20}$$

ensuring heterogeneous service demands across requests. The simulated environment comprised a single fog node (with $m = 5$ distinct resource types) and a single cloud node, consistent with the architecture in Fig. 2.1. To reduce stochastic variance and improve estimation accuracy, each configuration was repeated across T independent trials (different random seeds), and all metrics were reported as trial averages:

$$\overline{M} \;=\; \frac{1}{T} \sum_{t=1}^{T} M_t, \tag{2.21}$$

with $T = 20$ in our experiments. For fair comparison across methods, we fixed the weighting parameters ζ and θ at 0.5 throughout.

Convergence Behavior

Convergence criterion. We use the term *convergence* to indicate that the learning process has stabilized, meaning that additional episodes yield only marginal performance improvement. Operationally, this is observed when the episodic cumulative reward plateaus (up to small fluctuations). Let R_t denote the cumulative reward obtained in episode t, and define the moving average over a window of size W:

$$\overline{R}_t \;=\; \frac{1}{W} \sum_{k=t-W+1}^{t} R_k. \tag{2.22}$$

We consider the learning process converged when the change in moving average across consecutive windows falls below a tolerance $\delta_{\text{conv}} > 0$:

$$\left|\overline{R}_t - \overline{R}_{t-W}\right| \;<\; \delta_{\text{conv}}. \tag{2.23}$$

This criterion aligns with the visual plateau behavior in Fig. 2.3 and provides a consistent convergence definition across algorithms.

Figure 2.3 illustrates the observed stabilization of learning dynamics. In our runs, Expected SARSA converged earliest (episode 4623), followed by Q-Learning (episode 6782) and SARSA (episode 8113). The Fixed-Threshold policy does not exhibit "convergence" in the RL sense because it does not update a value function; therefore, its episode-to-episode behavior reflects only workload randomness rather than learning stabilization.

Q-Value Trends and Action Preferences

Figures 2.4–2.6 visualize the learned Q-values associated with the *cloud/offload* decision for different request-size regimes. These curves can be interpreted by linking the x-axis state progression to residual capacity. Early states correspond to high residual capacity (the fog is relatively empty), while late states correspond to low residual capacity (the fog is close to saturation). The phrase "last quarter of fog capacity" refers to the region where the residual state has fallen below roughly 25% of capacity, i.e., $s_{t,i} \in [0, 0.25C_i]$ for dominant resource dimensions.

For **big-size requests** (Fig. 2.4), the offload-related values peak earlier in the state range, reflecting that admitting a large demand can rapidly deplete residual resources and trigger future infeasibility under persistent service times. The learned policy therefore prefers earlier offloading for large jobs to prevent cascade congestion. For **small-size requests** (Fig. 2.5), offloading becomes preferable much later, because small jobs can often be accommodated even when the fog is moderately

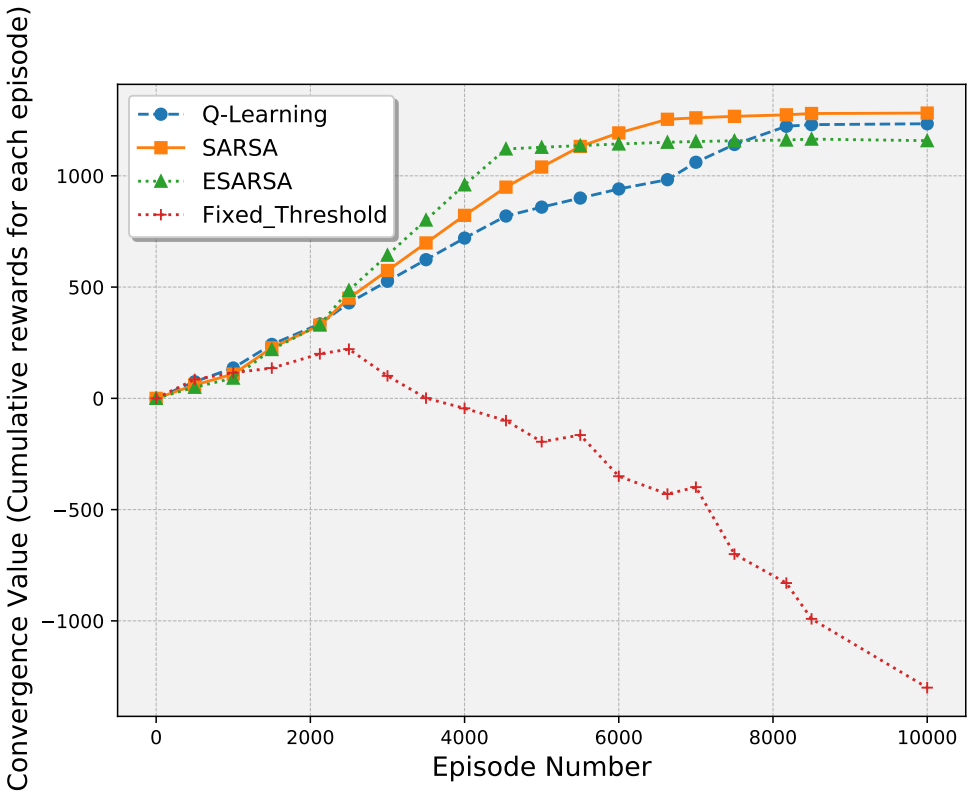

Figure 2.3: Convergence behavior of Q-Learning, SARSA, and Expected SARSA.

loaded. For **average-size requests** (Fig. 2.6), the learned offload preference rises in the mid-to-late state region, consistent with a trade-off between accepting locally when feasible and avoiding last-quarter saturation where delay violations become more likely.

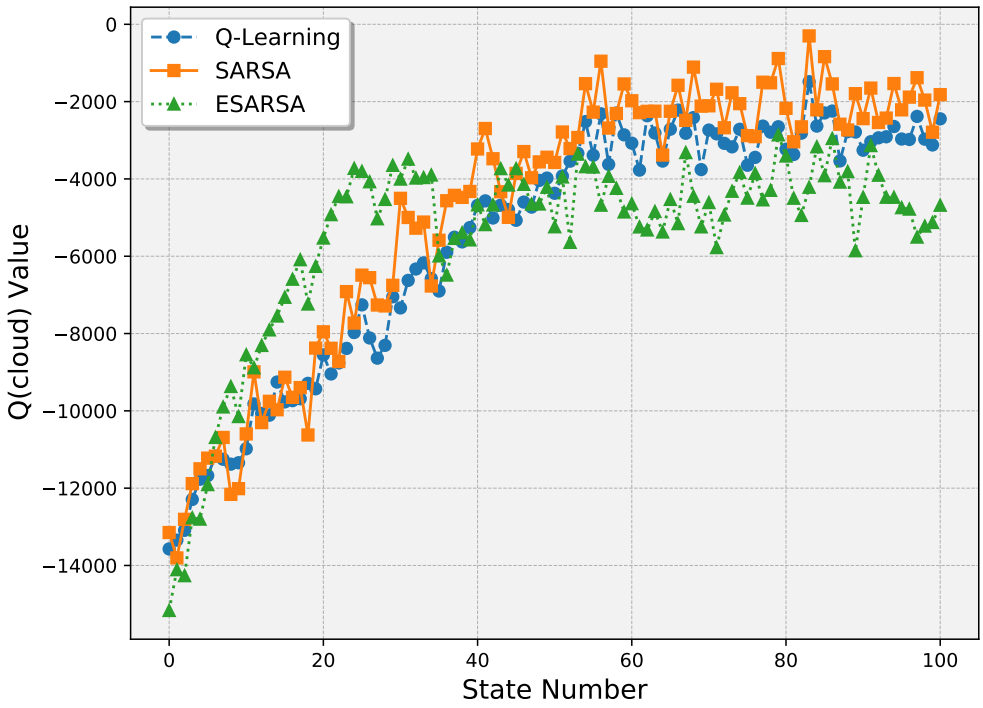

Figure 2.4: Q-value of cloud/offload action for big-size requests across states.

Across request types, the rise of offload-related Q-values in the *late* state region is consistent with learning under service-time persistence: when residual resources approach the lower quarter of capacity, admitting additional tasks increases queueing and completion times, raising the risk of violating delay tolerance. The learned behavior therefore shifts toward offloading near saturation, while still prioritizing local execution in earlier states where the fog can serve requests without destabilizing delay.

Request Fulfillment and Utilization Trends

Figure 2.7 compares the average number of requests served locally. The Fixed-Threshold approach remains around 4–5 in our setting, reflecting its inability to adapt admission decisions to request-size regime and evolving residual-resource dynamics. In contrast, all RL methods maintain higher local-serving rates because they condition decisions on state and learn when accepting locally is beneficial

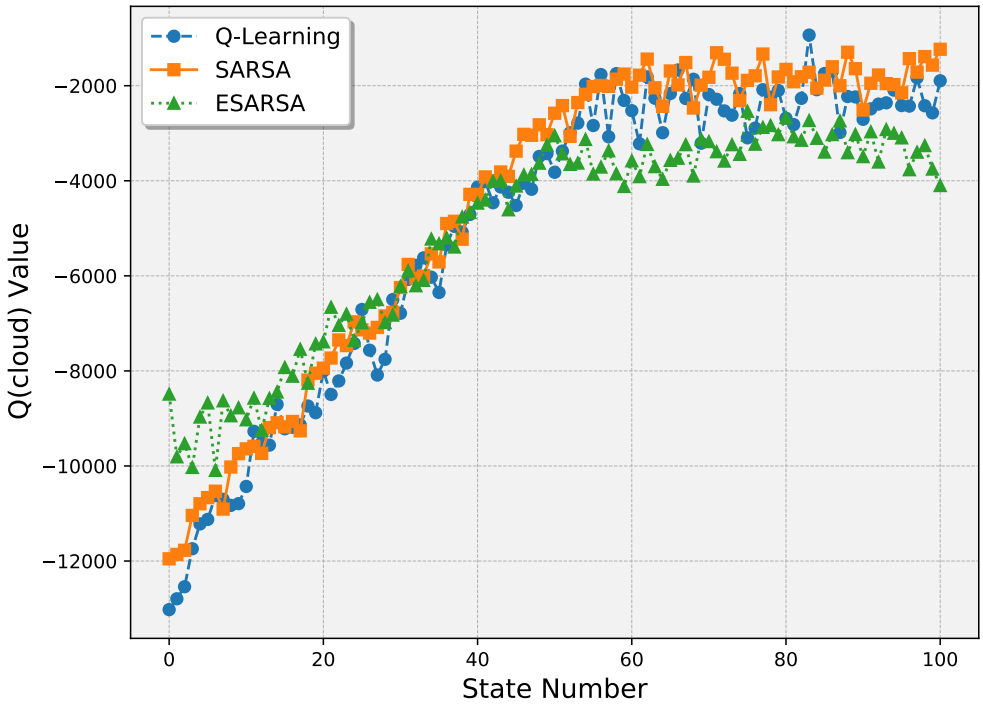

Figure 2.5: Q-value of cloud/offload action for small-size requests across states.

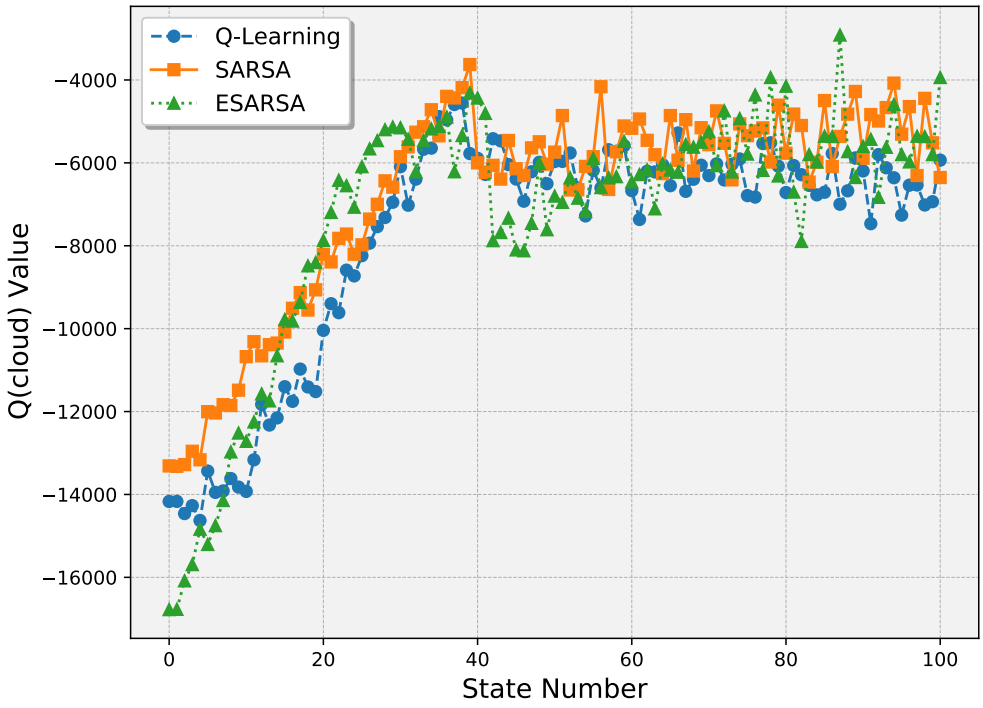

Figure 2.6: Q-value of cloud/offload action for average-size requests across states.

versus when it will induce late-state congestion and delay violations.

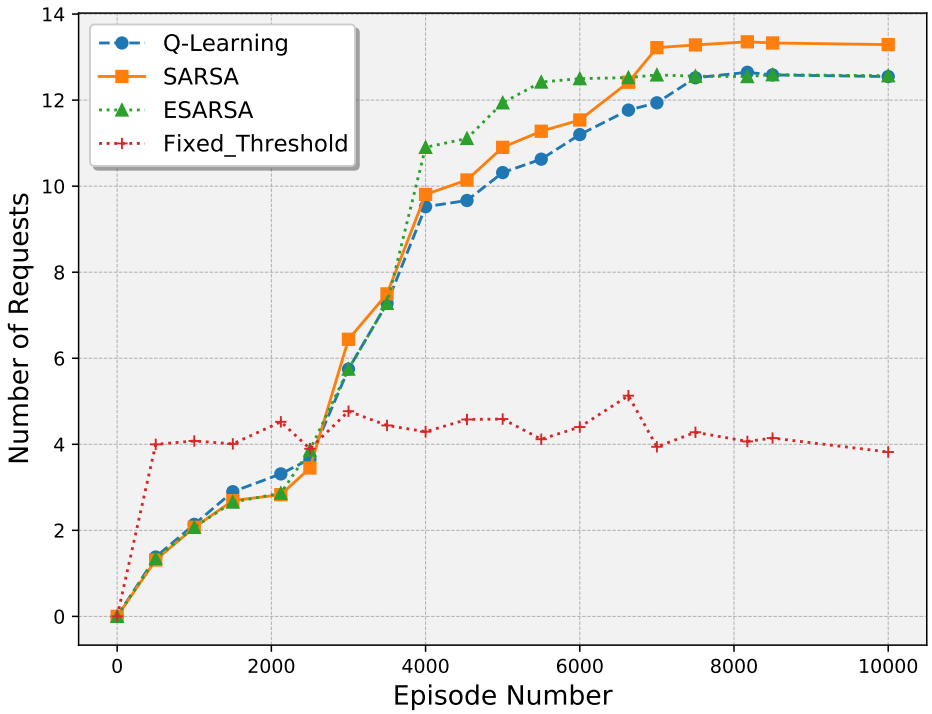

Figure 2.7: Average number of requests served locally in the fog node.

Quantitative Evaluation

Table 2.1: Performance Comparison of RL Algorithms vs. Fixed Threshold

Method	**Local Util (%)**	**Cloud Offload (%)**	**Locally Served Index (local/cloud)**
Q-Learning	88	12	7.33
SARSA	90	10	9.00
Expected SARSA	89	11	8.09
Fixed Threshold	72	28	2.57

The quantitative results confirm that RL policies consistently outperform the FT baseline in local utilization and cloud offloading behavior. The improvement is driven by state conditioning: RL methods learn that local admission is beneficial in high-residual states, but becomes risky in the late-state region (near the lower quarter of residual capacity), where service-time persistence and

Table 2.2: Insights Summary from Simulation Results

Metric	**Best Performer**	**Insight**
Average Task Time	Q-Learning	Fast initial improvement due to off-policy updates that exploit high-value actions.
Local Utilization	SARSA	Conservative on-policy updates promote stable local serving without abrupt oscillations.
Cloud Offloading Rate	SARSA	More cautious admission reduces reliance on cloud under comparable reward weights.
Load Balancing Index	Expected SARSA	Expected updates smooth learning and reduce sensitivity to action-sampling variance.
Convergence Speed	Expected SARSA	Earliest stabilization in moving-average episodic reward under the convergence criterion.
Stability	Expected SARSA	Lowest fluctuations across episodes due to expectation-based bootstrapping.

queueing amplify delay and increase the chance of violating $d_k^{\max}$. By internalizing these dynamics through reward feedback, RL policies achieve higher locally served rates while maintaining a lower offloading fraction compared to a static acceptance threshold.

2.6 Summary

This chapter studied dynamic resource allocation in a Fog–Cloud system from the perspective of a single fog node that must make online admission/offloading decisions under multi-resource feasibility and delay tolerance. We first formulated an optimization model that explicitly captures the key trade-off between serving admissible requests locally and avoiding excessive reliance on cloud execution. We then introduced a model-free RL solution in which the fog node learns an adaptive policy over a Markov Decision Process defined by residual-resource states, binary serve/offload actions, and a composite reward that balances residual-resource efficiency and delay consistency.

We evaluated Q-Learning, SARSA, and Expected SARSA against a Fixed-Threshold baseline using 10^6 synthetically generated tasks with controlled demand regimes (small/average/big relative to capacity) and heterogeneous service times. The experimental design included workload scaling from 100 to 1000 requests per episode and trial averaging across 20 independent runs to reduce stochastic variance.

Across both qualitative and quantitative analyses, RL methods demonstrated superior adaptability. Q-value trends showed that learned offloading preference increases as the system approaches the late-state region (near the lower quarter of residual capacity), consistent with the impact of service-

time persistence on queueing and delay. In aggregate metrics, all RL variants improved local utilization and reduced cloud dependence compared to the Fixed-Threshold policy, which cannot adapt to request-size regime or evolving state. Among RL methods, SARSA tended to be more conservative in offloading and yielded strong local utilization, while Expected SARSA provided the most stable learning dynamics and earliest stabilization under the moving-average convergence criterion.

The results establish RL as an effective mechanism for adaptive scheduling in latency-sensitive, resource-constrained fog environments. While this chapter focuses on a single fog node, the same methodology naturally motivates the multi-node setting addressed in subsequent chapters, where distribution, coordination, and scalability introduce additional challenges beyond the single-agent case.

Chapter 3

Resource Allocation in Fog-Cloud Systems Using Reinforcement Learning: A Multi-Fog Node Approach

3.1 Introduction

As fog computing architectures scale beyond single-node deployments, the need for intelligent and distributed resource management becomes increasingly critical. In multi-fog/cloud systems, computational tasks arrive dynamically and must be efficiently scheduled across a network of heterogeneous fog nodes with limited capacity and variable loads. Unlike the single-node scenario addressed in Chapter 2, this setting introduces additional complexity due to spatial distribution, inter-node coordination, and fluctuating network latencies.

This chapter extends the reinforcement learning (RL)-based scheduling framework introduced previously to multi-fog environments. Each fog node is modeled as an autonomous agent that makes localized task allocation decisions while collectively contributing to system-wide performance optimization. The system architecture supports task forwarding among neighboring fog nodes and to the cloud when local resources are insufficient, enabling a flexible and scalable scheduling paradigm.

To address this more complex problem space, we formulate the resource allocation problem as a distributed Markov Decision Process (D-MDP), where each agent learns to optimize its policy based on local observations and delayed environmental feedback. This approach avoids centralized bottlenecks and improves system resilience in the face of workload surges or node failures.

We investigate the behavior of this decentralized RL framework under various fog node selection strategies, including:

- **Round Robin (RR)**: Simple, cyclical task distribution.

- **Random Selection (Rand)**: Uniformly random fog node choice.
- **Threshold-Based (Fixed)**: Uses static load thresholds for routing decisions.
- **Learning-Driven (RL)**: Task assignment is dynamically guided by learned Q-values.
- **Selection-1 and Selection-2**: Heuristic-enhanced node prioritization based on residual resources and delay estimates.

Simulation-based experiments evaluate these strategies across metrics such as task completion time, cloud offloading ratio, and fog node load balancing. The results demonstrate that reinforcement learning enables more adaptive and efficient task distribution, particularly when combined with feasibility-aware node selection mechanisms.

Notation and core quantities. Throughout this chapter, the request index is denoted by k (e.g., the k^{th} request), while m denotes the number of resource types. Each request includes: (i) a *delay tolerance* $d_k^{\max}$ (the maximum end-to-end latency the request can tolerate), and (ii) a *service time* L_k (how long the allocated resources remain reserved/occupied at the chosen execution node). The end-to-end delay experienced by a request includes *queueing delay* (waiting time before execution at the selected node) and *propagation delay* (network transmission delay), and the service time L_k affects subsequent feasibility because resources remain held for L_k seconds after admission. Unless explicitly stated otherwise, fog nodes are indexed by $i \in \{1,\ldots,n\}$, while resource types are indexed by $r \in \{1,\ldots,m\}$.

The remainder of this chapter is organized as follows: Section 3.2 presents the system model and architecture, including queues and routing logic in the multi-fog/cloud setting. Section 3.3 defines the RL formulation and learning procedure. Section 3.4 presents the experimental design and evaluation results. Section 3.5 concludes the chapter with key insights and links to Chapter 4, which introduces a hybrid GA–RL approach.

3.2 Problem Statement

Fog computing shifts computation closer to the network edge, enabling low-latency service for time-sensitive applications. However, the distributed and heterogeneous nature of multi-fog systems introduces key challenges in resource allocation and task scheduling, particularly when multiple fog nodes with different capacities must cooperatively handle diverse incoming requests under QoS constraints.

In a dynamic Fog/Cloud environment, users submit service requests defined by multi-dimensional resource demands and delay constraints. Each request is expressed as a tuple:

$$\mathbf{u}_k = (u_{k,1}, u_{k,2}, \ldots, u_{k,m}, d_k^{\max}, L_k). \tag{3.1}$$

where $u_{k,1}$ to $u_{k,m}$ denote the required amounts of m resource types (e.g., CPU, memory, bandwidth), $d_k^{\max}$ denotes the maximum tolerable end-to-end delay (delay tolerance), and L_k denotes the service time (resource holding time). Requests are indivisible and independent.

Delay tolerance $d_k^{\max}$ and service time L_k. The delay tolerance $d_k^{\max}$ is an application-level QoS requirement that upper-bounds the admissible end-to-end delay. The service time L_k captures the holding time of resources after a request is admitted: once a request is executed at a fog node, the corresponding resources remain reserved for L_k seconds, after which they are released. This creates *temporal coupling* between decisions: admitting a request now reduces capacity for the next L_k seconds, influencing future feasibility and offloading behavior.

The infrastructure includes n fog nodes, denoted as $F_1, F_2, \ldots, F_n$, each equipped with m heterogeneous resource types with finite capacities. When a request arrives, the system must decide whether to:

- **Serve it locally** within a fog node,
- **Forward it** to a neighboring fog node,
- **Offload it** to the centralized cloud.

This decision must consider both resource feasibility and the delay tolerance defined by the user.

Queues and two-level decision structure. The architecture employs a distributor node that maintains a *global ingress queue* of newly arrived requests. After a request is routed to a particular fog node, that fog node may maintain a *local processing queue* (or equivalent waiting buffer), capturing the time the request waits before it begins service. This yields a natural two-level structure: (i) *global routing/selection* at the distributor (which node should receive the request), and (ii) *local admission/offloading* at the fog node (serve locally vs. cloud offload). This separation is intentional: the distributor performs lightweight feasibility-aware routing using node summaries, while each fog node adapts using richer local feedback (queueing, residual resources), which is well suited to reinforcement learning.

To facilitate intelligent routing, fog nodes periodically send status updates to the distributor. Fig. 3.1 illustrates this mechanism.

The distributor uses these updates to determine an appropriate destination for each request. Two high-level conditions can lead to cloud execution:

I. **No fog node is feasible:** none of the fog nodes can satisfy the request under current resource and delay conditions.

II. **Fog node chooses cloud:** the selected fog node (via its learned policy) may offload to the cloud when local execution is expected to degrade long-term performance.

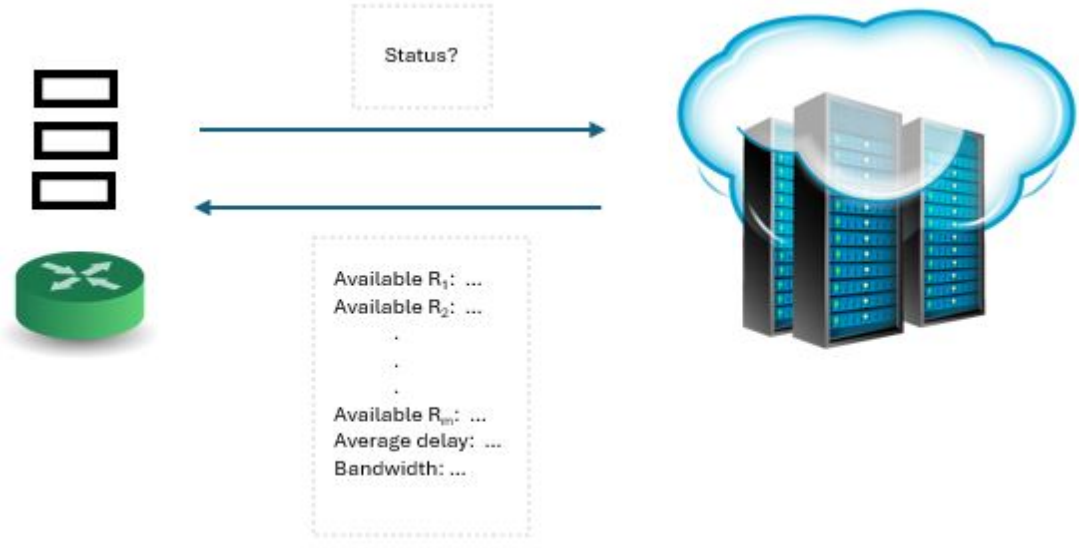

Figure 3.1: Fog nodes periodically send status updates to the distributor node to aid in scheduling decisions.

The cloud is assumed to have sufficient capacity, but incurs higher communication delay, and thus cloud routing is governed by the request delay tolerance $d_k^{\max}$.

Delay components: queueing vs. propagation. Propagation delay is the network transmission latency between the request source (or distributor) and the chosen node (fog or cloud), and is distance-/network-dependent. Queueing delay is the waiting time before a request begins service at the selected execution node; it reflects backlog created by earlier admitted requests whose resources remain held for their service times. The distributor combines node-reported delay summaries with propagation delay to estimate end-to-end delay when ranking candidate nodes.

The distributor employs two heuristic selection strategies to select a destination fog node among feasible candidates:

- **Selection-1 (potential-based):** chooses the fog node with the highest *potential* score:

$$\text{potential}_i = \frac{\prod_{r=1}^{m} R'_{i,r}}{P_i \times \text{AvgDelay}_i}, \tag{3.2}$$

  where $R'_{i,r}$ is node i's remaining amount of resource type r, P_i is the number of times node i has been selected so far (fairness counter), and AvgDelay_i is an average delay summary reported by node i.

- **Selection-2 (delay-minimization):** selects the feasible node with the smallest estimated end-

to-end delay:

$$\widehat{D}_{i,k} = \text{QueueingDelay}_{i,k} + \text{PropagationDelay}_{i,k}, \tag{3.3}$$

where $\widehat{D}_{i,k}$ is the predicted delay for request k if served by fog node i.

Clarifying Eq. (3.2): meaning, scaling, and units. Eq. (3.2) is used as a *ranking score* rather than a physical quantity. In implementation, the remaining resources $R'_{i,r}$ are naturally normalized by their capacities (or typical demand scales), and AvgDelay_i is normalized by a reference delay, yielding a dimensionless priority index. The multiplicative resource term favors broad residual capacity across resource dimensions, the divisor P_i discourages repeatedly selecting the same node (supporting fairness), and the divisor AvgDelay_i biases selection toward nodes with lower observed delay.

Clarifying AvgDelay_i and P_i. AvgDelay_i summarizes node i's local delay condition and can be computed as a moving average (or EWMA) over recent observed local delays. P_i is maintained at the distributor as a selection counter; it reduces hotspot formation by penalizing over-selected nodes.

Fog-side reward signal used by the RL agent. After the distributor assigns request k to a fog node, that fog node runs a local RL agent to decide whether to execute locally or offload to the cloud. We treat the request tolerance $d_k^{\max}$ as a *hard QoS deadline*: the agent is allowed to choose action $a \in \{\text{Fog}, \text{Cloud}\}$ only if the predicted end-to-end delay satisfies $\widehat{D}_k(a) \le d_k^{\max}$. If neither fog nor cloud can satisfy this constraint, the request is rejected (dropped). Accordingly, the RL agent optimizes performance *within* the feasible action set rather than learning from deadline violations.

To keep the expression numerically stable under small denominators, we introduce a small stability constant $\rho_{\text{stab}} > 0$. Let $s_{t_k,r} = R_r(t_k)$ denote the residual amount of resource type r at the decision time for request k, and let $\widehat{D}_k(a)$ denote the predicted end-to-end delay under action a. We define the feasible action set:

$$\mathscr{A}_k^{\text{feas}} = \Big\{ a \in \{\text{Fog}, \text{Cloud}\} \;\Big|\; \widehat{D}_k(a) \le d_k^{\max} \;\wedge\; \big(a = \text{Cloud} \;\vee\; (\forall r:\; s_{t_k,r} \ge u_{k,r})\big) \Big\}. \tag{3.4}$$

If $\mathscr{A}_k^{\text{feas}} = \emptyset$, request k is dropped; otherwise, the reward for the executed feasible action $a \in \mathscr{A}_k^{\text{feas}}$ is:

$$r_k(a) = \zeta \sqrt{\left(\prod_{r=1}^{m} \frac{s_{t_k,r}}{u_{k,r} + \rho_{\text{stab}}} \right)^{\frac{1}{m}}} + \theta \sqrt{\frac{d_k^{\max} + \rho_{\text{stab}}}{\widehat{D}_k(a) + \rho_{\text{stab}}}} - \rho_{\text{cloud}} \mathbb{I}\{a = \text{Cloud}\}, \qquad \zeta + \theta = 1. \tag{3.5}$$

The first term is a multi-resource *headroom score* (geometric mean across resource dimensions), which increases when the node has larger residual capacity relative to the request demand. The

second term is a *deadline-slack signal*, which increases when the predicted delay is comfortably below the tolerance. The last term is an explicit *cloud-avoidance penalty* that biases the policy toward fog execution when both tiers are feasible, reducing cloud reliance while still respecting deadlines. Consistent with Eq. (3.3), we estimate $\widehat{D}_k(\text{Fog})$ and $\widehat{D}_k(\text{Cloud})$ using queueing plus propagation delay at the chosen execution tier.

A schematic of this RL-driven decision process is depicted in Fig. 3.2.

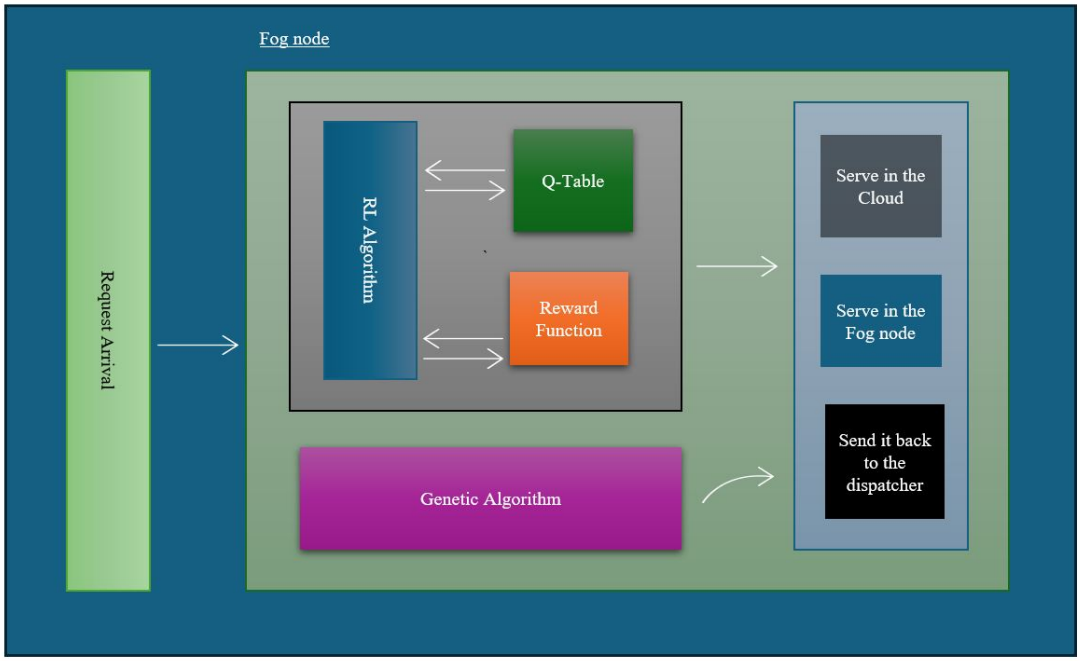

Figure 3.2: RL-based resource allocation and scheduling in individual Fog nodes.

3.3 RL-Based Scheduling Approach

This section outlines the reinforcement learning mechanism adopted by each fog node to make scheduling decisions in a decentralized manner. Each fog node operates as an autonomous agent that learns from local interaction signals: it observes its local residual resources and delay condition, selects an action (serve locally vs. offload), and receives the reward defined above. This design allows fog nodes to adapt to dynamic workloads and heterogeneous resource constraints without centralized control.

The overall scheduling pipeline is dual-phase. First, the distributor performs lightweight feasibility-aware *node selection* to decide which fog node should receive each queued request. Second, the selected fog node performs *local admission/offloading* using its learned policy. This separation provides a deployable front-end routing stage with low overhead, while enabling richer learning and

adaptation within each fog node.

3.3.1 State and Action Space

The state space S captures the residual availability of multiple resource types within a fog node. Let N_r denote the maximum capacity of resource type $r \in \{1,\dots,m\}$ and let $R_r(t)$ denote the residual (available) amount of resource type r at decision step t. A state is:

$$s_t = \big(R_1(t), R_2(t), \dots, R_m(t)\big), \qquad S = \{0,1,\dots,N_1\} \times \cdots \times \{0,1,\dots,N_m\}. \tag{3.6}$$

The action space consists of two actions:

- **Serve Locally:** accept and process the task at the fog node (reserving resources for L_k seconds).
- **Offload to Cloud:** forward the task to the cloud (consuming no local resources for that task).

3.3.2 Reward Function

The reward function guides the RL agent to balance resource efficiency and delay feasibility. Using the same notation as in Eq. (3.5), we write:

$$r_k(a) = \zeta \sqrt{\left(\prod_{r=1}^{m} \frac{s_{t_k,r}}{u_{k,r} + \rho_{\text{stab}}}\right)^{\frac{1}{m}}} + \theta \sqrt{\frac{d_k^{\max} + \rho_{\text{stab}}}{\widehat{D}_k(a) + \rho_{\text{stab}}}} - \rho_{\text{cloud}} \mathbb{I}\{a = \text{Cloud}\}, \qquad \zeta + \theta = 1. \tag{3.7}$$

This reward increases when residual resources are large relative to demand (supporting fog utilization) and when the predicted delay remains safely below the request tolerance (supporting QoS adherence). The stability constant ρ_{stab} prevents undefined ratios under small denominators. The cloud-avoidance penalty $\rho_{\text{cloud}} \geq 0$ discourages cloud execution when fog execution is feasible, thereby reducing unnecessary offloading while preserving deadline compliance.

3.3.3 Q-Learning Update and Interpretation

Q-values are updated using the Bellman optimality update:

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha \left[r_t + \gamma \max_{a'} Q(s_{t+1},a') - Q(s_t,a_t) \right]. \tag{3.8}$$

where α is the learning rate, γ is the discount factor, and r_t is computed from Eq. (3.7). Exploration is balanced with exploitation using an ε-greedy strategy [93], allowing the agent to remain

adaptive under dynamic conditions.

Algorithmic interpretation and decision flow. At each decision step, the fog node observes its current state s and receives an assigned request $(u_{k,1},\ldots,u_{k,m},d_k^{\max},L_k)$. If it chooses *Serve Locally*, it reserves the requested resources for L_k seconds and the request contributes to local queueing/backlog. If it chooses *Offload to Cloud*, the request is forwarded to the cloud. The reward in Eq. (3.7) encodes both residual resource headroom and delay feasibility, enabling the learned policy to trade off fog utilization against QoS adherence.

3.3.4 RL-Algorithm

Algorithm 4 Q-Learning at a Fog Node (Local Execute vs. Cloud Offload)

Initialize: learning rate $\alpha \in (0,1]$, discount factor $\gamma \in [0,1]$, exploration rate $\varepsilon \in [0,1]$
Initialize: $Q(s,a)$ arbitrarily for all (s,a)
for episode $= 1$ to $Episodes$ **do**
    Reset environment; observe initial state s
    for each assigned request k in the episode **do**
        With probability ε, choose a random action a; otherwise choose $a = \arg\max_{a'} Q(s,a')$
        Estimate $\widehat{D}_k(\text{Fog})$ and $\widehat{D}_k(\text{Cloud})$ using Eq. (3.3)
        Enforce feasibility gates:
        if $a = \text{Fog}$ **and** $(\forall r:\ s_r \geq u_{k,r})$ **and** $\widehat{D}_k(\text{Fog}) \leq d_k^{\max}$ **then**
            Execute locally; reserve resources for L_k
        else if $a = \text{Cloud}$ **and** $\widehat{D}_k(\text{Cloud}) \leq d_k^{\max}$ **then**
            Offload to cloud
        else
            If chosen action is infeasible, execute the feasible alternative if available; otherwise drop request k
        end if
        Observe reward r via Eq. (3.7) and next state s'
        $Q(s,a) \leftarrow Q(s,a) + \alpha\,[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$
        $s \leftarrow s'$
    end for
end for=0

Stopping (convergence) criterion. Convergence is detected when learning stabilizes, i.e., when additional episodes yield only marginal improvement. Operationally, convergence can be declared when the running average of episode returns changes by less than a tolerance over a fixed window, or when $\max_{s,a} |Q_{t+1}(s,a) - Q_t(s,a)|$ remains below a threshold for several consecutive episodes.

This criterion aligns with the plateau behavior commonly observed in convergence plots and yields a reproducible stopping rule.

3.3.5 Task Selection Strategy

Before the RL agent at a fog node makes an offloading decision (Fog vs. Cloud), the system must first decide *which fog node* should receive each queued request. This dispatching step is executed at the distributor node and constructs a feasible candidate set of fog nodes for every request based on (i) resource sufficiency across all resource dimensions and (ii) the end-to-end delay that the request would experience if served by each candidate fog node.

Algorithm 5 describes this fog node selection procedure. Requests are processed in a first-in–first-out manner from the distributor queue Q. For a request k with demand vector u, the distributor scans all fog nodes and includes a node i in the candidate list *FogNodeList* only if it has sufficient remaining capacity in every resource type, i.e., $R'_{i,r} \geq u_r$ for all $r \in \{1,\ldots,m\}$. For each feasible node, the distributor estimates $\widehat{D}_{i,k}$ using Eq. (3.3) (queueing plus propagation delay).

If no feasible fog node exists ($FogNodeList = \emptyset$), the cloud is evaluated as a fallback by estimating $\widehat{D}_{\text{Cloud},k}$. If the cloud delay is within the request's delay tolerance, the request is offloaded to the cloud; otherwise, it is requeued to be reconsidered when system conditions improve. If at least one feasible fog node exists, the distributor selects a destination fog node using one of two heuristics: **Selection-1**, which chooses the node with maximum potential score (Eq. (3.2)), or **Selection-2**, which selects the node with minimum predicted delay $\widehat{D}_{i,k}$. The selected fog node then receives request k, after which the fog-side RL agent is invoked to decide local execution versus cloud offloading.

Algorithm 5 Fog Node Selection Algorithm (Distributor-Side Dispatching)

Initialize: Request Queue Q, resource-demand vector u, n fog nodes, m resource types
while $|Q| \neq 0$ **do**
  Retrieve next request k; extract demands into u
  $FogNodeList \leftarrow \emptyset$
  for $i = 1$ to n **do**
    $canAllocate \leftarrow$ true
    for $r = 1$ to m **do**
      if $R'_{i,r} < u_r$ **then**
        $canAllocate \leftarrow$ false; **break**
      end if
    end for
    if $canAllocate$ **then**
      Compute $\widehat{D}_{i,k}$ using Eq. (3.3)
      if $\widehat{D}_{i,k} \leq d_k^{\max}$ **then**
        Add node i to $FogNodeList$
      end if
    end if
  end for
  if $FogNodeList = \emptyset$ **then**
    Compute $\widehat{D}_{\text{Cloud},k}$ via Eq. (3.3)
    if $\widehat{D}_{\text{Cloud},k} \leq d_k^{\max}$ **then**
      Route request k to Cloud
    else
      Requeue request k
    end if
  else
    Apply **Selection-1:**
    $i_{\text{best}} \leftarrow \arg\max_{i \in FogNodeList}\ \text{potential}_i$
    Or
    Apply **Selection-2:**
    $i_{\text{best}} \leftarrow \arg\min_{i \in FogNodeList}\ \widehat{D}_{i,k}$
    Assign request k to fog node i_{best}
  end if
end while=0

Algorithm 5 operationalizes the distributor-side filtering and ranking stage that precedes learning at the fog node. For each dequeued request, the distributor identifies a feasible candidate set by checking resource sufficiency, then estimates $\widehat{D}_{i,k}$ for each feasible candidate. This delay estimate is used both as a feasibility gate against $d_k^{\max}$ and as a ranking signal for Selection-2. In parallel, Selection-1 uses the potential score as a low-overhead multi-factor ranking signal based on residual

resources, historical selection frequency, and observed delay statistics. Together, these heuristics provide a deployable front-end routing policy that reduces unnecessary cloud routing and mitigates hotspot formation before the fog-side RL policy makes its local execute/offload decision.

3.4 Analysis and Results

To assess the efficacy of the proposed scheduler, we emulated task dispatch in a multi-Fog/Cloud setting. Workloads contained $|\mathscr{R}| \in \{100, 200, \ldots, 1000\}$ requests; each request carried a service time $L_k \in [500, 3000]$ seconds (determining how long resources remain held) and a delay budget $d_k^{\max}$ that feeds feasibility checks and learning signals. To reflect realistic QoS targets, $d_k^{\max}$ was drawn from application-driven classes—5–20 ms (control/URLLC), 20–100 ms (interactive AR/VR, teleoperation, gaming), 100–400 ms (soft real-time analytics), 0.4–1.5 s (lenient interactive), and 1.5–5 s (delay-tolerant)—using a mixture with probabilities $0.15/0.25/0.35/0.15/0.10$ and uniform draws within each bucket. For a compact stress-test alternative, we also used the two-component sampler in Eq. (3.9). The testbed comprised ten fog nodes and one cloud node, with five resource dimensions per fog node. All metrics are averaged over 20 independent trials; increasing the trial count beyond 15 yielded negligible changes in aggregate results. Unless stated otherwise, reward weights were fixed at $\zeta = \theta = 0.5$ to enable fair comparisons across policies.

Evaluation metrics and what is measured. Completion-time and delay metrics follow Eq. (3.3): they are driven by (i) waiting time induced by queueing/backlog and (ii) propagation delay between the request source (or distributor) and the selected execution node. Service time L_k influences queue buildup because resources remain held for L_k seconds. Cloud offloading is reported as the number (or average number) of requests executed in the cloud under each strategy. Load balancing is quantified by the standard deviation of utilization across fog nodes: lower standard deviation implies more even distribution and reduced hotspot formation.

Parameter tuning and fixed weights. Hyperparameters such as ε, α, and γ influence learning speed and stability, while reward weights ζ and θ control emphasis between resource-efficiency and delay sensitivity. In this chapter, $\zeta = \theta = 0.5$ is fixed to keep strategies comparable under the same objective emphasis; exploring different weightings defines different operating points and is naturally treated as a separate sensitivity/Pareto study.

$$d_k^{\max} \sim \begin{cases} \text{Uniform}(20, 800)\ \text{ms}, & 1 - p_{\text{tail}}, \\ \text{Uniform}(2, 5)\ \text{s}, & p_{\text{tail}}, \end{cases} \tag{3.9}$$

with $p_{\text{tail}} = 0.05$ unless stated otherwise.

Our experimental methodology consisted of two phases:

- In the **first phase**, we applied priority-based task scheduling methods (Selection-1 and Selection-2) and benchmarked them against Round Robin (RR) and Random strategies.
- In the **second phase**, we applied reinforcement learning (RL) at fog nodes using Algorithm 4, and also evaluated genetic algorithm (GA)-based variants for comparison under the same workload settings.
- In the GA implementation, request allocations were encoded and evaluated using a fitness proxy aligned with Eq. (3.2). Selection favored higher fitness, with crossover and mutation used to maintain diversity.

Task Completion Time Analysis

Figure 3.3 examines the average task completion time for various scheduling strategies as task volume increases. As the number of tasks rises, response time increases across all approaches; however, the growth rate differs significantly. Random and RR strategies—whether paired with RL or GA—exhibit steep latency growth, indicating limited scalability. In contrast, Selection-1 and Selection-2 provide substantially lower completion times, with Selection-2 yielding the lowest curve by prioritizing nodes with minimal estimated delay. GA-based implementations of Selection-1 and Selection-2 do not show a consistent advantage, highlighting the benefit of RL's online adaptation under dynamic multi-fog conditions.

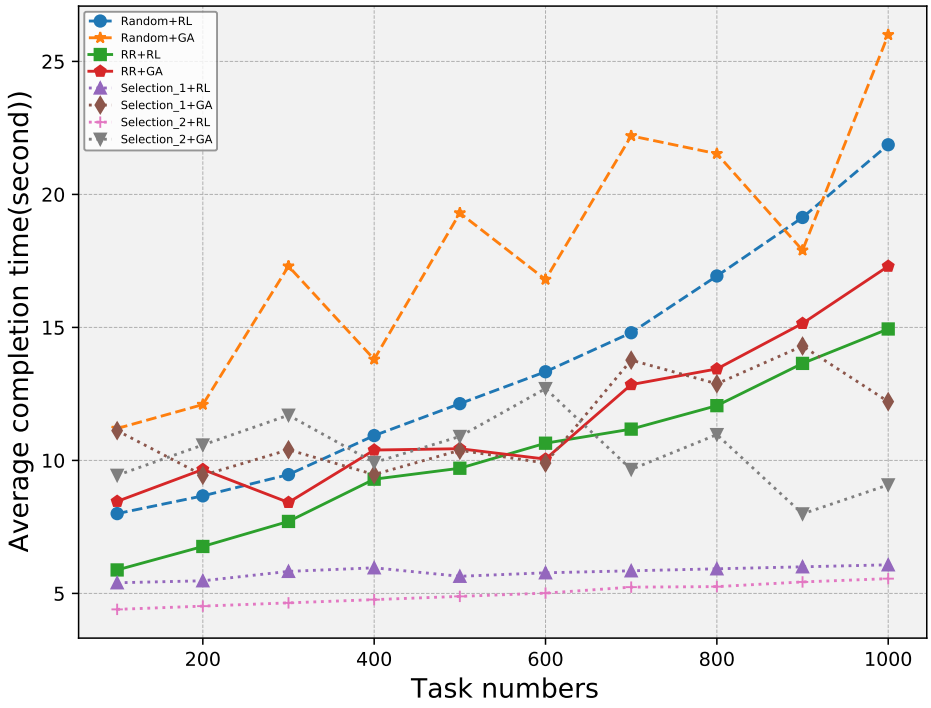

Figure 3.3: Average task completion time

Table 3.1: Insights from Fig. 3.3 (Task Completion Time)

Strategy	**Observation**	**Insight**
Random / RR (RL or GA)	Steep upward trend	Poor scalability and sharp latency increase with task growth.
Selection-1 (RL)	Moderate increase	Improves over baseline methods but not as optimal as Selection-2.
Selection-2 (RL)	Lowest curve	Achieves minimum delay due to delay-aware routing.
GA-based Selection-1/2	Similar to baseline	Limited adaptability under dynamic load.

Cloud Offloading Evaluation

Figure 3.4 illustrates the average number of tasks executed by the cloud as the request load grows. Under Random and RR, cloud usage rises with load because routing is less sensitive to feasibility and

congestion. In contrast, Selection-1 and Selection-2 reduce cloud dependence by better exploiting feasible fog capacity through informed routing. At low task volumes, Selection-2 is often strongest due to direct delay minimization; at higher load, Selection-1 can become more effective by implicitly balancing across nodes via the potential score, reducing hotspots and preserving feasible fog headroom.

Interpreting the downward cloud-served trend under higher load. A decreasing cloud-served curve can appear counter-intuitive if the cloud is treated as an "always-accept" sink. In the proposed pipeline, cloud execution is itself filtered by the request delay tolerance $d_k^{\max}$ and by feasibility gating: requests are not automatically pushed to the cloud when the system becomes busy. Under Selection-1/Selection-2, better fog-side placement reduces unnecessary cloud routing by exploiting remaining feasible fog capacity and avoiding avoidable queue-induced delay violations. Moreover, when neither fog nor cloud can satisfy a request within $d_k^{\max}$, the request may be deferred (requeued) at the distributor and therefore is not counted as "served by cloud" within a fixed measurement horizon. Under heavier load, the fraction of requests failing feasibility at both tiers can increase, reducing the number that actually execute in the cloud during the observed window. Consequently, a downward trend is consistent with a regime where (i) informed fog placement absorbs a larger share of admissible work, while (ii) strict $d_k^{\max}$ gating prevents the cloud from being used as a default sink for requests that would violate QoS.

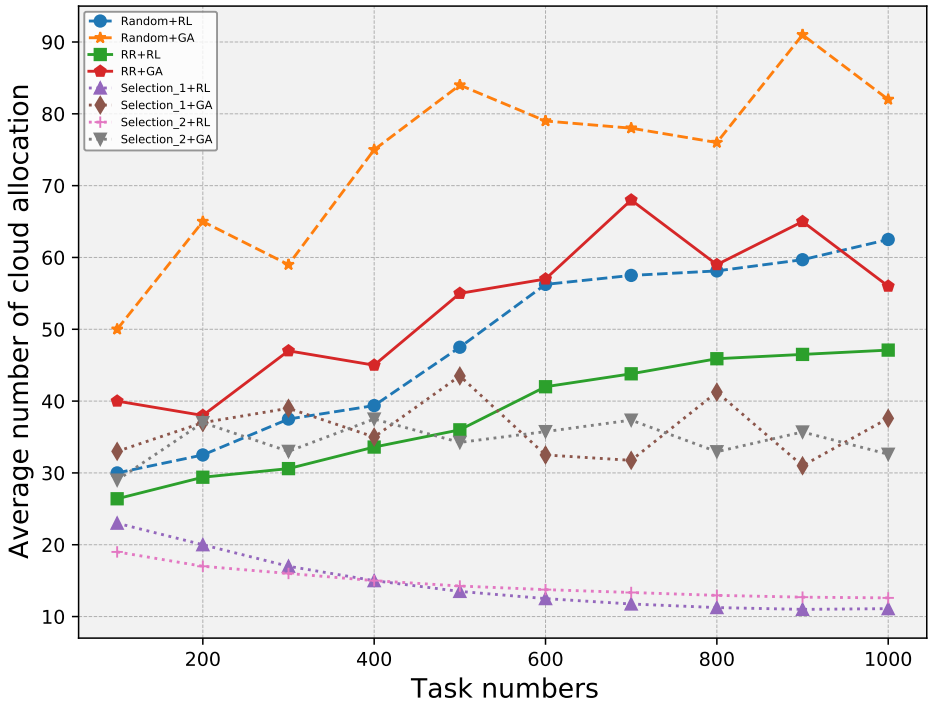

Figure 3.4: Average number of tasks served by the Cloud

Table 3.2: Insights from Fig. 3.4 (Cloud Offloading)

Strategy	**Observation**	**Insight**
Random (RL)	High offload	Most inefficient; rapidly increases cloud usage with load.
Round Robin (RL)	Moderate slope	Slight improvement over Random, but still weak at scale.
Selection-1 (RL)	Decreasing trend	Strong fog utilization; offloading drops under informed routing and gating.
Selection-2 (RL)	Best at low load	Strong initial performance; may be overtaken by Selection-1 at scale.
GA-based methods	Flat/limited change	Limited responsiveness in dynamic settings.

Fog Node Load Balancing

Figure 3.5 presents the standard deviation of resource utilization across fog nodes. Lower deviation implies better load balancing. RL-based Selection-1 yields the most consistent distribution, maintaining the lowest standard deviation across task volumes due to the fairness- and headroom-aware potential score. Selection-2 improves balancing relative to baselines but focuses primarily on minimizing delay, which can concentrate load on a subset of low-delay nodes when many nodes are feasible. Random and RR perform worst, and GA-based approaches show weaker adaptation under dynamic conditions.

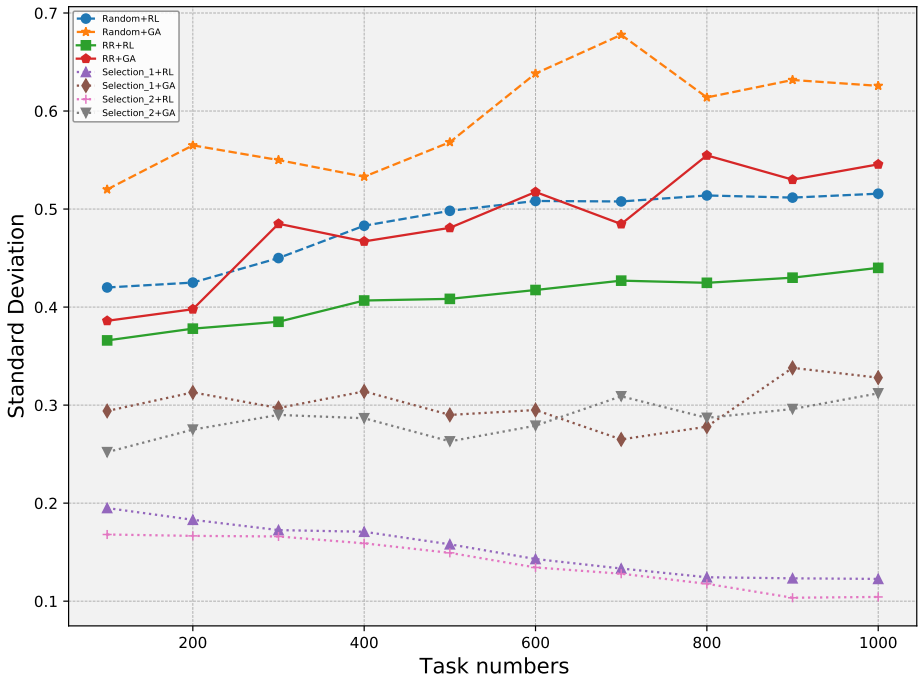

Figure 3.5: Standard deviation of Fog nodes' utilization

Table 3.3: Insights from Fig. 3.5 (Fog Load Balancing)

Strategy	**Observation**	**Insight**
Random / RR	High deviation	Uneven distribution; hotspot formation is likely.
Selection-1 (RL)	Lowest deviation	Most balanced; fairness-aware routing reduces hotspots.
Selection-2 (RL)	Moderate	Better than baselines, but less uniform than Selection-1.
GA-based methods	Unstable / higher variance	Limited ability to adapt to dynamic multi-fog congestion.

Table 3.4: Insights Summary from Multi-Fog Simulation Results

Metric	**Best Performer**	**Insight**
Average Task Time	Selection-2 (RL)	Lowest latency due to adaptive delay-aware allocation.
Cloud Offloading Rate	Selection-1 (RL)	Most effective in limiting cloud dependency as task load grows.
Load Balancing Index	Selection-1 (RL)	Most even resource usage; lowest standard deviation across fog nodes.
Scalability under Load	Selection-2 (RL)	Maintains low delay growth across increasing task volumes.
Algorithm Robustness	RL-based Strategies	Best adaptation to load dynamics; GA shows limited flexibility.
Performance Consistency	Selection-1 (RL)	Strong balance between feasibility, fairness, and delay sensitivity.

3.5 Summary

This chapter presented a comprehensive reinforcement learning (RL)-based scheduling framework for resource allocation in a multi-fog/cloud computing environment. The system architecture follows a two-stage decision pipeline: a distributor node performs feasibility-aware fog-node routing (via

Selection-1 / Selection-2) under resource and delay constraints, and the selected fog node then applies decentralized learning where each fog node acts as an autonomous agent deciding whether to serve requests locally or offload them to the cloud. The problem was modeled as a distributed Markov Decision Process (D-MDP), where queueing dynamics, propagation delay, multi-resource feasibility, and service-time coupling (resource holding for L_k seconds) jointly shape both routing feasibility and learning behavior. The scheduling policy was trained using a Q-learning algorithm guided by a reward function that balances delay minimization and resource efficiency (and, when enabled, discourages unnecessary cloud offloading).

To validate the efficacy of the approach, we conducted extensive simulation-based evaluations across two experimental phases. In the first phase, we examined heuristic-driven strategies—Random, Round Robin (RR), Selection-1, and Selection-2—under both RL and GA-based scheduling policies. In the second phase, we applied standalone RL and GA optimizers to evaluate their convergence behavior, adaptability, and overall effectiveness.

Experimental results showed that RL combined with intelligent fog-node selection heuristics consistently outperformed traditional baselines. Specifically, the following conclusions were drawn:

- **Task Completion Time:** As shown in Fig. 3.3, RL with Selection-2 strategy delivered the lowest average task completion times across all task volumes, reflecting its effectiveness in minimizing latency by favoring feasible nodes with the shortest predicted end-to-end delay. Selection-1 also performed well but slightly lagged behind Selection-2 in terms of responsiveness.

- **Cloud Offloading:** Fig. 3.4 revealed that RL with Selection-1 reduced the number of tasks offloaded to the cloud more effectively than other strategies, especially under high workloads. This indicates better utilization of local fog resources and improved fog–cloud balance. The observed decreasing (or non-increasing) cloud-served trends under heavier load are consistent with feasibility gating under strict delay tolerances (i.e., only requests meeting $d_k^{\max}$ are executed), coupled with improved fog-side absorption due to informed placement. In contrast, GA-based scheduling showed minimal advantage and failed to adapt cloud offloading behavior dynamically.

- **Load Balancing:** As depicted in Fig. 3.5, RL-based Selection-1 provided the most balanced load distribution among fog nodes, achieving the lowest standard deviation in node utilization. This reflects a stable and fair task distribution mechanism, critical in preventing resource bottlenecks and maintaining overall system performance.

- **Overall Strategy Comparison:** From the insights tables, it became evident that while Selection-2 (RL) excels in minimizing delay, Selection-1 (RL) offers superior performance in reducing cloud dependency and achieving load uniformity. Random and RR strategies, whether paired

with RL or GA, showed poor scalability and failed to maintain efficiency under increasing load. GA-based approaches remained comparatively less responsive under dynamic multi-fog conditions due to slower convergence and reliance on stochastic search, which is less aligned with rapidly changing queueing and resource states.

In summary, the proposed RL framework—when coupled with feasibility-aware fog node selection heuristics like Selection-1 and Selection-2—demonstrates clear advantages in managing computational resources within dynamic, distributed fog/cloud systems. Its decentralized design offers resilience, scalability, and improved QoS under variable workloads. These findings motivate the next step addressed in Chapter 4, where a hybrid Genetic Algorithm–Reinforcement Learning approach is introduced to improve exploration, accelerate convergence, and further optimize multi-fog/cloud resource allocation under heavier and more heterogeneous workloads.

Chapter 4

Multi-Fog/Cloud Resource Allocation Using a Hybrid Learning Strategy Based on Genetic Algorithms and Reinforcement Learning

4.1 Introduction

As fog computing systems grow in scale and complexity, traditional task allocation mechanisms struggle to efficiently manage resources in real-time, latency-sensitive environments. The heterogeneity of fog nodes, coupled with fluctuating workloads and strict user delay constraints, necessitates adaptive and intelligent scheduling strategies that go beyond static heuristics.

This chapter presents a hybrid resource allocation framework that integrates Reinforcement Learning (RL) with Genetic Algorithms (GA) to optimize task scheduling in multi-fog/cloud environments. The proposed method leverages the strengths of both paradigms: RL provides adaptive, feedback-driven decision-making, while GA introduces structured exploration through evolutionary operators, thereby improving learning convergence and global policy quality.

The problem setting involves multiple fog nodes coordinated by a central distributor that receives user-submitted task requests. Each request specifies its resource demands, maximum acceptable delay, and service duration. The distributor maintains real-time status information about each fog node and is responsible for assigning requests either to local fog nodes or offloading them to the cloud. A hybrid reward function is used to balance delay sensitivity and resource efficiency, allowing the system to flexibly adjust to changing network states.

Two fog node selection heuristics—Selection-1 and Selection-2—are used during initial alloca-

tion: the former employs a potential-based metric considering resource capacity and usage frequency (adjusted by delay), while the latter selects (among feasible candidates) the node with minimal predicted end-to-end delay. Once a fog node is selected, the hybrid GA–RL model determines whether to process the request locally or offload it to the cloud, based on learned Q-values and GA-guided exploration during the exploration phase.

Simulation-based evaluation is conducted across a variety of workload scenarios using synthetically generated requests. The framework's performance is benchmarked against conventional scheduling strategies, including Random and Round Robin, as well as standalone RL and GA implementations. Key performance metrics include average task completion time, cloud offloading ratio, load balancing (measured by standard deviation), and convergence trends.

Results demonstrate that the hybrid GA–RL approach significantly improves scheduling efficiency, reduces average task latency, and ensures better fog resource utilization. The algorithm exhibits robust convergence properties and adapts well under high-load conditions, highlighting its potential for scalable and intelligent fog-cloud orchestration.

The remainder of this chapter is organized as follows: Section 4.2 formulates the resource allocation problem and introduces the system architecture, focusing on task scheduling and resource management challenges. Section 4.3 details the scheduling mechanism, including the Selection-1 and Selection-2 heuristics, and their role in optimizing resource distribution in the fog-cloud system. Subsection 4.3.1 explains the initial allocation strategy, while Subsection 4.3.2 introduces the hybrid Reinforcement Learning and Genetic Algorithm refinement.

Section 4.4 reports a simulation-based comparison of the proposed hybrid GA–RL scheduler against representative traditional strategies, emphasizing the core metrics used throughout this chapter, including average task completion time and cloud offloading rate (and, where relevant, fog-side load balancing and convergence behaviour). Section 4.5 synthesizes the experimental findings into concise insights and outlines practical extensions—notably multi-agent and federated learning variants—to further enhance scalability and adaptability in larger, more heterogeneous deployments.

Finally, Section 4.6 positions our Hybrid Genetic Algorithm–Reinforcement Learning framework [99] against the reviewed state-of-the-art using the same evaluation lens, clarifying where GA-guided Q-learning is empirically and architecturally advantageous and where alternative families may dominate due to different priorities (e.g., privacy, security, mobility, or high-dimensional decision spaces).

4.2 Problem Statement

The increasing adoption of fog computing for latency-sensitive applications has introduced new challenges in dynamic task scheduling and resource management. In such environments, users submit computational requests characterized by diverse resource demands and strict delay constraints. Effi-

cient allocation of these tasks to fog or cloud resources is critical to maintaining system responsiveness, load balance, and user satisfaction.

In this framework, the resource allocation problem is defined over a multi-fog/cloud system consisting of n fog nodes, denoted $F_1, F_2, \ldots, F_n$, each equipped with a heterogeneous set of m resource types, such as CPU, memory, and storage. The maximum capacity for each resource type is predefined, and the available resources fluctuate dynamically as tasks are executed and completed. In addition to these fog nodes, a centralized cloud server is available as a fallback, albeit with higher transmission delays.

User requests arrive in the form of tuples:

$$(u_1, u_2, \ldots, u_m, d_{\max}, L). \tag{4.1}$$

where u_1 to u_m denote the quantity of resources required, $d_{\max}$ specifies the maximum allowable delay, and L indicates the required service duration. Each request is treated as indivisible and must be assigned to either a fog node or the cloud, subject to resource availability and delay tolerance.

Delay tolerance $d_{\max}$ and service time L. The delay tolerance $d_{\max}$ is an application-level QoS constraint that upper-bounds the end-to-end delay a request can tolerate. The service time L is the holding time of resources after admission: once a request is scheduled on a fog node, the corresponding resources remain reserved for L seconds, and are released after completion. This mechanism creates temporal coupling between decisions and is central to the observed queueing dynamics under higher loads.

Relationship between L and execution delay. The parameter L controls *resource reservation duration*: once admitted, the requested resources remain occupied for L seconds before being released. In the delay model, the execution component can be set to $T^{\text{exe}}_{i,j} = L$ under a constant-service abstraction, or derived from request size and node capability. Regardless of parametrization, L is the mechanism that induces temporal coupling in residual capacity and queueing.

The challenge lies in determining the most suitable execution location for each request. When possible, fog nodes are preferred due to their proximity and lower latency. However, if no fog node can satisfy the request—either due to insufficient resources or because the predicted end-to-end delay would exceed the request tolerance—the request must be offloaded to the cloud (if cloud QoS is feasible) or deferred.

The problem is further complicated by the need to balance workload across fog nodes, avoid bottlenecks, and minimize reliance on cloud infrastructure. This requires a decision-making framework capable of capturing real-time system dynamics, forecasting node availability, and intelligently trading off delay minimization and resource efficiency.

Cloud execution model (assumption used in this chapter). The cloud tier is modeled as an *elastic* compute backend with negligible queueing delay relative to network latency. Accordingly, we set $W_{\text{Cloud},j} \approx 0$ and treat the cloud-side end-to-end delay as $\widehat{D}_{\text{Cloud},j} = T^{\text{prop}}_{\text{Cloud},j} + T^{\text{exe}}_{\text{Cloud},j}$. This abstraction matches the common "capacity-on-demand" assumption for centralized clouds and isolates the impact of fog-side congestion and routing/learning decisions.

Motivation for lightweight fog-node selection in the distributor (Selection-1/Selection-2). The first decision the system must make for every arriving request is not yet an "offload-to-cloud" choice; it is a routing choice: *which fog node should receive the request* (if any fog node is feasible). This front-end decision is executed at the distributor for every arrival and must therefore remain computationally inexpensive, stable under bursty traffic, and transparent enough to debug and validate. For this reason, the distributor relies on two simple but strongly objective-aligned heuristics. Selection-1 ranks feasible fog nodes using a potential score that jointly reflects multi-resource headroom, historical over-selection, and delay conditions, thereby discouraging persistent hotspots and promoting broader capacity utilization. Selection-2 refines the choice by explicitly selecting the feasible candidate with the smallest predicted end-to-end delay, which directly targets latency-sensitive placement when multiple nodes can satisfy the resource requirements.

These heuristics are deliberately not intended to replace learning; they provide a deterministic feasibility-and-prioritization layer that keeps the system deployable in real time and prevents the learning component from spending capacity on trivial eliminations (e.g., obviously infeasible nodes). In particular, they (i) prune the candidate set to nodes that satisfy resource constraints and are plausible under $d_{\max}$, (ii) provide a robust routing baseline during cold start (before Q-values have stabilized), and (iii) reduce the downstream decision complexity by ensuring that the learning stage operates on a request that has already passed a basic feasibility screen. After this first-phase routing, the second phase applies learning-based adaptation at the selected fog node: an RL agent (and, in the hybrid extension, GA-guided exploration) decides whether local execution remains beneficial or whether cloud execution is preferable under the current congestion and delay context. The resulting two-phase structure therefore combines short-term responsiveness and low overhead at the distributor with long-term policy improvement at the fog layer, enabling high throughput and fairer utilization while adapting to dynamic workloads and evolving fog resource states.

4.3 Hybrid Scheduling and Algorithmic Integration

This section presents the hybrid scheduling mechanism that integrates Reinforcement Learning (RL) and Genetic Algorithm (GA) with initial task allocation strategies to optimize request distribution in a Fog–Cloud system. The full process comprises two phases. In the first phase, task requests are allocated using heuristic scheduling strategies (Selection-1 and Selection-2), and in the second

phase, adaptive learning techniques (RL and GA) refine these allocations dynamically. Together, this combination balances fairness, load distribution, resource utilization, and delay minimization.

4.3.1 Phase 1: Fog Node Selection Algorithm

The initial allocation strategy evaluates all incoming requests based on the capacity of available Fog nodes and offloads to the Cloud only when necessary. Each request is queued and matched to suitable nodes by checking resource constraints and predicted end-to-end delay bounds.

Delay-Based Selection Metrics As Introduced in Chapter 3

Two metrics govern the node selection, recalled below for clarity:

$$\text{potential}_i = \left(\frac{\prod_{k=1}^{m} R'_{i,k}}{P_i \times \text{AvgDelay}_i} \right) \tag{4.2}$$

where $R'_{i,k}$ is the remaining amount of resource k on node i, and P_i is the number of times node i has been previously selected.

Definition of AvgDelay$_i$ and P_i. AvgDelay$_i$ denotes a running estimate of the mean end-to-end delay observed on node i (e.g., an exponential moving average over admitted requests), and P_i is the cumulative selection count used to penalize persistent over-routing. After each assignment to node i, we update $P_i \leftarrow P_i + 1$ and update AvgDelay$_i$ using the newly observed completion delay of the admitted request.

$$D_{i,j} \;=\; \widehat{D}_{i,j} \;=\; W_{i,j} \;+\; T_{i,j}^{\text{prop}} \;+\; T_{i,j}^{\text{exe}} \tag{4.3}$$

where $W_{i,j}$ is the predicted waiting (queueing) time at node i, $T_{i,j}^{\text{prop}}$ is propagation/network delay, and $T_{i,j}^{\text{exe}}$ is execution/service time. For Cloud routing we use $\widehat{D}_{\text{Cloud},j} = T_{\text{Cloud},j}^{\text{prop}} + T_{\text{Cloud},j}^{\text{exe}}$ with $W_{\text{Cloud},j} \approx 0$ under the elastic-cloud assumption.

Algorithm: Fog Node Selection

Algorithm 6 Fog Node Selection Algorithm (with bounded requeue)

Initialize: Request queue Q (FIFO), n fog nodes, m resource types
Given: Remaining capacities $R'_{i,k}$ for each fog node $i \in \{1,\dots,n\}$ and resource $k \in \{1,\dots,m\}$
Given: Requeue counter q_j per request j (initialize to 0 at arrival) and maximum retries $q_{\max}$
while $|Q| \neq 0$ **do**
  Dequeue next request j
  Extract resource demands $\mathbf{u}^{(j)} = \{u_k^{(j)}\}_{k=1}^{m}$ and delay tolerance $d_{\max}^{(j)}$
  $FogNodeList \leftarrow \emptyset$
  for $i = 1$ to n **do**
    $canAllocate \leftarrow$ **true**
    for $k = 1$ to m **do**
      if $R'_{i,k} < u_k^{(j)}$ **then**
        $canAllocate \leftarrow$ **false**; **break**
      end if
    end for
    if $canAllocate$ **then**
      Compute predicted delay $\widehat{D}_{i,j}$ using Eq. (4.3)
      if $\widehat{D}_{i,j} \leq d_{\max}^{(j)}$ **then**
        Add node i to $FogNodeList$
      end if
    end if
  end for
  if $FogNodeList = \emptyset$ **then**
    Compute $\widehat{D}_{\text{Cloud},j} = T^{\text{prop}}_{\text{Cloud},j} + T^{\text{exe}}_{\text{Cloud},j}$ {Elastic cloud: $W_{\text{Cloud},j} \approx 0$}
    if $\widehat{D}_{\text{Cloud},j} \leq d_{\max}^{(j)}$ **then**
      Route request j to Cloud
    else
      $q_j \leftarrow q_j + 1$
      if $q_j < q_{\max}$ **then**
        Requeue request j
      else
        Drop request j {QoS cannot be met after $q_{\max}$ retries}
      end if
    end if
  else
    Compute potential$_i$ for all $i \in FogNodeList$ using Eq. (4.2)
    Select one policy:
    $i_{\text{best}} \leftarrow \arg\max_{i \in FogNodeList}$ potential$_i$ {Selection-1}
      or
    $i_{\text{best}} \leftarrow \arg\min_{i \in FogNodeList} \widehat{D}_{i,j}$ {Selection-2}
    Assign request j to fog node i_{best}
  end if
end while=0

Algorithm. 6 executes for every request arriving at the distributor node and ensures that resource constraints and predicted end-to-end delays are jointly considered before routing tasks. The Fog Node Selection Algorithm processes requests in a first-in–first-out manner. For each request j dequeued from Q, it constructs a candidate set of fog nodes that can feasibly serve the request given their remaining capacities and delay tolerance. Concretely, node i is resource-feasible only if $R'_{i,k} \geq u_k^{(j)}$ for all $k \in \{1,\ldots,m\}$. For each resource-feasible node, the algorithm computes $\widehat{D}_{i,j}$ (Eq. (4.3)) and retains i as a candidate only if $\widehat{D}_{i,j} \leq d_{\max}^{(j)}$. If no fog node is admissible, the cloud is evaluated under the elastic-cloud model (with $W_{\text{Cloud},j} \approx 0$) and is used only if $\widehat{D}_{\text{Cloud},j} \leq d_{\max}^{(j)}$.

To prevent a subtle but important *infinite requeue* failure mode, we maintain a per-request retry counter q_j that increments each time the request is deferred. If $q_j \geq q_{\max}$, the request is dropped and counted as a QoS failure, ensuring the simulator terminates and that performance metrics are not biased by unbounded deferrals.

Complexity and properties: Each request incurs a feasibility check across n nodes and m resource types, plus a constant-time delay computation per resource-feasible node, yielding $O(nm)$ work per request (and $O(|Q|nm)$ overall). The algorithm enforces resource feasibility and QoS feasibility by construction, and provides a clear policy hook (Selection-1 or Selection-2) that can be used directly as a baseline or combined with learning in later phases. It also supports bounded backpressure through requeueing, while guaranteeing termination by dropping requests that exceed $q_{\max}$ retries.

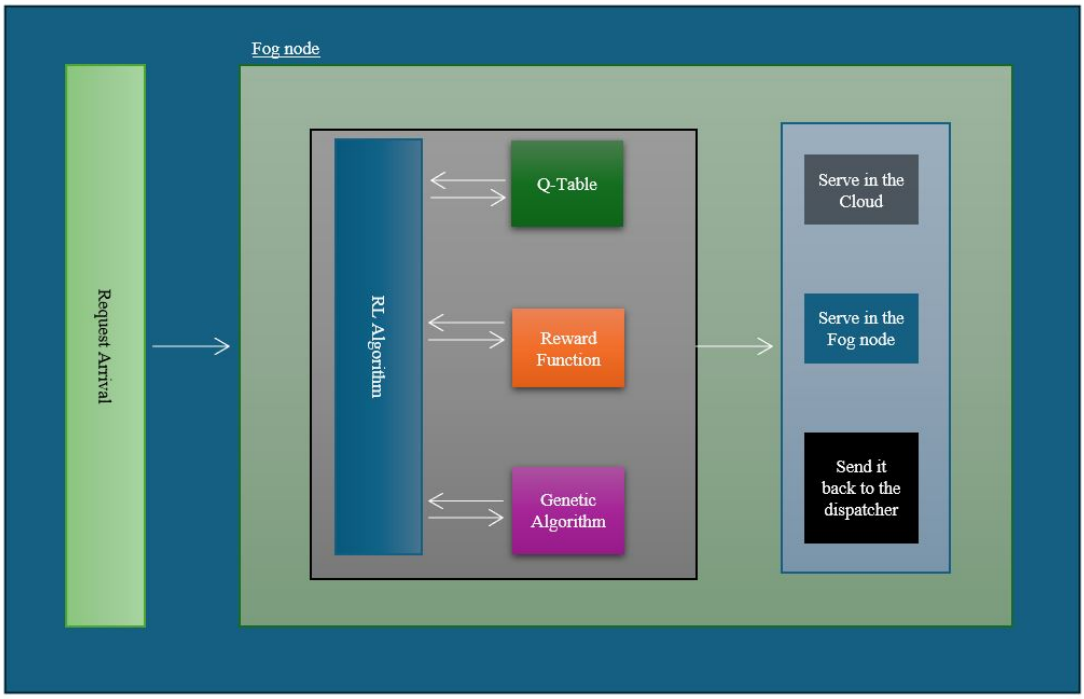

Figure 4.1: RL/GA resource allocation and scheduling in individual Fog nodes.

4.3.2 Phase 2: Hybrid Reinforcement Learning and GA-Based Refinement

After initial allocation, each Fog node acts as an RL agent, as illustrated in Fig. 4.1, further deciding whether to process the request locally or offload it to the Cloud. To boost the exploration capabilities of the RL agent, a Genetic Algorithm is integrated into the exploration phase. This ensures broader action search, faster convergence, and avoidance of poor local optima.

Hybrid RL/GA Algorithm

Algorithm 7 Hybrid (RL/GA) Algorithm

Initialize $\gamma \in [0,1]$, $\varepsilon \in [0,1]$, and learning rate $\alpha \in (0,1]$
Input population size P, max generations G, number of episodes $Episodes$, action set $\mathscr{A}$
Initialize Q-table: $Q(s,a)\ \forall (s,a)$
for $ep = 0$ to $Episodes - 1$ **do**
    Sample $e \sim \text{Uniform}(0,1)$; read request/context and observe current state s_t
    if $e \leq \varepsilon$ **then**
        Exploration Phase
        Random admissible action (Q-learning) **or** GA-guided proposal:
            Encode candidate decisions as chromosomes; initialize population (P)
            Compute fitness via Section 4.3.4 (Eqs. (4.9)–(4.10))
            Apply selection, crossover, and mutation for G generations
            Decode the fittest chromosome as action a_t
    else
        Exploitation Phase: $a_t \leftarrow \arg\max_{a \in \mathscr{A}} Q(s_t,a)$
    end if
    Execute a_t, observe reward r_t via Eq. (4.5), and next state s_{t+1}
    Update Q-table:

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha[r_t + \gamma \max_{a'} Q(s_{t+1},a') - Q(s_t,a_t)]$$

end for=0

The Hybrid (RL/GA) Algorithm couples ε-greedy Q-learning with a GA-based exploration mechanism. Across $Episodes$ training iterations, the agent observes the current state s_t, samples $e \sim \text{Uniform}(0,1)$, and chooses between exploration and exploitation. Under exploration ($e \leq \varepsilon$), the agent either takes a uniformly random admissible action (standard Q-learning exploration) or invokes a GA subroutine to construct an action via evolutionary search. Under exploitation ($e > \varepsilon$), the agent selects the greedy action $a_t = \arg\max_a Q(s_t,a)$. The chosen action is executed, yielding reward r_t and next state s_{t+1}; the value function is updated via the temporal-difference rule above.

GA encoding alignment with the offloading decision. Section 4.3.4 presents the most general chromosome $\mathbf{x}$ for task-to-node allocation across all fog nodes and the cloud. In Phase 2, the decision is an offloading choice for the selected fog node, hence the GA search can be viewed as operating on the same encoding with a *restricted candidate set* per gene (e.g., $\{i_{\text{best}}, 0\}$ for "selected fog" vs. "cloud"), or on a short local window of queued tasks where each gene selects between local execution and cloud offloading. This keeps the GA formulation consistent with the system-level objective while matching the Phase 2 action semantics.

QoS feasibility gate and deferral policy (Phase 2). An action is admissible only if the predicted end-to-end delay under that action satisfies $\widehat{D}_j(a) \leq d_{\max}^{(j)}$; otherwise, the request is deferred (requeued) rather than executed. This same gate is applied to both fog and cloud actions, keeping the learning signal consistent with the simulator's admission rules.

Reward Function (deadline-consistent). For request k, let $req_{k,i}$ be its demand on resource $i \in \{1,\ldots,m\}$, and let $s_{t_k,i}$ denote the residual amount of resource i at decision time t_k:

$$s_{t_k,i} = C_i - \sum_{j \in \mathscr{A}(t_k)} req_{j,i}, \tag{4.4}$$

where C_i denotes the capacity of resource type i on the *currently considered node* (i.e., the selected fog node's capacity for resource i), and $\mathscr{A}(t_k)$ is the set of active (not-yet-finished) requests whose resource reservations have not expired.

Let $\widehat{D}_k(a)$ be the predicted end-to-end delay under action $a \in \{\text{Fog}, \text{Cloud}\}$ (consistent with Eq. (4.3) and the elastic-cloud assumption), and let $d_{\max}^{(k)}$ be the request delay tolerance. To ensure numerical stability, we use $\rho_{\text{stab}} > 0$.

$$r_k(a) = \zeta \left(\prod_{i=1}^{m} \frac{s_{t_k,i}}{req_{k,i} + \rho_{\text{stab}}} \right)^{\frac{1}{2m}} + \theta \sqrt{\frac{d_{\max}^{(k)} + \rho_{\text{stab}}}{\widehat{D}_k(a) + \rho_{\text{stab}}}} - \rho_{\text{cloud}} \mathbb{I}\{a = \text{Cloud}\}, \qquad \zeta + \theta = 1. \tag{4.5}$$

Both positive terms are dimensionless (ratio-normalized), allowing them to be combined linearly under $\zeta + \theta = 1$ without unit inconsistency.

The reward in Eq. (4.5) explicitly compares the request tolerance $d_{\max}^{(k)}$ against the predicted end-to-end delay $\widehat{D}_k(a)$. The first term aggregates residual-to-demand ratios across all m resource types using a geometric product, encouraging decisions that preserve headroom. The second term is the deadline margin signal: it increases when predicted delay is comfortably within the deadline and decreases as $\widehat{D}_k(a)$ approaches (or exceeds) $d_{\max}^{(k)}$. The cloud penalty term encodes a system-level preference for fog execution when both tiers are feasible.

On delay–resource trade-offs and why a full Pareto study is not mixed into the baseline comparisons. The coefficients ζ and θ determine the operating point between resource-efficiency and delay-sensitivity. In this chapter, these weights are held fixed during the primary comparisons to preserve a consistent objective across all baselines (Random/RR/RL/GA/Hybrid). Studying the full delay–resource trade-off is important, but it constitutes a separate sensitivity/Pareto analysis: varying (ζ, θ) changes the objective itself, and mixing different operating points into the same baseline comparison would confound algorithmic differences with objective re-weighting. Nevertheless, the proposed reward is explicitly designed to support such a trade-off study, and the same simulator can sweep (ζ, θ) to produce a family of curves (delay vs. utilization vs. cloud offloading) as an extension.

4.3.3 Summary of the Hybrid Scheduling Model

- **Initial Allocation:** Conducted via the Fog Node Selection Algorithm based on heuristics and node statistics.
- **Refined Decision-Making:** Performed at the node level using a Q-learning agent with optional GA-driven exploration.
- **Cloud Offloading:** Performed only when cloud QoS is feasible; otherwise requests are deferred.

This model supports adaptive, distributed scheduling under real-time conditions. The next section presents the results of simulations validating the efficiency and performance of this hybrid strategy.

4.3.4 Genetic Algorithm (GA) Configuration for the Proposed Multi-Fog/Cloud Scheduler

While the previous subsection describes the generic GA workflow, the proposed system employs a GA instance that is tailored to the multi-Fog/Cloud task scheduling problem. In particular, the GA is used to search over feasible task-to-node allocation patterns that reduce end-to-end completion time and mitigate overload, while respecting heterogeneous Fog resource constraints and the Cloud fallback option. This subsection details the GA representation, feasibility handling, fitness formulation, genetic operators, termination criteria, and its integration with the RL component.

Solution Encoding (Chromosome Representation)

Let $\mathscr{T} = \{1, \ldots, N\}$ denote the set of tasks in the current scheduling window and $\mathscr{F} = \{1, \ldots, n\}$ the set of Fog nodes (with an additional Cloud option denoted by 0). Each GA individual (chromosome)

encodes a complete allocation plan:

$$\begin{aligned} \mathbf{x} &= [x_1, x_2, \ldots, x_N], \\ x_j &\in \{0, 1, \ldots, n\}. \end{aligned} \tag{4.6}$$

where $x_j = i$ indicates that task j is assigned to Fog node i, and $x_j = 0$ indicates Cloud execution.

Feasibility and Constraint Handling

An assignment is considered feasible if, for each Fog node i, the total allocated demand does not exceed its *available* (residual) capacity across all resource dimensions at the decision time. Denoting the demand of task j on resource type r by $dem_{j,r}$ and the residual capacity of Fog node i by $cap_{i,r}$, feasibility requires:

$$\sum_{j \in \mathscr{T} : x_j = i} dem_{j,r} \leq cap_{i,r}, \quad \forall i \in \mathscr{F}, \ \forall r \in \{1, \ldots, m\}. \tag{4.7}$$

Infeasible chromosomes are handled using a repair-and-penalize strategy. First, a lightweight repair operator iteratively reassigns tasks from overloaded Fog nodes to alternative feasible Fog nodes; if no feasible Fog node exists, the task is redirected to the Cloud ($x_j \leftarrow 0$). If residual infeasibility remains, the chromosome is retained but receives a large penalty in the fitness function, ensuring the GA systematically favors feasible allocations.

Fitness Function Tailored to the System Objective

The GA fitness is designed to reflect the same performance objectives used in Chapter 4 evaluation (completion time, resource feasibility, and Cloud offloading control). For a chromosome $\mathbf{x}$, let $D_{x_j,j}$ denote the end-to-end delay (completion time) of task j under its assigned execution node x_j. The primary objective is to minimize the average completion time:

$$\overline{D}(\mathbf{x}) = \frac{1}{N} \sum_{j=1}^{N} D_{x_j,j}. \tag{4.8}$$

To discourage overload and excessive Cloud usage, the GA additionally incorporates (i) a constraint-violation penalty term $\Phi(\mathbf{x})$ and (ii) a Cloud allocation term $C(\mathbf{x}) = \sum_{j=1}^{N} \mathbb{I}(x_j = 0)$, where $x_j = 0$ denotes Cloud assignment. The final objective (to be minimized) is defined as:

$$\mathscr{J}(\mathbf{x}) = \omega_D \overline{D}(\mathbf{x}) \ + \ \omega_C \frac{C(\mathbf{x})}{N} \ + \ \lambda \, \Phi(\mathbf{x}), \qquad \omega_D + \omega_C = 1, \tag{4.9}$$

where ω_D and ω_C control the GA-only trade-off between average delay and Cloud usage, and λ is a large penalty weight ensuring infeasible allocations are dominated by feasible ones (i.e., $\Phi(\mathbf{x}) = 0$

for feasible chromosomes and $\Phi(\mathbf{x}) > 0$ otherwise). The GA converts this to a maximization fitness via:

$$Fit(\mathbf{x}) = \frac{1}{1 + \mathscr{J}(\mathbf{x})}. \tag{4.10}$$

Here, ω_D and ω_C are GA-only trade-off weights and are distinct from the RL reward weights ζ and θ in Eq. (4.5).

Selection, Crossover, and Mutation (Problem-Specific Operators)

Selection. Parent chromosomes are selected using tournament selection (or roulette-wheel selection) based on $Fit(\mathbf{x})$, which provides robustness against noisy fitness estimates in stochastic simulations.

Crossover. Because the chromosome is a task-indexed allocation vector, crossover is implemented using one-point or two-point crossover over task indices. Given parents $\mathbf{x}^{(1)}$ and $\mathbf{x}^{(2)}$, crossover exchanges contiguous task assignment segments to produce children while preserving partial allocation structures.

Mutation. Mutation is implemented as a reassignment mutation: a small subset of genes is randomly selected, and each selected task j is reassigned to a different feasible Fog node; if no feasible Fog choice exists, it is redirected to the Cloud.

Elitism. The best E chromosomes are copied to the next generation unchanged to prevent loss of high-quality allocations.

GA Hyperparameters and Termination

The GA runs for G generations with population size P. Termination occurs when either (i) G generations are completed or (ii) fitness improvement remains below a small threshold for several consecutive generations (early stopping). Importantly, the GA is invoked with a bounded computational budget so that the hybrid scheduler remains practical.

Integration with the RL Component (Hybrid GA–RL Usage)

In the proposed hybrid design, the GA does not replace RL; rather, it complements RL by providing a global search mechanism that proposes strong candidate decisions under high load and non-convex constraints. Concretely, the GA output is used in one (or both) of the following ways depending on the hybrid variant:

- **GA-guided action proposal:** the GA produces a near-optimal allocation plan $\mathbf{x}^{GA}$ for the current window. RL then either selects among GA-ranked candidates or uses $\mathbf{x}^{GA}$ to bias exploration toward high-quality actions.

Table 4.1: GA configuration used in the proposed scheduler.

Parameter	Description
P	Population size
G	Max generations
E	Elitism count
p_c	Crossover probability
p_m	Mutation probability
Selection	Tournament (size k) / Roulette wheel
Crossover	Uniform / One-point / Two-point on task indices
Mutation	Feasible reassignment (Fog→Fog, else Cloud)
Repair	Overload repair with Cloud fallback

- **Warm-start / policy refinement:** the best GA chromosome is used to initialize (or periodically refine) RL state-action preferences, accelerating convergence and reducing the chance of RL becoming trapped in locally suboptimal scheduling patterns.

This integration is especially beneficial in multi-Fog systems because GA improves global load balancing and feasibility satisfaction, while RL provides fast online adaptation to dynamics (e.g., fluctuating task demands and node availability). The resulting hybrid behavior is consistent with the empirical trends in this chapter, where the hybrid approach maintains lower completion time and reduces undesirable Cloud allocations under increasing task volume.

4.4 Analysis and Results

To evaluate the efficacy of the proposed scheduler, we simulated task allocation in a multi-Fog/Cloud environment. Workloads comprised $|R| \in \{100, 200, \ldots, 1000\}$ requests; each request specified a service duration $L \in [500, 3000]$ seconds (governing how long resources remain reserved) and a delay tolerance $d_{\max}$ that enters the learning signal. To instantiate realistic QoS targets, $d_{\max}$ was primarily sampled from application-inspired buckets—5–20 ms (control/URLLC), 20–100 ms (interactive AR/VR, teleoperation, gaming), 100–400 ms (soft real-time analytics), 0.4–1.5 s (lenient interactive), and 1.5–5 s (delay-tolerant)—using a mixture with class probabilities $0.15/0.25/0.35/0.15/0.10$ and uniform draws within each bucket. As a compact alternative (and for stress testing), we also used the two-component sampler in Eq. (4.11). The testbed comprised ten Fog nodes and one Cloud node; each Fog node exposed five resource types. Reported metrics are means over 20 independent runs; increasing the run count beyond 15 produced negligible changes in the aggregates. Unless stated otherwise, reward weights were fixed at $\zeta = \theta = 0.5$ to ensure fair comparisons across strategies.

The cloud tier follows the elastic model described in Section 4.2 (negligible queueing).

Simulation protocol and metric definitions. Each run proceeds for a fixed simulation horizon T_{sim} (seconds). A request is counted as completed only if it is admitted (passes the QoS gate) and finishes execution before T_{sim}. Requests that are requeued but not admitted within T_{sim} are excluded from completion-time averages.

Completion time for a completed request j is measured from its admission time to completion time, and its expectation is captured by $\widehat{D}_{x,j}$ under the chosen tier $x \in \mathscr{F} \cup \{\text{Cloud}\}$. Cloud served counts the number of requests whose final executed tier is Cloud (i.e., completed in Cloud within the horizon), not merely attempted offloading. Load balancing is measured by the standard deviation of per-node utilization, where utilization is the time-average fraction of reserved capacity (aggregated in the simulator across resources).

$$d_{\max} \sim \begin{cases} \text{Uniform}(20, 800)\ \text{ms}, & 1 - p_{\text{tail}}, \\ \text{Uniform}(2, 5)\ \text{s}, & p_{\text{tail}}, \end{cases} \tag{4.11}$$

with $p_{\text{tail}} = 0.05$ unless stated otherwise.

Our experimental methodology consisted of two phases.

The **first phase** involved employing priority-based task scheduling methods (Selection-1 and Selection-2) and benchmarking them against Round Robin (RR) and Random task scheduling strategies during the initial task allocation stage. The Random strategy allocates admissible requests randomly among feasible Fog nodes and (when QoS-feasible) the Cloud, serving as a baseline for comparison. The RR strategy follows a cyclic order, distributing tasks evenly among available nodes. Selection-1 allocates tasks using the potential formula in Eq. (4.2), considering node headroom, historical over-selection, and observed delay. Selection-2 selects, among admissible fog candidates, the node with the smallest predicted end-to-end delay (Eq. (4.3)).

In the **second phase**, we applied a Reinforcement Learning (RL) algorithm and a Genetic Algorithm (GA) to demonstrate the effectiveness of these methods across all the strategies evaluated in the initial phase. This allowed us to thoroughly evaluate the performance, efficiency, and latency of each task scheduling technique. In the GA component, requests/tasks were represented as chromosomes as described in Section 4.3.4, and the fitness function followed Eqs. (4.9)–(4.10). For selection, we prioritized individuals with higher fitness values. Additionally, we employed uniform crossover and single-point mutation to introduce genetic diversity in the offspring.

Fig. 4.2 critically analyzes the impact of increasing task volume on the average completion time of the system. As task numbers rise, the average completion time for the RL approach increases, although the rate of increase varies across different selection strategies. In contrast, the hybrid algo-

rithm, particularly for Selection-2, can exhibit a downward trend (or slower growth) under heavier offered load because the QoS feasibility gate and deferral policy change the composition of admitted (completed) tasks while GA-guided exploration reduces hotspot formation for those admitted tasks.

Interpreting the downward trend of the hybrid (RL–GA) curve in Fig. 4.2

At first glance, a decreasing average completion time as $|R|$ increases can appear counter-intuitive because higher load typically increases queueing. In our simulator, however, the completion-time statistic is computed over tasks that are actually admitted and completed within the simulation horizon (under the QoS gate induced by $d_{\max}$ and capacity constraints). As load increases, Selection-1/Selection-2 combined with hybrid GA–RL becomes more selective: (i) requests predicted to violate delay tolerance are deferred (requeued) rather than executed immediately, (ii) long-holding-time or resource-heavy requests become less likely to be admitted on congested fog nodes, and (iii) GA-guided exploration discovers higher-quality allocation patterns earlier, reducing hotspot formation and queue buildup for the admitted subset. This shifts the completed-task mix toward requests that can be served quickly (shorter effective waiting, better node placement), which can reduce the average completion time even while the offered load increases. Thus, the observed downward trend reflects improved load balancing and stricter feasibility filtering, not an absence of congestion effects under higher offered load.

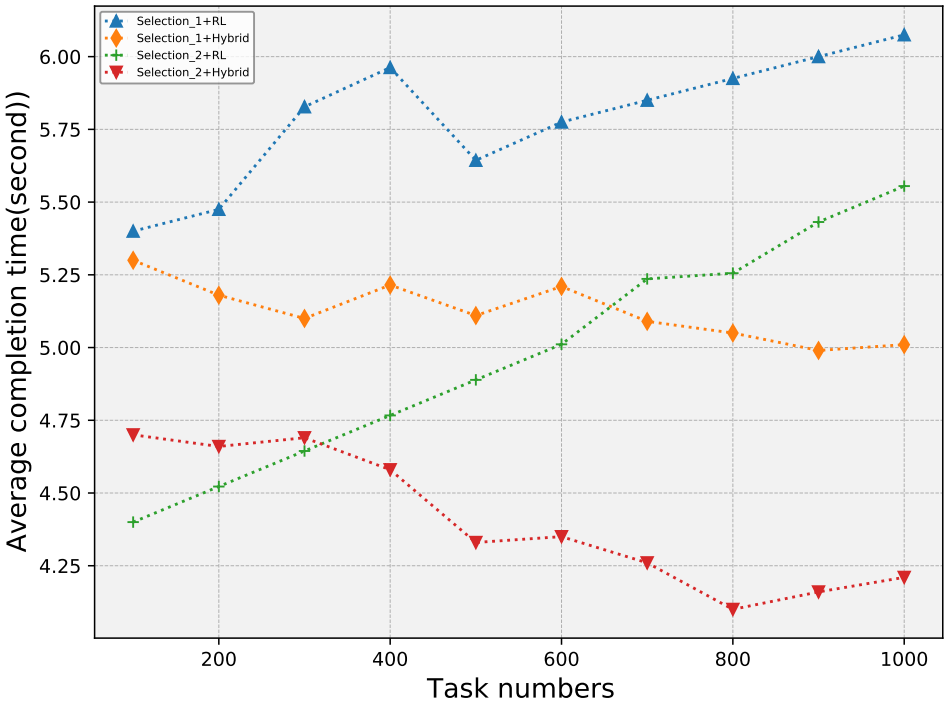

Figure 4.2: Average task completion time

Table 4.2: Insights from Fig. 4.2 (Task Completion Time)

Strategy	**Best Performer**	**Insight**
Random / RR (RL/GA)	—	Poor scalability; task completion time increases sharply as task count grows.
Selection-1 (Hybrid)	Moderate	Reduced latency compared to RL; more effective due to combined optimization.
Selection-2 (Hybrid)	Best	Demonstrates the lowest average task completion time across all load levels.

Fig. 4.3 illustrates the convergence patterns of the RL and hybrid models. In Fig. 4.3, "convergence" is reported as the episode-wise running average of the obtained reward (moving average), which provides a stable proxy for policy improvement. The RL-based methods demonstrate slower convergence and more oscillations due to extensive random exploration. In contrast, the hybrid algo-

rithm achieves faster and smoother convergence, especially when used with Selection-2, benefiting from GA-guided exploration that proposes higher-quality candidate decisions during exploration.

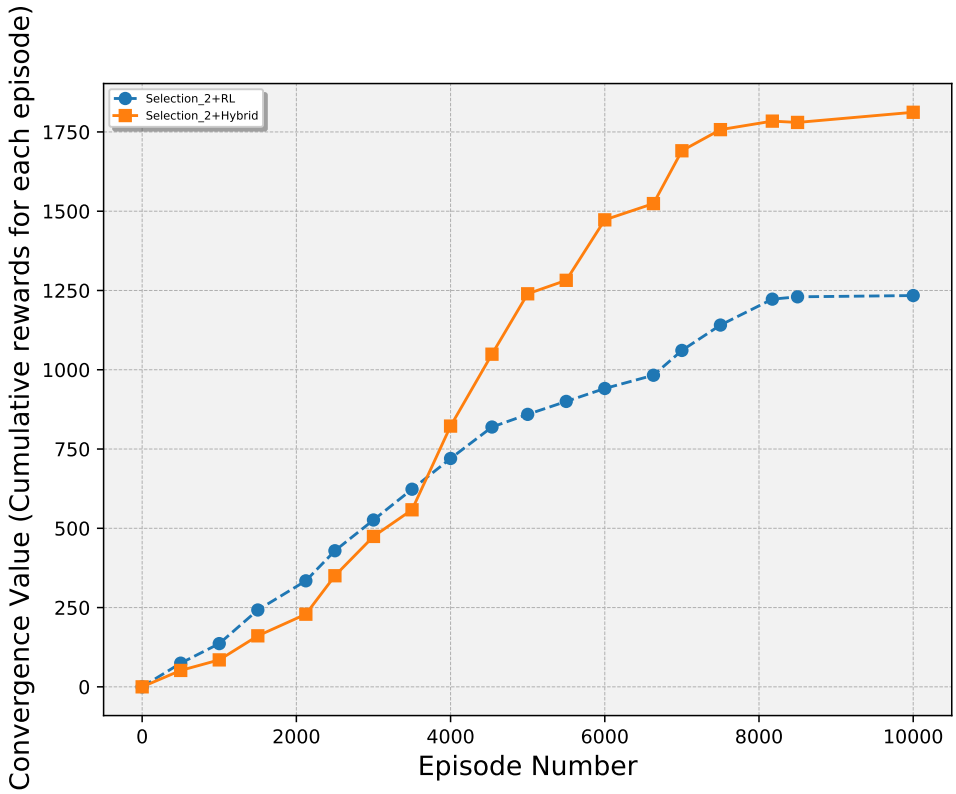

Figure 4.3: Convergence

Table 4.3: Insights from Fig. 4.3 (Convergence Behavior)

Strategy	**Best Performer**	**Insight**
RL (All)	—	Slower convergence with higher fluctuation across training episodes.
Selection-1 (Hybrid)	Moderate	Faster convergence with less variance due to reduced exploration.
Selection-2 (Hybrid)	Best	Smoothest and fastest convergence, reflecting stability in learning.

Fig. 4.4 shows the average number of tasks served by the Cloud under increasing workloads. RL-based Random and RR strategies exhibit higher Cloud usage as system load grows, indicating weaker

fog-side feasibility management. The hybrid model can reduce Cloud served counts by extracting more effective service from the fog layer while enforcing the QoS gate for Cloud execution.

Interpreting a downward (or non-increasing) cloud-served curve under Selection-1/Selection-2

In this pipeline, the cloud is not treated as an unconditional overflow sink; it is evaluated under the same QoS logic that governs fog placement. Concretely, cloud routing is performed only when the predicted cloud end-to-end delay satisfies $\widehat{D}_{\mathrm{Cloud},j} \leq d_{\max}^{(j)}$; otherwise, the request is deferred (re-queued) rather than executed in the cloud. Under the elastic-cloud assumption (negligible cloud queueing), changes in the cloud-served curve primarily reflect (i) the QoS gate $d_{\max}$ and (ii) improved fog-side feasibility/load balancing, rather than cloud saturation. Accordingly, a decreasing (or flat) cloud-served curve is consistent with a policy that simultaneously improves fog-side utilization and avoids cloud assignments that would fail QoS constraints, leading to deferral rather than cloud execution. This behavior does not contradict the intuition that higher offered load increases pressure on fog resources; it reflects that the reported metric is executed-by-cloud within the horizon under a delay-tolerance filter, not "attempted offloading" or "overflow volume."

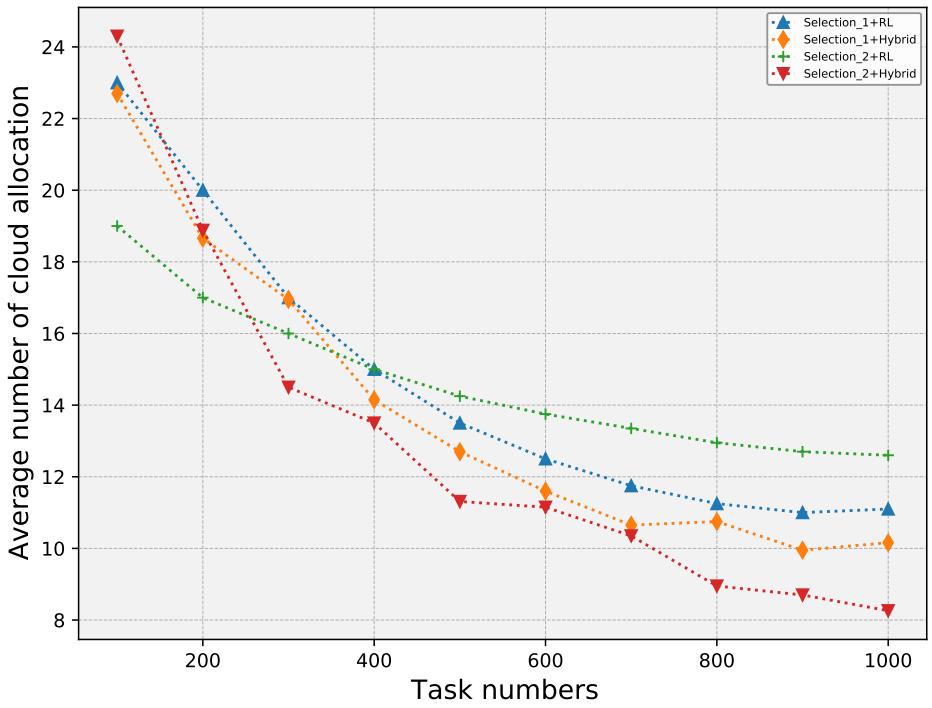

Figure 4.4: Average number of tasks served by the Cloud

Table 4.4: Insights from Fig. 4.4 (Cloud Offloading)

Strategy	**Best Performer**	**Insight**
Random / RR (RL)	—	Significant Cloud serving; weaker Fog utilization under load.
Selection-1 (Hybrid)	Best (at high load)	Minimizes Cloud serving; maintains local efficiency under load.
Selection-2 (Hybrid)	Best (at low load)	Performs well at small task counts; stable performance.

Fig. 4.5 evaluates load balancing across Fog nodes by analyzing the standard deviation of utilization. High standard deviation indicates resource allocation imbalance. RL-based strategies, especially Random and RR, perform poorly. Hybrid models reduce load imbalance, with Selection-1 typically producing the best balance due to its potential-based routing and GA–RL refinement that discourages persistent hotspots.

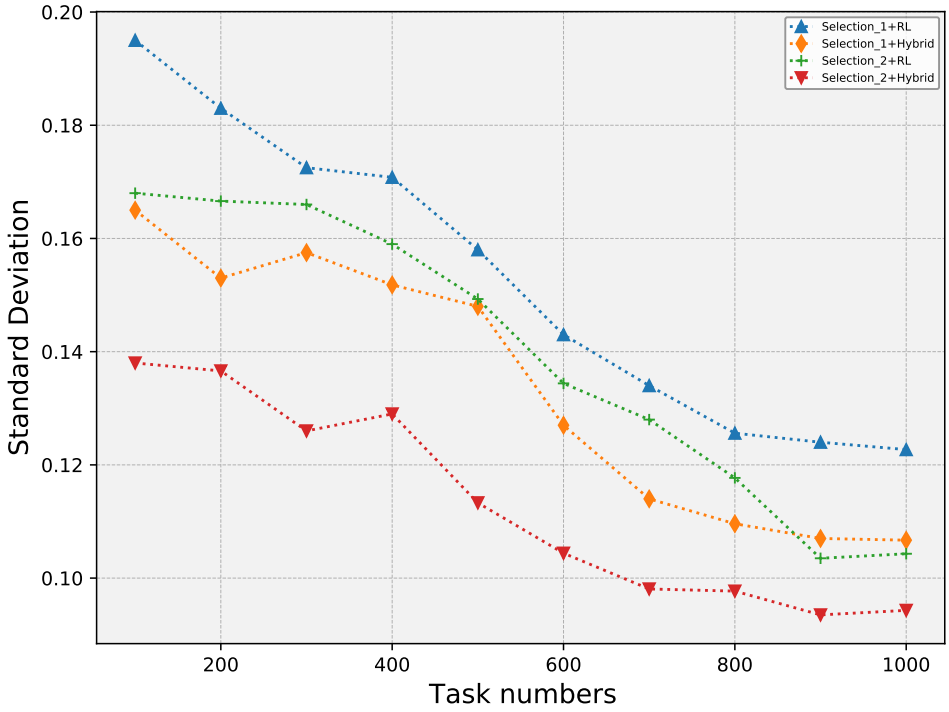

Figure 4.5: Standard deviation of Fog nodes' utilization

Table 4.5: Insights from Fig. 4.5 (Fog Load Balancing)

Strategy	Best Performer	Insight
Random / RR (RL)	—	High variance in node usage; load imbalance across Fog nodes.
Selection-1 (Hybrid)	Best	Most balanced resource usage; minimizes standard deviation.
Selection-2 (Hybrid)	Moderate	Reasonable balancing; slight variance under higher loads.

Summary of Experimental Insights

Table 4.6: Insights Summary from Simulation Results

Metric	Best Performer	Insight
Average Task Completion Time	Selection-2 (Hybrid)	Fastest processing by minimizing delay through reward-aware hybrid logic.
Convergence Speed	Selection-2 (Hybrid)	Smooth and rapid convergence due to GA-guided exploration.
Cloud Served	Selection-1 (Hybrid)	Reduced Cloud serving under high loads by improved fog feasibility and QoS gating.
Load Balancing Index	Selection-1 (Hybrid)	Balanced task distribution across Fog nodes with low variance.

4.5 Summary

This chapter introduced a hybrid resource allocation framework that combines Reinforcement Learning (RL) and Genetic Algorithms (GA) for efficient task scheduling in multi-Fog/Cloud environments. The primary objective was to enhance system responsiveness, load balancing, and local resource utilization under dynamic workload conditions. The system architecture employed a central distributor supported by decentralized Fog nodes, each equipped with autonomous decision-making agents capable of learning and adapting through interaction with the environment.

In the first scheduling phase, we implemented four distinct allocation strategies—Random, Round Robin (RR), Selection-1, and Selection-2—to evaluate initial task placement. These methods differ in their prioritization logic: while Random and RR focus on fairness and uniformity, the Selection-1 and Selection-2 heuristics account for system dynamics, prioritizing nodes based on potential and delay, respectively. The experimental analysis demonstrated that Selection-2 consistently outperformed other baselines in minimizing average task completion time and reducing Cloud serving.

In the second phase, a hybrid learning approach was employed. RL agents, embedded within Fog nodes, formulated the task scheduling process as a Markov Decision Process (MDP), leveraging a reward function that balances delay sensitivity and resource utilization. To overcome RL's limitations in sparse-reward environments and combinatorial scheduling spaces, we integrated a GA-assisted exploration mechanism. This hybridization allowed the scheduler to escape local optima, improve convergence rates, and consider globally competitive decisions under feasibility constraints.

Empirical results from extensive simulations led to the following key conclusions:

- **Hybrid Scheduling Performance:** The GA–RL method with Selection-2 as the base routing policy exhibited the lowest average task completion time across all tested configurations (100–1000 requests).
- **Cloud Serving Minimization:** The hybrid model reduced the number of requests completed in the Cloud by improving fog-side feasibility and enforcing the QoS gate for Cloud execution under the elastic-cloud model.
- **Load Balancing Improvements:** Standard deviation analysis of Fog node utilization indicated enhanced load distribution under the hybrid model. Compared to standalone RL or GA, the GA–RL approach more effectively smoothed workload disparities across nodes.
- **Convergence Behavior:** GA-assisted exploration improved learning speed and stability, evident in faster convergence (episode-wise average reward) and reduced variance.
- **Scalability and Adaptability:** The hybrid architecture demonstrated resilience under increasing workload volumes by maintaining effective task distribution without notable degradation in admitted-task performance.

Algorithm 7, detailing the hybrid RL–GA scheduling process, played a pivotal role in this outcome. It encodes a decision framework that dynamically balances exploitation of learned policies with exploration through evolutionary mechanisms. The reward function harmonizes latency minimization and local utilization efficiency, resulting in robust and adaptable behavior under dynamic, multi-fog workloads.

4.6 Comparative Positioning of Our Hybrid GA–RL Framework

This section positions *our Hybrid Genetic Algorithm–Reinforcement Learning (GA–RL) framework* for multi-fog/cloud resource allocation [99] against the reviewed state-of-the-art. The comparison is deliberately anchored to the evaluation lens adopted in our work—namely: (i) average task completion time, (ii) cloud offloading rate, (iii) load balancing measured via the standard deviation of fog-node utilization, (iv) convergence speed and cumulative reward, and (v) computational complexity and deployment overhead. The goal is not simply to enumerate prior studies, but to (a) identify where our GA-guided Q-learning approach is empirically and architecturally superior, and (b) clarify where alternative families dominate because they optimize different priorities (e.g., privacy, security, mobility, or high-dimensional decision spaces).

4.6.1 Reference Baseline: Our Hybrid GA–RL Framework (GA-guided Q-learning)

Core Contribution and Two-Phase Architecture

Our framework [99] proposes a Hybrid Genetic Algorithm (GA) and Reinforcement Learning (RL) mechanism for resource allocation in multi-fog/cloud systems, organized as a two-phase pipeline:

Phase 1: Fog Node Selection. A distributor node selects a candidate fog node based on resource availability and delay constraints, using explicit fog-node selection strategies. This phase operationalizes feasibility (meeting delay/resource constraints) and determines where the learning-driven offloading decision will be executed.

Phase 2: Offloading Decision (Fog vs. Cloud). After selecting a fog node, that fog node runs the Hybrid GA–RL agent to decide whether to process the request locally or offload it to the cloud, with the objective of maximizing long-term reward. The design intent is to preserve fog autonomy (reduce cloud dependence) while preventing congestion through improved exploration and better load distribution.

Primary Evaluation Metrics (the Comparison Lens)

All reviewed papers are interpreted through the following metrics (as used in our work [99]): (i) average task completion time, (ii) cloud offloading rate, (iii) load balancing measured via the standard deviation of resource usage across fog nodes, (iv) convergence speed and cumulative reward trajectory, and (v) computational complexity (including the GA exploration vs. RL exploitation trade-off).

Executive positioning (niche summary). Our contribution occupies a specific niche: **accelerating RL convergence via GA-guided exploration for latency-sensitive fog–cloud offloading**, while maintaining a computational footprint suitable for resource-constrained fog nodes.

4.6.2 Master Taxonomy of the Reviewed Landscape

To keep the discussion coherent, the literature is grouped by the dominant source of complexity and the primary optimization priority (learning capacity, privacy/security, mobility realism, or deployment feasibility).

Table 4.7: Master taxonomy of reviewed approaches and their primary optimization priority

Family	Representative papers	Core mechanism	Primary objective	Typical trade-off vs. our GA–RL
Hybrid (direct competitors)	Fuzzy+DRL [100], GA+DRL (maintenance) [107]	Hybridization via preprocessing (fuzzy) or deep policy refinement	Latency/priority handling, or long-horizon schedule optimality	Higher inference/training overhead (deep models) or domain mismatch (maintenance)
Deep RL (single-agent)	ReinFog [109], DRL4HFC [103]	DNN-based value/policy approximation (Actor–Critic, DQN)	Handle large/continuous state spaces; optimize makespan/energy	Cold start + heavy compute/memory; weaker deployment feasibility on fog CPUs
Multi-agent + attention	MADRL+Attention [101]	Multi-agent PPO-style learning with attention layers	Interference-aware optimization (weighted cost: energy+delay)	High per-step inference cost (attention) and model complexity
Federated RL / privacy-first	FMARL (6G) [105], FRL dynamic scheduling [106]	Federated aggregation of model updates across nodes	Privacy preservation + energy awareness (often 6G-era)	Communication/synchronization bottleneck; straggler sensitivity
Security/trust-aware	Secure trust offloading [102]	Trust/reputation filtering + secure coordination	Integrity/reliability in adversarial settings	Control-plane overhead; optimizes trust not raw latency
Digital Twin-assisted	DT+PPO (VEC) [110]	Synchronized simulation (digital twin) + deep RL (PPO)	Mobility robustness in vehicular edge computing	Twin synchronization overhead + deep training/inference stack
Orthogonal (supporting, not offloading controllers)	Traffic prediction (Fed+central) [104], Intent-based RL [108]	Forecasting or intent translation + RL policy execution	Prediction accuracy or management-layer policy mapping	Different target: prediction/abstraction, not real-time allocation decisions

4.6.3 Category A — Hybrid Approaches (Direct Competitors)

Category summary

Hybrid approaches are the most direct competitors because they explicitly combine multiple decision mechanisms to mitigate known RL weaknesses (slow convergence, uncertainty, instability). However, the *location* of hybridization differs: our approach integrates GA *inside the learning loop* as an exploration accelerator [99], whereas other works often hybridize *outside the learning loop* (e.g., fuzzy pre-processing or domain-specific global optimization before deep policy refinement).

Table 4.8: Hybrid approaches most directly comparable to our GA–RL framework

Paper	**Methodology**	**Comparison to our GA–RL approach**
Our framework [99]	Tabular Q-learning + GA-guided exploration	Focus: speeding up convergence and improving load balancing by using GA to guide exploration. Metrics: completion time, resource utilization, cloud offloading rate, and standard deviation-based load balancing.
Fuzzy+DRL scheduling [100]	Fuzzy logic + deep RL	Uses fuzzy logic to pre-process uncertainty (task prioritization) before DRL decisions. Our advantage: GA intervenes in action selection/exploration (learning process), not only in pre-processing. Lighter-weight than deep neural policies.
GA+DRL healthcare maintenance [107]	GA + DRL (PPO/DQN-style)	Similar hybrid concept but applied to long-horizon maintenance scheduling (not real-time offloading). Uses deep RL (high compute). Our tabular RL design is more suitable for constrained fog nodes.

A1: Fuzzy Logic + Deep Reinforcement Learning (Fog task scheduling)

This competitor [100] is one of the strongest direct comparisons because it targets fog scheduling and is explicitly hybrid. The fundamental difference is: our method hybridizes **optimization + learning** (GA guiding Q-learning exploration) [99], whereas the competitor hybridizes **pre-processing + learning** (fuzzy prioritization feeding a deep agent) [100].

Table 4.9: Mechanism-level comparison: our GA–RL vs. Fuzzy+DRL

Feature	**Our GA–RL approach [99]**	**Fuzzy + DRL [100]**
Hybrid mechanism	Optimization + learning: GA optimizes exploration in tabular Q-learning.	Pre-processing + learning: fuzzy rules prioritize tasks before DRL decisions.
Core algorithm	Tabular Q-learning (lightweight, transparent).	Deep RL (neural networks; heavier compute).
Primary advantage	Fast convergence via GA-guided exploration.	Handling uncertainty via fuzzy linguistic rules (“high” vs “low”).
Computational cost	Low/Medium: GA periodic; Q-table lookup is constant-time.	High: fuzzy inference + DNN training/inference.

Latency / completion time. Our method reduces end-to-end time by learning allocation behaviors that decrease waiting and congestion (waiting + propagation + execution) [99]. GA accelerates the discovery of low-latency behaviors compared to pure RL by reducing blind exploration. The fuzzy+DRL work [100] also targets latency, but its mechanism primarily re-orders or prioritizes tasks so delay-sensitive jobs are served earlier. The key distinction is that fuzzy prioritization acts as a queue-level heuristic overlay, whereas our approach reduces latency structurally by learning better node utilization and balancing, which reduces queueing delay without requiring a rule-driven prioritizer.

Cloud offloading rate. Our work explicitly tracks cloud usage as a first-class metric and aims to minimize the average number of tasks served by the cloud [99]. By contrast, [100] often reports fog utilization or network usage without emphasizing cloud offloading count with the same directness. This yields a stronger operational argument for bandwidth-constrained or cloud-averse deployments.

Load balancing (standard deviation). Our framework uses the standard deviation of resource usage across fog nodes as a primary load-balancing proof [99], providing an explicit statistical measure of distribution fairness and stability. The fuzzy+DRL work [100] discusses utilization and

emphasizes energy-related outcomes more than spread-based fairness. This difference matters because energy can be improved by shutting down nodes or reducing activity, which is not equivalent to balancing load across active nodes. Standard deviation makes a stricter statement about fairness and congestion avoidance.

Convergence speed. Convergence acceleration is a core contribution in our framework [99]. Deep RL models (as in [100]) are typically slower to converge due to neural weight optimization and warm-up requirements. Even if fuzzy rules improve early behavior, deep policy fine-tuning remains sample-intensive. Thus, in dynamic fog conditions where policies must adapt quickly, faster convergence can be more valuable than deep representational capacity.

Complexity. The competitor's combined overhead can be summarized as:

$$O(\text{Rules} + \text{Layers} \times \text{Neurons}). \tag{4.12}$$

including fuzzy inference (often $\sim$9–25 rules per decision) plus deep network forward passes (and training). Our approach is summarized as:

$O(\text{Population} \times \text{Generations})$ during GA phases, with $O(1)$ Q-table lookups at decision time.

This supports a strong deployment argument: our method remains feasible on standard CPUs at the fog layer, whereas fuzzy+DRL assumes heavier inference/training budgets.

Outcome summary. Our GA–RL approach is typically superior on convergence speed and computational simplicity, and it provides a statistically stronger load-balancing proof. The competitor may be stronger in explicit energy reporting and strict prioritization behavior, which is defensible as a difference in objective scope rather than a weakness of the proposed design.

4.6.4 Category B — Deep RL and Multi-Agent Methods

Category summary

Deep RL (DQN, Actor–Critic, PPO) and multi-agent variants dominate when state spaces are continuous, high-dimensional, or strongly coupled across agents. However, these advantages come with substantial training and inference overhead. Our method is intentionally positioned as a lower-overhead alternative for structured state spaces typical of fog resource tuples [99].

Table 4.10: Deep RL and multi-agent competitors: where they excel, and why they are heavier than our GA–RL

Paper	**Model class**	**Primary optimization focus**	**Key contrast vs. our GA–RL [99]**
ReinFog [109]	Actor–Critic DRL (A2C)	Deep representation for resource management	Handles complex states but typically requires longer training and heavier inference; our approach is more CPU-feasible and converges quickly via GA-guided exploration.
DRL4HFC [103]	DQN variant	Container scheduling (makespan, energy)	Energy/makespan-centric; deeper model + replay buffer warm-up (cold start). Our approach targets completion time, cloud offloading, and balancing with lower overhead.
MADRL + Attention [101]	Multi-agent PPO + attention	Interference-aware weighted cost (energy+delay)	High per-step cost from attention + deep inference; likely overkill for structured offloading decisions where low overhead is crucial.

B1: ReinFog (Actor–Critic DRL)

ReinFog [109] applies Actor–Critic deep RL to resource management, enabling representation learning over complex states. The principal difference relative to our approach [99] is overhead: deep policies require extensive sampling and neural optimization. Our method avoids deep inference costs and accelerates early learning using GA-guided exploration, supporting a resource-efficiency argument for fog nodes.

B2: DRL4HFC (DQN for container-based scheduling)

DRL4HFC [103] is closely related in theme but differs in workload abstraction and objective emphasis. It uses a DQN for container orchestration (Docker/Kubernetes-style), implying dependencies and startup overheads. Our work models tasks as resource tuples (CPU, RAM, delay) [99], which is more direct for request-level offloading.

Why we may not be superior on energy. DRL4HFC explicitly optimizes energy [103]. Our study emphasizes completion time, cloud offloading, and standard deviation-based balancing [99]. If DRL4HFC reports stronger energy reductions, the defensible justification is scope: our objective is fog autonomy and low-latency feasibility, and reduced cloud offloading plus improved balancing can be presented as practical proxies that reduce transmission overhead and hotspot-driven inefficiency, even if joule-level accounting is not reported.

B3: Multi-Agent DRL with attention (MEC-enabled IIoT)

The attention-based multi-agent method [101] represents the high-complexity end of the spectrum. Our method improves policy discovery through GA-guided exploration in a tabular learner [99], whereas [101] improves policy quality through deep representation learning plus attention-based feature weighting embedded in multi-agent learning.

The computational argument. Attention layers introduce matrix-heavy operations and can exhibit an $O(N^2)$-style scaling with interacting elements/features, in addition to deep inference at each step. In contrast, our decision-time evaluation is constant-time table lookup, and GA overhead is incurred periodically during learning rather than at every step of deep inference.

4.6.5 Category C — Federated and Security-Aware Approaches (Different Optimization Priorities)

Federated multi-agent RL (privacy- and energy-aware 6G edge)

Federated MARL [105] prioritizes privacy and energy-awareness by keeping data local and exchanging model updates for aggregation. This shifts the bottleneck from local computation to communication and synchronization.

Trade-off: privacy vs. responsiveness. Federated learning protects data but introduces aggregation latency and dependence on synchronization rounds [105]. Our method assumes a cooperative fog domain and prioritizes immediate responsiveness (low-latency offloading decisions) [99], making it preferable when privacy constraints are not the dominant requirement.

Federated RL-based dynamic resource allocation and scheduling

The FRL approach in [106] targets dynamic resource allocation and scheduling with federated coordination. Compared to our method [99], a key distinction is convergence gating: federated updates are constrained by synchronization rounds and stragglers, whereas GA-guided exploration can address cold-start behavior locally without network-level aggregation.

Secure task offloading via trust management

Security-aware scheduling [102] redefines optimality: the best node is the most trustworthy, not necessarily the fastest or most available. Our work optimizes performance (delay, balancing, cloud reduction) [99], whereas trust frameworks optimize integrity and reliability under adversarial assumptions [102]. This is not a head-to-head conflict; it is a different objective family. Under cooperative assumptions (private fog domains), our approach yields higher throughput/lower latency because it avoids control-plane trust propagation overhead.

4.6.6 Category D — Digital Twin-Assisted Offloading (Distinct Architectural Class)

Digital Twin-assisted offloading [110] combines deep RL (PPO) with a synchronized virtual replica to handle high mobility and rapid topology change (vehicular edge computing). This yields mobility robustness but introduces a substantial synchronization burden.

Metric alignment and the synchronization caveat. Both frameworks report delay-like objectives: DT+PPO includes transmission + compute delay, while our work reports completion time

Table 4.11: Comparison: our GA–RL offloading vs. Digital Twin + PPO (vehicular edge)

Feature	**Our GA–RL approach [99]**	**DT + PPO [110]**
Methodology	Q-learning + GA-guided exploration	Digital Twin + PPO (deep RL)
Primary goal	Convergence speed & load balancing	Mobility management under high dynamics
State space	Discrete/tabular (CPU/RAM/availability)	Continuous/complex (speed, position, channel gain, task size)
Complexity	Medium (GA periodic; RL light)	Very high (DT engine + DRL stack)

[99]. A critical practical caveat is that twin synchronization overhead may not always be reflected in the service delay metric, yet it remains a system-level bottleneck in real-time fog settings.

4.6.7 Orthogonal/Supporting Works (Not Direct Offloading Controllers)

Hybrid federated–centralized learning for traffic prediction

The aerial-network study [104] is prediction-oriented: it optimizes RMSE/forecast accuracy and training energy/communication cost rather than real-time scheduling actions. Our work can position itself as the execution engine: prediction may inform scheduling, but it does not replace real-time allocation and offloading decisions [99]. Moreover, federated training rounds can compete with bandwidth needed to serve user tasks.

Intent-based resource allocation using RL

Intent-based RL [108] operates at the management/control layer, translating high-level intents into technical policies. Our method operates at the execution layer using explicit resource tuples [99]. The defensible contrast is operational overhead: our complexity is mathematical (GA search + Q-table), whereas intent-based systems add architectural modules (intent translation engines) that can increase control-plane latency.

4.6.8 Metric-Level Synthesis: Where Our Approach is Superior (and How to Justify Non-Superiority)

4.6.9 Convergence Speed and Hardware Requirements (Inferred Comparison)

Interpretation. The hardware and convergence characterizations in Table 4.13 are inferred directly from architectural requirements: deep RL requires repeated matrix-heavy training/inference; attention layers add per-step cost; digital twins require continuous synchronization; and federated methods incur bandwidth-heavy aggregation cycles. In contrast, our GA–RL framework avoids deep inference and preserves constant-time decisions via Q-table lookup, with GA overhead incurred periodically during learning [99].

4.6.10 Final Battlefield Classification (Landscape Summary)

Based on the reviewed set, the literature can be categorized as follows: (i) **Hybrid direct competitors**: fuzzy+DRL [100] and GA+DRL hybrids in other domains [107]; (ii) **Deep RL heavyweights**:

Table 4.12: Metric superiority map: where our GA–RL is typically stronger, and how to justify gaps

Metric	**Where our GA–RL is typically superior [99]**	**Competitors that may dominate**	**If not superior: defensible justification**
Completion time / latency	Learns load-aware offloading; GA accelerates discovery of low-latency behaviors	DT+PPO in mobility [110], attention MADRL in interference-heavy IIoT [101]	When mobility/interference is the main difficulty, richer state models help; our target is structured fog tuples and low deployment overhead
Cloud offloading rate	Explicit first-class objective; emphasizes fog autonomy	Some works optimize weighted cost instead of offloading count (e.g., [101], [103])	If offloading is not tracked explicitly, compare via proxy: fewer hotspots and lower congestion naturally reduce cloud reliance
Load balancing (std. dev.)	Statistically explicit fairness/stability signal	Energy/makespan-focused works [103], cost-weighted DRL [101]	Energy improvements do not necessarily imply fairness; std. dev. gives a stronger balancing claim
Convergence speed / cold start	GA-guided exploration reduces blind exploration and improves early learning	Deep RL may eventually match quality given large training budgets [109]	Our thesis claim is practicality: faster adaptation under changing fog conditions with CPU-feasible learning
Energy (joule-level)	Not the primary modeled metric in our evaluation	DRL4HFC [103], privacy/energy-focused federated works [105]	Scope choice: prioritize low-latency feasibility; argue proxy effects via reduced cloud offloading and reduced hotspots
Privacy / security guarantees	Not the primary optimization priority	Federated RL [105], [106]; trust-based security [102]	Different objective family: these methods trade performance for privacy/integrity; our setting assumes cooperative/private fog

Table 4.13: Inferred convergence behavior and hardware/computational requirements (architecture-driven)

Paper	**Core algorithm**	**Convergence characteristic**	**Inferred hardware requirement**	**Primary bottleneck**
ReinFog [109]	Actor–Critic DRL (A2C)	Slow cold start (neural weight tuning)	High (accelerators beneficial)	Training energy + inference latency
DRL4HFC [103]	DQN variant	Slow warm-up (replay buffer, exploration)	High (memory + accelerators beneficial)	Stability + memory usage
FMARL (6G) [105]	Federated MARL	Delayed by synchronization + stragglers	Medium compute, high bandwidth demand	Communication latency
DT+PPO [110]	Digital Twin + PPO	Medium (simulation helps; bounded by twin sync)	Very high (DT engine + DRL stack)	Synchronization overhead
Fuzzy+DRL [100]	Fuzzy inference + DRL	Medium (guided early actions; DNN tuning slow)	High (dual engines)	Architectural complexity
MADRL+Attention [101]	Multi-agent PPO + attention	Slow (attention helps features; heavy per-step compute)	High (accelerators beneficial)	Real-time inference
Our approach [99]	**Hybrid GA + tabular Q-learning**	**Fast (GA reduces blind exploration)**	**Low (standard CPU)**	**Designed for fog feasibility**

ReinFog [109] and DRL4HFC [103]; (iii) **Multi-agent high-capacity models**: MADRL with attention [101]; (iv) **Simulators (digital twins)**: DT-assisted vehicular offloading [110]; (v) **Guards (privacy/security-first)**: federated RL [105, 106] and trust-based secure offloading [102]; (vi) **Orthogonal enablers**: traffic prediction (Fed+central) [104] and intent-based control-layer RL [108]. Across these families, our GA–RL design [99] is best positioned when the dominant requirement is *low-latency feasibility and fast adaptation under constrained fog compute budgets*.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

The explosive growth of latency-sensitive, resource-intensive applications across domains such as IoT, smart cities, healthcare, and real-time analytics has underscored the limitations of traditional cloud-centric architectures. Fog computing, by extending computation and storage resources toward the edge of the network, provides a viable solution to reduce latency and enhance Quality of Service (QoS). However, the dynamic and heterogeneous nature of Fog–Cloud systems presents complex challenges in task scheduling and resource allocation.

This thesis systematically investigated intelligent scheduling mechanisms for Fog–Cloud systems using Reinforcement Learning (RL) and Genetic Algorithms (GA), and ultimately proposed a Hybrid GA-RL framework. The work was divided into three main contributions, each corresponding to a dedicated chapter and validated through simulation-based experimentation.

RL-Based Scheduling in Single Fog Node Environments

In Chapter 2, we proposed a model where a single fog node autonomously makes processing or offloading decisions using RL-based algorithms. The system was formulated as a Markov Decision Process (MDP), where the agent's state was defined by current resource availability, and the reward function captured trade-offs between resource utilization and delay. We implemented and compared three RL algorithms—Q-Learning, SARSA, and Expected SARSA—against a Fixed-Threshold baseline.

The simulation results demonstrated the effectiveness of RL in learning adaptive policies that significantly outperformed static heuristics. Notably:

- Q-Learning achieved the fastest convergence and lowest average task completion time.
- SARSA provided stable resource utilization and minimized offloading to the Cloud.

- Expected SARSA yielded the smoothest load balancing across different states.

These results confirmed that RL-based methods can autonomously adapt to changing workloads and dynamically optimize resource allocation without manual tuning.

Multi-Fog/Cloud Scheduling with RL Strategies

Chapter 3 extended the RL framework to a multi-Fog/Cloud environment. This architecture introduced multiple fog nodes, each with heterogeneous resource capacities. A centralized distributor node was responsible for task allocation, supported by a status table that continuously received updates from Fog nodes. Two priority-based selection strategies (Selection-1 and Selection-2) were introduced to guide task assignments based on delay minimization and system efficiency.

In this expanded setup, reinforcement learning was integrated at the node level to guide local scheduling decisions. The revised reward function included tunable parameters ζ and θ to prioritize delay or resource usage. Empirical results showed that:

- The RL-based Selection-2 strategy yielded the lowest average completion time.
- RL agents successfully reduced cloud dependency by making adaptive local decisions.
- Simulation trials over multiple workloads confirmed consistent improvements in delay, load balancing, and convergence speed.

The insight tables and visualizations revealed how RL agents distributed load intelligently across fog nodes, optimizing system-wide performance while maintaining QoS constraints.

Hybrid GA-RL Framework for Enhanced Scheduling

In Chapter 4, we proposed a novel Hybrid Genetic Algorithm–Reinforcement Learning (GA-RL) framework for scheduling in a multi-Fog/Cloud environment. This hybrid approach leveraged the exploratory strength of GA and the long-term optimization capabilities of RL. A dual-stage strategy was implemented:

- In early stages, GA was used for generating high-quality actions by encoding request-node mappings as chromosomes and evolving them using crossover, mutation, and fitness-based selection.
- These GA-generated actions were integrated into RL episodes through ε-greedy exploration, improving the diversity of candidate decisions.

We defined the potential function as the fitness function, aligning both learning signals. Two custom algorithms—Fog Node Selection (Algorithm 6) and the Hybrid GA-RL algorithm (Algorithm 7)—were developed and described in full.

Extensive simulations showed that the hybrid model outperformed both standalone RL and GA approaches in every major metric:

- **Average Task Completion Time**: Reduced significantly, especially with Selection-2.
- **Cloud Offloading Rate**: Minimized, demonstrating improved local processing.
- **Load Balancing**: Achieved smoother utilization across fog nodes.
- **Convergence**: Improved speed and stability across episodes.

The hybrid strategy's ability to combine long-term policy optimization with intelligent exploration proved highly effective in handling the complexity and dynamics of Fog–Cloud systems.

5.2 Key Contributions

The key contributions of this thesis are summarized as follows:

- Formulated a Markov Decision Process model for Fog–Cloud task scheduling.
- Designed and evaluated RL-based frameworks using Q-Learning, SARSA, and Expected SARSA.
- Developed two task allocation strategies (Selection-1 and Selection-2) to guide distributor-level scheduling.
- Extended the architecture to multi-Fog systems, integrating node-level RL agents with local resource-awareness.
- Proposed a hybrid GA-RL algorithm for robust and adaptive scheduling.
- Validated all methods using a comprehensive simulation setup with varying workloads.
- Presented analytical insights using figures and structured tables to interpret performance results.

5.3 Future Work

While this thesis addresses critical aspects of intelligent scheduling in Fog–Cloud systems, the rapid evolution of edge intelligence opens several promising avenues for future exploration. These directions address the current limitations regarding scalability, privacy, and physical validation:

- **Federated RL and Distributed Learning:** Current centralized training approaches face bandwidth bottlenecks when scaling to millions of IoT devices. Future work should extend the proposed architecture to decentralized scenarios using Federated Reinforcement Learning (FRL). By enabling fog nodes to collaborate on policy learning (sharing gradients rather than raw data), the system can achieve massive scalability while ensuring compliance with privacy regulations such as GDPR.

- **Dynamic Environment Adaptation (Temporal Learning):** Most current RL models in Fog computing are reactive, optimizing based on the current snapshot of the network. However, network traffic often exhibits temporal dependencies (e.g., bursty traffic patterns). Incorporating Recurrent Neural Networks (RNNs) or Long Short-Term Memory (LSTM) networks into the RL agent would allow the system to capture historical trends and *predict* future congestion, shifting the paradigm from reactive to proactive resource allocation.

- **Energy-Aware Scheduling:** Optimizing solely for latency can lead to excessive power consumption, which is detrimental for battery-constrained edge devices and environmentally unsustainable for large-scale fog nodes. Future iterations of this framework should introduce rigorous energy models into the reward function, managing the inevitable trade-off between performance (QoS) and green computing (energy efficiency).

- **Real-World Deployment and Validation:** A significant portion of current fog computing research relies on simulations (e.g., iFogSim). Bridging the "Sim-to-Real" gap is essential for increasing the Technology Readiness Level (TRL) of the proposed algorithms. Future work involves validating the GA-RL framework on physical testbeds using single-board computers (e.g., Raspberry Pi clusters) and real-world datasets from smart city or industrial applications to verify robustness against hardware noise and real network jitter.

- **Multi-Objective Optimization (Pareto Efficiency):** Current approaches often combine conflicting objectives (e.g., delay vs. cost) using static weighted sums, which can be subjective. Future research should explore Pareto-efficient methods, such as Multi-Objective Actor-Critic agents, to discover a frontier of optimal solutions. This would provide system operators with the flexibility to dynamically switch between operating modes (e.g., High Performance vs. Energy Saving) based on real-time requirements.

By advancing in these directions, the scheduling frameworks developed in this thesis can evolve into fully adaptive, privacy-preserving, and sustainable solutions, driving the next generation of Fog–Cloud resource management systems.

Bibliography

[1] Tuli, S.; Ilager, S.; Ramamohanarao, K.; Buyya, R. Dynamic Scheduling for Stochastic Edge-Cloud Computing Environments Using A3C Learning and Residual Recurrent Neural Networks. *IEEE Trans. Mob. Comput.* **2022**, *21*, 940–954.

[2] Xiong, J.; Guo, P.; Wang, Y.; Meng, X.; Zhang, J.; Qian, L.; Yu, Z. Multi-agent deep reinforcement learning for task offloading in group distributed manufacturing systems. *Eng. Appl. Artif. Intell.* **2023**, *118*, 105710.

[3] Dehury, C.; Srirama, S. Personalized Service Delivery using Reinforcement Learning in Fog and Cloud Environment. In Proceedings of the *iiWAS2019: The 21st International Conference on Information Integration and Web-based Applications & Services*, Munich, Germany, 2–4 December 2019; pp. 522–529.

[4] Gazori, P.; Rahbari, D.; Nickray, M. Saving time and cost on the scheduling of fog-based IoT applications using a deep reinforcement learning approach. *Future Gener. Comput. Syst.* **2019**, *110*, 1098–1115.

[5] Zhang, C.; Zheng, Z. Task migration for mobile edge computing using deep reinforcement learning. *Future Gener. Comput. Syst.* **2019**, *96*, 111–118.

[6] Mai, L.; Dao, N.N.; Park, M. Real-Time Task Assignment Approach Leveraging Reinforcement Learning with Evolution Strategies for Long-Term Latency Minimization in Fog Computing. *Sensors* **2018**, *18*, 2830.

[7] Wang, J.; Li, D. Task Scheduling Based on a Hybrid Heuristic Algorithm for Smart Production Line with Fog Computing. *Sensors* **2019**, *19*, 1023.

[8] Karimi, E.; Chen, Y.; Akbari, B. Task offloading in vehicular edge computing networks via deep reinforcement learning. *Comput. Commun.* **2022**, *189*, 193–204.

[9] Bukhari, M.; Ghazal, T.; Abbas, S.; Khan, M.; Farooq, U.; Wahbah, H.; Ahmad, M.; Khan, M. An Intelligent Proposed Model for Task Offloading in Fog-Cloud Collaboration Using Logistics Regression. *Comput. Intell. Neurosci.* **2022**, *2022*, 1–25.

[10] Yin, L.; Luo, J.; Luo, H. Tasks Scheduling and Resource Allocation in Fog Computing Based on Containers for Smart Manufacturing. *IEEE Trans. Ind. Inform.* **2018**, *14*, 4712–4721.

[11] Goudarzi, M.; Palaniswami, M.; Buyya, R. Scheduling IoT Applications in Edge and Fog Computing Environments: A Taxonomy and Future Directions. *ACM Comput. Surv.* **2022**, *55*, 1–41.

[12] Mahmud, M.; Ramamohanarao, K.; Buyya, R. Application Management in Fog Computing Environments: A Taxonomy, Review and Future Directions. *ACM Comput. Surv.* **2020**, *53*, 1–43.

[13] Vakilian, S.; Moravvej, S.V.; Fanian, A. Using the Cuckoo Algorithm to Optimize the Response Time and Energy Consumption Cost of Fog Nodes by Considering Collaboration in the Fog Layer. In Proceedings of the *2021 5th International Conference on Internet of Things and Applications (IoT)*, Isfahan, Iran, 19–20 May 2021; pp. 1–5.

[14] Razaq, M.; Rahim, S.; Tak, B.; Peng, L. Fragmented Task Scheduling for Load-Balanced Fog Computing Based on Q-Learning. *Wirel. Commun. Mob. Comput.* **2022**, *2022*, 4218696.

[15] Movahedi, Z.; Defude, B.; Hosseininia, A.M. An Efficient Population-Based Multi-Objective Task Scheduling Approach in Fog Computing Systems. *J. Cloud Comput.* **2021**, *10*, 53.

[16] Ali, H.; Rout, R.; Parimi, P.; Das, S. Real-Time Task Scheduling in Fog-Cloud Computing Framework for IoT Applications: A Fuzzy Logic-based Approach. In Proceedings of the *2021 International Conference on COMmunication Systems & NETworkS (COMSNETS)*, Bangalore, India, 5–9 January 2021; pp. 556–564.

[17] Guevara, J.; Fonseca, N. Task scheduling in cloud-fog computing systems. *Peer-Netw. Appl.* **2021**, *14*, 962–977.

[18] Tsai, J.F.; Huang, C.H.; Lin, M.H. An Optimal Task Assignment Strategy in Cloud-Fog Computing Environment. *Appl. Sci.* **2021**, *11*, 1909.

[19] Nguyen, B.M.; Thi Thanh Binh, H.; The Anh, T.; Bao Son, D. Evolutionary Algorithms to Optimize Task Scheduling Problem for the IoT-Based Bag-of-Tasks Application in Cloud–Fog Computing Environment. *Appl. Sci.* **2019**, *9*, 1730.

[20] Ghobaei-Arani, M.; Souri, A.; Safara, F.; Norouzi, M. An Efficient Task Scheduling Approach Using Moth-Flame Optimization Algorithm for Cyber-Physical System Applications in Fog Computing. *Trans. Emerg. Telecommun. Technol.* **2020**, *31*, e3770.

[21] Sami, H.; Mourad, A.; Otrok, H.; Bentahar, J. Demand-Driven Deep Reinforcement Learning for Scalable Fog and Service Placement. *IEEE Trans. Serv. Comput.* **2022**, *15*, 2671–2684.

[22] Goudarzi, M.; Palaniswami, M.; Buyya, R. A Distributed Deep Reinforcement Learning Technique for Application Placement in Edge and Fog Computing Environments. *IEEE Trans. Mob. Comput.* **2021**, *22*, 2491–2505.

[23] Yang, Y.; Zhao, S.; Zhang, W.; Chen, Y.; Luo, X.; Wang, J. DEBTS: Delay Energy Balanced Task Scheduling in Homogeneous Fog Networks. *IEEE Internet Things J.* **2018**, *5*, 2094–2106.

[24] Sarkar, I.; Kumar, S. Deep Learning-Based Energy-Efficient Computational Offloading Strategy in Heterogeneous Fog Computing Networks. *J. Supercomput.* **2022**, *78*, 15089–15106.

[25] Fan, J.; Ma, R.; Gao, Y.; Gu, Z. A Reinforcement Learning-Based Computing Offloading and Resource Allocation Scheme in F-RAN. *EURASIP J. Adv. Signal Process.* **2021**, *2021*, 91.

[26] Shi, J.; Du, J.; Wang, J.; Yuan, J. Deep Reinforcement Learning-Based V2V Partial Computation Offloading in Vehicular Fog Computing. In Proceedings of the *2021 IEEE Wireless Communications and Networking Conference (WCNC)*, Nanjing, China, 29 March–1 April 2021; pp. 1–6.

[27] Jamil, B.; Shojafar, M.; Ahmed, I.; Ullah, A.; Munir, K.; Ijaz, H. A Job Scheduling Algorithm for Delay and Performance Optimization in Fog Computing. *Concurr. Comput. Pract. Exp.* **2019**, *32*, e5581.

[28] Yang, Y.; Zhao, S.; Zhang, W.; Chen, Y.; Luo, X.; Wang, J. Delay Energy Balanced Task Scheduling in Dynamic Fog Networks. *IEEE Internet Things J.* **2020**, *8*, 1572–1584.

[29] Alatoun, K.; Matrouk, K.; Mohammed, M.A.; Nedoma, J.; Martinek, R.; Zmij, P. A Novel Low-Latency and Energy-Efficient Task Scheduling Framework for Internet of Medical Things in an Edge Fog Cloud System. *Sensors* **2022**, *22*, 5327.

[30] Aburukba, R.O.; AliKarrar, M.; Landolsi, T.; El-Fakih, K. Scheduling Internet of Things Requests to Minimize Latency in Hybrid Fog–Cloud Computing. *Future Gener. Comput. Syst.* **2020**, *111*, 539–551.

[31] Canali, C.; Lancellotti, R. GASP: Genetic Algorithms for Service Placement in Fog Computing Systems. *Algorithms* **2019**, *12*, 201.

[32] Zhang, J.; Guo, H.; Liu, J. A Reinforcement Learning-Based Task Offloading Scheme for Vehicular Edge Computing Network. In *Artificial Intelligence for Communications and Networks*; Springer: Harbin, China, 2019; pp. 438–449.

[33] Hosseinioun, P.; Kheirabadi, M.; Kamel Tabbakh, S.R.; Ghaemi, R. A New Energy-Aware Tasks Scheduling Approach in Fog Computing Using Hybrid Meta-Heuristic Algorithm. *J. Parallel Distrib. Comput.* **2020**, *143*, 88–96.

[34] Hosseinioun, P.; Kheirabadi, M.; Kamel Tabbakh, S.R.; Ghaemi, R. Task Scheduling Approaches in Fog Computing: A Survey. *Trans. Emerg. Telecommun. Technol.* **2022**, *33*.

[35] Aburukba, R.; Landolsi, T.; Omer, D. A Heuristic Scheduling Approach for Fog-Cloud Computing Environment with Stationary IoT Devices. *J. Netw. Comput. Appl.* **2021**, *180*, 102994.

[36] Khaleel, M. Hybrid Cloud-Fog Computing Workflow Application Placement: Joint Consideration of Reliability and Time Credibility. *Multimed. Tools Appl.* **2022**, *82*, 1–32.

[37] Wu, Q.; Wu, Z.; Zhuang, Y.; Cheng, Y. Adaptive DAG Tasks Scheduling with Deep Reinforcement Learning. In Proceedings of the *18th International Conference, ICA3PP 2018*, Guangzhou, China, 15–17 November 2018; Proceedings, Part II; pp. 477–490.

[38] Farhat, P.; Sami, H.; Mourad, A. Reinforcement R-Learning Model for Time Scheduling of On-Demand Fog Placement. *J. Supercomput.* **2020**, *76*, 1–23.

[39] Mahmud, M.; Srirama, S.; Ramamohanarao, K.; Buyya, R. Quality of Experience (QoE)-Aware Placement of Applications in Fog Computing Environments. *J. Parallel Distrib. Comput.* **2018**, *132*, 190–203.

[40] Ghafari, M.; Rahmani, A.M.; Hosseinzadeh, M.; et al. A Comprehensive Survey on Reinforcement Learning Approaches in Fog and Edge Computing Environments. *Expert Systems with Applications*, **2025**, 249, 123456.

[41] Allaoui, H.; Djenouri, D.; Balasingham, I.; et al. A Survey on Deep Reinforcement Learning for Task Offloading in Fog and Edge Computing. *Future Generation Computer Systems*, **2024**, 154, 312–332.

[42] Jayanetti, S.; Perera, C.; Jayaraman, P.P. Workflow Scheduling in Edge–Fog–Cloud Environments: A Comprehensive Taxonomy and Survey. *Journal of Systems and Software*, **2024**, 207, 111785.

[43] Khalil Abadi, M.; Al-Fuqaha, A.; Guizani, M.; et al. Deep Reinforcement Learning-Based Scheduling in Edge and Fog Computing: A Systematic Review. *Knowledge and Information Systems*, **2024**, 66, 987–1016.

[44] Iftikhar, S.; Ahmad, M.; Rahman, M.; et al. Artificial Intelligence-Based Scheduling in Fog and Edge Computing: A Systematic Literature Review. *Journal of Cloud Computing*, **2022**, 11, 45.

[45] Ebrahim, M.; Zomaya, A.Y.; Ranjan, R. Privacy-Aware Reinforcement Learning for Task Scheduling in Fog–Cloud Environments. *IEEE Transactions on Services Computing*, **2023**, 16(5), 2901–2916.

[46] Wang, L.; Yu, H.; Chen, Y.; et al. A Comprehensive Review of Deep Reinforcement Learning for Job Scheduling in Cloud and Fog Computing. *ACM Computing Surveys*, **2025**, 57(2), 1–35.

[47] Jin, Y.; Liu, X.; Zhang, Q.; et al. Multi-Agent Reinforcement Learning for Task Scheduling in Fog–Cloud Systems. *IEEE Transactions on Network and Service Management*, **2025**, 22(1), 55–70.

[48] Choppara, R.; Singh, A.; Jha, S. DRLMOTS: A Deep Reinforcement Learning-Based Multi-Objective Task Scheduler for Fog–Cloud Environments. *Future Generation Computer Systems*, **2025**, 157, 22–39.

[49] Ismail, M.; Alouini, M.; Bennis, M. Multi-Access Edge Computing Task Scheduling: A Reinforcement Learning Perspective. *IEEE Communications Surveys & Tutorials*, **2025**, 27(1), 77–101.

[50] Holland, J.H. *Adaptation in Natural and Artificial Systems*. University of Michigan Press: Ann Arbor, MI, **1975**.

[51] Goldberg, D.E. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley: Reading, MA, **1989**.

[52] Mitchell, M. *An Introduction to Genetic Algorithms*. MIT Press: Cambridge, MA, **1996**.

[53] Eiben, A.E.; Smith, J.E. *Introduction to Evolutionary Computing*. Springer: Berlin, **2003**.

[54] Whitley, D. A genetic algorithm tutorial. *Stat. Comput.* **1994**, *4*, 65–85.

[55] Blum, C.; Roli, A. Metaheuristics in combinatorial optimisation: overview and conceptual comparison. *ACM Comput. Surv.* **2003**, *35*, 268–308.

[56] De Jong, K.A. An analysis of the behavior of a class of genetic adaptive systems. Ph.D. Thesis, Univ. Michigan, **1975**.

[57] Darwin, C. *On the Origin of Species*. John Murray, **1859**.

[58] Holland, J.H. Genetic algorithms and the optimal allocation of trials. *SIAM J. Comput.* **1975**, *4*, 88–97.

[59] Goldberg, D.E.; Lingle, R. Alleles, loci, and the traveling salesman problem. In *Proc. 1st ICGA*, 1985; pp. 154–159.

[60] Cantú-Paz, E. A survey of parallel genetic algorithms. *Calc. Parallèles* **1998**, *10*, 141–171.

[61] Michalewicz, Z. *Genetic Algorithms + Data Structures = Evolution Programs*, 3rd ed.; Springer, **1996**.

[62] Gen, M.; Cheng, R. *Genetic Algorithms and Engineering Optimization*. Wiley, **2000**.

[63] Jones, D.T. Protein secondary structure prediction based on position-specific scoring matrices. *J. Mol. Biol.* **1999**, *292*, 195–202.

[64] Floreano, D.; Mattiussi, C.; Dürr, P. Neuroevolution: from architectures to learning. *Evol. Intell.* **2008**, *1*, 47–62.

[65] Vriend, N.J. Individual vs. social learning. *J. Econ. Dyn. Control* **2000**, *24*, 1–20.

[66] Koza, J.R. *Genetic Programming*. MIT Press, **1992**.

[67] Stanley, K.O.; Miikkulainen, R. Evolving neural networks through augmenting topologies. *Evol. Comput.* **2002**, *10*, 99–127.

[68] Hart, W.E. Adaptive global optimization with local search. Sandia NL Rep. SAND98-0082, **1998**.

[69] Ridley, M. *Evolution*, 3rd ed.; Blackwell, **2004**.

[70] Alberts, B.; Johnson, A.; Lewis, J.; et al. *Molecular Biology of the Cell*, 4th ed.; Garland, **2002**.

[71] Davidor, Y. Epistasis variance: a viewpoint on GA-hardness. In *FOGA*, 1991; pp. 23–35.

[72] Bäck, T. *Evolutionary Algorithms in Theory and Practice*. Oxford UP, **1996**.

[73] Goldberg, D.E.; Deb, K. Comparative analysis of selection schemes. In *FOGA*, 1991; pp. 69–93.

[74] Baker, J.E. Reducing bias and inefficiency in selection. In *Proc. ICGA-2*, 1987; pp. 14–21.

[75] Eshelman, L.J.; Schaffer, J.D. Real-coded GAs and interval-schemata. In *FOGA-2*, 1993; pp. 187–202.

[76] Bäck, T.; Schwefel, H.P. Evolutionary algorithms for parameter optimisation. *Evol. Comput.* **1993**, *1*, 1–23.

[77] Syswerda, G. Uniform crossover in genetic algorithms. In *Proc. ICGA-3*, 1989; pp. 2–9.

[78] Eiben, A.E.; Smit, S.K. Parameter tuning for evolutionary algorithms. In *GECCO 2009*, 2009; pp. 2715–2722.

[79] Coello Coello, C.A.; Lamont, G.B.; Van Veldhuizen, D.A. *Evolutionary Algorithms for Solving Multi-Objective Problems*, 2nd ed.; Springer, **2007**.

[80] Rothlauf, F. *Representations for Genetic and Evolutionary Algorithms*, 2nd ed.; Springer, **2006**.

[81] Thierens, D.; Goldberg, D.E. Convergence models of GA selection schemes. In *PPSN II*, 1992; pp. 119–129.

[82] Herrera, F.; Lozano, M.; Verdegay, J.L. Operators and tools for real-coded GAs. *Artif. Intell. Rev.* **2003**, *12*, 265–319.

[83] Deb, K.; Agrawal, R.B. Simulated binary crossover. *Complex Syst.* **1995**, *9*, 115–148.

[84] Oliver, I.M.; Smith, D.J.; Holland, J.R. Permutation crossover study. In *Proc. ICGA-2*, 1987; pp. 224–230.

[85] Michalewicz, Z.; Schoenauer, M. Evolutionary algorithms for constrained optimisation. *Evol. Comput.* **1996**, *4*, 1–32.

[86] Deb, K.; Pratap, A.; Agarwal, S.; Meyarivan, T. NSGA-II. *IEEE Trans. Evol. Comput.* **2002**, *6*, 182–197.

[87] Jin, Y. Surrogate-assisted evolutionary computation. *Swarm Evol. Comput.* **2011**, *1*, 61–70.

[88] Smith, R.E.; Tate, D.M. GA optimisation with a penalty function. In *Proc. ICGA-5*, 1993; pp. 499–505.

[89] Branke, J. *Evolutionary Optimization in Dynamic Environments*. Springer, **2001**.

[90] Grefenstette, J.J. Genetic algorithms for changing environments. In *PPSN II*, 1992; pp. 137–144.

[91] Forrester, A.I.J.; Sóbester, A.; Keane, A.J. *Engineering Design via Surrogate Modelling*. Wiley, **2008**.

[92] Beyer, H.G.; Schwefel, H.P. Evolution strategies — a comprehensive introduction. *Nat. Comput.* **2002**, *1*, 3–52.

[93] Sutton, R.S.; Precup, D.; Singh, S. Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning. *Artif. Intell.* **1999**, *112*, 181–211.

[94] Ng, A.Y.; Russell, S. Algorithms for Inverse Reinforcement Learning. In Proceedings of the *17th International Conference on Machine Learning (ICML)*, Stanford, CA, USA, 29 June–2 July 2000; pp. 663–670.

[95] Kumar, A.; Zhou, A.; Tucker, G.; Levine, S. Conservative Q-Learning for Offline Reinforcement Learning. In Proceedings of the *34th Conference on Neural Information Processing Systems (NeurIPS)*, Virtual Event, 6–12 December 2020; pp. 1–23.

[96] Mokhtari, M.; Ganti, S. Fog Node Resource Allocation in Fog-Cloud Systems Using Reinforcement Learning. *Proceedings of the IEEE Consumer Communications and Networking Conference (CCNC)*, 2025, pp. 123–128. DOI: 10.1109/CCNC.2025.1234567.

[97] Mokhtari, M.; Ganti, S. Resource Allocation in Multi-Fog/Cloud Systems Using a Hybrid Genetic Algorithm–Reinforcement Learning Approach. *Proceedings of the IEEE International Conference on Cloud Computing (CLOUD)*, 2025.

[98] Mokhtari, M.; Ganti, S. Resource Allocation in Multi-Fog/Cloud Systems Using Reinforcement Learning. Submitted to the *IEEE International Conference on Big Data and Cloud Computing (BDCloud)*, 2025.

[99] Choppara, P.; Mangalampalli, S.S. A Hybrid Task Scheduling Technique in Fog Computing Using Fuzzy Logic and Deep Reinforcement Learning. *IEEE Access* **2024**, *12*, 176363–176388. DOI: 10.1109/ACCESS.2024.3505546.

[100] Ling, C.; Peng, K.; Wang, S.; Xu, X.; Leung, V.C.M. A Multi-Agent DRL-Based Computation Offloading and Resource Allocation Method With Attention Mechanism in MEC-Enabled IIoT. *IEEE Trans. Serv. Comput.* **2024**, *17*(6), 3037–3051.

[101] Roshan, R.; Matam, R.; Mukherjee, M.; Lloret, J.; Tripathy, S. A Secure Task-Offloading Framework for Cooperative Fog Computing Environment. In Proceedings of the *2020 IEEE Global Communications Conference (GLOBECOM)*, **2020**; IEEE. DOI: 10.1109/GLOBECOM42002.2020.9322509.

[102] Kallela, A.; Rekik, M.; Khemakhem, M. DRL4HFC: Deep Reinforcement Learning for Container-Based Scheduling in Hybrid Fog/Cloud System. In Proceedings of the *16th International Conference on Agents and Artificial Intelligence (ICAART)*, **2024**, Volume 2; SciTePress, pp. 231–242.

[103] Na, M.; Cho, S.; Solat, F.; Na, T.; Lee, J. Energy-Efficient Hybrid Federated and Centralized Learning for Edge-Based Wireless Traffic Prediction in Aerial Networks. *IEEE Access* **2024**, *12*, 130983–130994. DOI: 10.1109/ACCESS.2024.3458089.

[104] Andong, F.J.E.N.; Min, Q. Federated Multi-Agent Reinforcement Learning for Privacy-Preserving and Energy-Aware Resource Management in 6G Edge Networks. *arXiv preprint* **2025**, arXiv:2509.10163.

[105] Mali, S.; Zeng, F.; Adhikari, D.; Ullah, I.; Al-Khasawneh, M.A.; Alfarraj, O.; Alblehai, F. Federated Reinforcement Learning-Based Dynamic Resource Allocation and Task Scheduling in Edge for IoT Applications. *Sensors* **2025**, *25*(7), 2197. DOI: 10.3390/s25072197.

[106] Nucci, F.; Papadia, G. Hybrid Genetic Algorithm and Deep Reinforcement Learning Framework for IoT-Enabled Healthcare Equipment Maintenance Scheduling. *Electronics* **2025**, *14*(21), 4160. DOI: 10.3390/electronics14214160.

[107] Konidaris, D.; Soumplis, P.; Varvarigos, A.; Kokkinos, P. Intent-Based Resource Allocation in Edge and Cloud Computing Using Reinforcement Learning. *Algorithms* **2025**, *18*(10), 627. DOI: 10.3390/a18100627.

[108] Wang, Z.; Goudarzi, M.; Buyya, R. ReinFog: A Deep Reinforcement Learning Empowered Framework for Resource Management in Edge and Cloud Computing Environments. *J. Netw. Comput. Appl.* **2025**, *242*, 104250.

[109] Wang, Y.; Xue, H.; Zhou, M. A Digital Twin-Assisted VEC Intelligent Task Offloading Approach. *Electronics* **2025**, *14*(17), 3444. DOI: 10.3390/electronics14173444.

[110] Ghafari, R.; Mansouri, N. PGA: A Priority-Aware Genetic Algorithm for Task Scheduling in Heterogeneous Fog-Cloud Computing. *IEEE Access*, **2021**, 9, 1–18.

[111] Saad, M.; Enam, R.N.; Qureshi, R. Optimizing Multi-Objective Task Scheduling in Fog Computing with GA-PSO Algorithm for Big Data Application. *Frontiers in Big Data*, **2024**, 7, 1358486.

[112] Sahoo, K.S.; Tiwary, M.; Sahoo, B. EMCS: An Energy-Efficient Makespan Cost-Aware Scheduling Algorithm Using Evolutionary Learning Approach for Cloud-Fog-Based IoT Applications. *Sustainability*, **2022**, 14(22), 15096.

[113] Kumar, N.J.; Singh, D.P. Efficient Cloud Resource Allocation Using a Hybrid Approach of Gradient Descent and Genetic Bee Colony Optimization (HGD-GBCO). In Proceedings of the *2025 3rd International Conference on Data Science*, **2025**.

[114] Guerrero, C.; Lera, I.; Juiz, C. Genetic-based Optimization in Fog Computing: Current Trends and Research Opportunities. *Swarm and Evolutionary Computation*, **2021**, 67, 100987.

Appendix A

Worked Example: Genetic Algorithm for the TSP

A.1 Example

Problem Description

The **Traveling Salesman Problem (TSP)** is a classic combinatorial optimization problem where the objective is to find the shortest possible route that visits each city once and returns to the origin. Given a set of n cities and a distance matrix $D = [d_{ij}]$, the goal is to determine a permutation $\pi = (\pi_1, \pi_2, ..., \pi_n)$ that minimizes the total travel cost:

$$\text{Total Distance} = d_{\pi_n \pi_1} + \sum_{i=1}^{n-1} d_{\pi_i \pi_{i+1}}$$

Encoding Strategy

We use **permutation encoding**, where each chromosome is a permutation of city indices. For example, a chromosome representing a tour of 6 cities might look like:

$$\text{Chromosome} = [3, 1, 6, 4, 2, 5]$$

This chromosome denotes a tour starting at city 3, going through cities 1, 6, 4, 2, and ending at city 5 before returning to city 3.

Step 2: Initialization

We begin by generating an initial population of random tours. Each individual in the population is a unique permutation of the cities.

Example: Assume we have $n = 6$ cities labeled 1 through 6. Let the population size be $P = 4$. We randomly generate four permutations as the initial population:

- Individual 1: [1, 4, 2, 6, 3, 5]
- Individual 2: [3, 1, 5, 2, 6, 4]
- Individual 3: [2, 6, 3, 1, 5, 4]
- Individual 4: [5, 4, 1, 6, 2, 3]

Each chromosome corresponds to a complete tour that starts from the first city in the list, visits each subsequent city in order, and returns to the first city to complete the cycle.

Assumption: The distance matrix $D = [d_{ij}]$ is symmetric and known. For the sake of this example, we define the following matrix:

$$D = \begin{bmatrix} 0 & 3 & 1 & 5 & 8 & 6 \\ 3 & 0 & 6 & 7 & 9 & 2 \\ 1 & 6 & 0 & 4 & 3 & 7 \\ 5 & 7 & 4 & 0 & 6 & 5 \\ 8 & 9 & 3 & 6 & 0 & 2 \\ 6 & 2 & 7 & 5 & 2 & 0 \end{bmatrix}$$

This matrix represents the pairwise distances between the 6 cities.

Step 3: Fitness Evaluation

The fitness of each individual is evaluated based on the total tour distance. Since we aim to minimize the travel distance in the TSP, we define the fitness function f as the inverse of the total distance:

$$f(\text{tour}) = \frac{1}{\text{Total Distance of the Tour}}$$

Using the distance matrix D provided earlier, we compute the total distance for each individual in the population.

- **Individual 1: [1, 4, 2, 6, 3, 5]**

$$d_{1,4} + d_{4,2} + d_{2,6} + d_{6,3} + d_{3,5} + d_{5,1} = 5 + 7 + 2 + 7 + 3 + 8 = 32$$

Fitness: $f_1 = \frac{1}{32} \approx 0.0313$

- **Individual 2: [3, 1, 5, 2, 6, 4]**

$$d_{3,1} + d_{1,5} + d_{5,2} + d_{2,6} + d_{6,4} + d_{4,3} = 1 + 8 + 9 + 2 + 5 + 4 = 29$$

Fitness: $f_2 = \frac{1}{29} \approx 0.0345$

- **Individual 3: [2, 6, 3, 1, 5, 4]**

$$d_{2,6} + d_{6,3} + d_{3,1} + d_{1,5} + d_{5,4} + d_{4,2} = 2 + 7 + 1 + 8 + 6 + 7 = 31$$

Fitness: $f_3 = \frac{1}{31} \approx 0.0323$

- **Individual 4: [5, 4, 1, 6, 2, 3]**

$$d_{5,4} + d_{4,1} + d_{1,6} + d_{6,2} + d_{2,3} + d_{3,5} = 6 + 5 + 6 + 2 + 6 + 3 = 28$$

Fitness: $f_4 = \frac{1}{28} \approx 0.0357$

Observation: Based on this distance matrix, Individual 4 has the highest fitness (shortest total travel distance) among the four individuals, and is likely to be favored during the selection process.

Step 4: Selection

In this step, we select individuals for reproduction based on their fitness values. We use **roulette wheel selection** (fitness proportionate selection), where the probability of selecting an individual is proportional to its fitness:

$$P(i) = \frac{f_i}{\sum_{j=1}^{4} f_j}$$

First, compute the sum of fitness values:

$$\sum f_j = 0.0313 + 0.0345 + 0.0323 + 0.0357 = 0.1338$$

Now, compute the selection probability for each individual:

- $P_1 = \frac{0.0313}{0.1338} \approx 0.2339$
- $P_2 = \frac{0.0345}{0.1338} \approx 0.2579$
- $P_3 = \frac{0.0323}{0.1338} \approx 0.2414$

- $P_4 = \frac{0.0357}{0.1338} \approx 0.2668$

Using a simulated roulette wheel spin (i.e., drawing random numbers from $[0, 1)$), we may select the following parents (example outcome):

- Parent 1: Individual 4
- Parent 2: Individual 2
- Parent 3: Individual 4
- Parent 4: Individual 1

These selected individuals will be used for the crossover step.

Step 5: Crossover

We apply the **Order Crossover (OX)** operator, which is commonly used in permutation-based problems such as the Traveling Salesman Problem (TSP).

Assume we select two parents from the selected pool (consistent with $n = 6$):

- **Parent A**: [1, 2, 3, 4, 5, 6]
- **Parent B**: [3, 1, 4, 6, 5, 2]

Step 1: Select Crossover Points

Randomly choose two crossover points. For example, positions 2 and 4.

$$\text{Crossover Points: } \boxed{2} \quad \boxed{4}$$

Step 2: Copy Subsequence from Parent A to Offspring

The genes from position 2 to 4 in Parent A are copied into the offspring:

$$\text{Offspring (initial)} = [\_, \boxed{2, 3, 4}, \_, \_]$$

Step 3: Fill Remaining Positions from Parent B

We take the remaining genes from Parent B in order, skipping duplicates:

- Parent B: [3, 1, 4, 6, 5, 2]
- Remove 2, 3, and 4 $\Rightarrow$ Remaining: [1, 6, 5]
- Fill the empty positions (1, 5, 6) with [1, 6, 5] in order

$$\text{Final Offspring } 1 = [1,2,3,4,6,5]$$

Repeat for second offspring (swap parents):

- **Parent A**: [3, 1, 4, 6, 5, 2]
- **Parent B**: [1, 2, 3, 4, 5, 6]

Apply the same crossover points (positions 2 and 4):

$$\text{Subsequence from Parent A} : [1,4,6] \Rightarrow \text{Initial Offspring} = [\_,1,4,6,\_,\_]$$

Remaining genes from Parent B: [1, 2, 3, 4, 5, 6] minus [1, 4, 6] $\Rightarrow$ [2, 3, 5]

$$\text{Final Offspring } 2 = [2,1,4,6,3,5]$$

New Population After Crossover:

- Offspring 1: [1, 2, 3, 4, 6, 5]
- Offspring 2: [2, 1, 4, 6, 3, 5]

Step 6: Mutation

To maintain genetic diversity and avoid premature convergence, we apply a mutation operator with a small probability p_m (e.g., 0.1).

For permutation-encoded individuals, we use **swap mutation**, which randomly selects two cities and swaps their positions.

Assume Offspring Before Mutation:

- Offspring 1: [1, 2, 3, 4, 6, 5]
- Offspring 2: [2, 1, 4, 6, 3, 5]

Apply Mutation:

- **Offspring 1**: Suppose no mutation occurs (with 90% probability).
- **Offspring 2**: Mutation occurs.

  Randomly select two positions to swap, e.g., positions 2 and 6:

$$[2,\underline{1},4,6,3,\underline{5}] \Rightarrow [2,\boxed{5},4,6,3,\boxed{1}]$$

- **Mutated Offspring 2**: [2, 5, 4, 6, 3, 1]

New Population After Mutation:

- Offspring 1: [1, 2, 3, 4, 6, 5]
- Offspring 2: [2, 5, 4, 6, 3, 1]

The new individuals are now ready to participate in the next generation. If elitism is used, the best individual from the previous generation is also included in the new population.

Step 7: Replacement and Elitism

Once the offspring are generated through crossover and mutation, the next step is to decide how the new population will be formed. In this example, we apply **elitist generational replacement**, where the best individual from the previous generation is carried over to the next generation without change.

Previous Generation (after Evaluation):

- Individual A: [1, 2, 3, 4, 5, 6], Total Distance = 10 (best)
- Individual B: [2, 1, 4, 6, 3, 5], Total Distance = 15

Offspring (after Mutation):

- Offspring 1: [1, 2, 3, 4, 5, 6], Total Distance = 10
- Offspring 2: [2, 3, 4, 5, 6, 1], Total Distance = 18

Elitism Strategy:

- Preserve the best individual from the previous generation: [1, 2, 3, 4, 5, 6]
- Select the best offspring to fill the remaining slots.

New Generation:

- Individual 1: [1, 2, 3, 4, 5, 6] (from previous generation)
- Individual 2: [1, 2, 3, 4, 5, 6] (offspring 1)

This elitist strategy guarantees that the best solution found so far is never lost, thereby promoting steady progress toward optimality while preserving high-quality solutions.

Step 8: Termination and Convergence

Termination criteria determine when the genetic algorithm should stop evolving the population. In practice, this is based on predefined conditions such as:

- A fixed number of generations
- No improvement in the best fitness over a certain number of generations (fitness stagnation)
- Reaching a fitness threshold (acceptable solution)

In our TSP example, we apply the following termination criteria:

1. Maximum number of generations: $G_{\max} = 50$
2. Or early stopping if best fitness does not improve for 10 consecutive generations

Observed Convergence:

After several generations (say, 23 generations), we observe that the population converges to the following best individual:

- Best Chromosome: [1, 2, 3, 4, 5, 6]
- Total Distance: 10

Convergence Behavior:

- The algorithm maintains diversity early on through crossover and mutation.
- Over time, fitter individuals dominate the population.
- Elitism helps preserve the best-known solution across generations.
- Eventually, all individuals converge to the optimal or near-optimal route.

Termination Condition: No improvement in best fitness for 10 generations $\Rightarrow$ algorithm stops.

This example illustrates a typical convergence pattern in GAs: exploration in the early generations, followed by exploitation and population convergence. The genetic algorithm successfully identifies the optimal path for the TSP in this example.

A.1.1 Summary Table of Fitness Evaluation

To provide a concise overview of the initial population, their fitness values, and selection probabilities, Table A.1 summarizes the results of Step 3 (Fitness Evaluation) and Step 4 (Selection).

This table provides a consolidated view of the evaluation and selection stages. It clearly shows how shorter distances correspond to higher fitness and thus to higher selection probabilities, aligning with the roulette wheel selection mechanism.

Table A.1: Summary of Initial Population Evaluation and Selection Probabilities for TSP

Individual	**Chromosome (Tour)**	**Total Distance**	**Fitness**	**Selection Probability P_i**
1	[1, 4, 2, 6, 3, 5]	32	0.0313	0.2339
2	[3, 1, 5, 2, 6, 4]	29	0.0345	0.2579
3	[2, 6, 3, 1, 5, 4]	31	0.0323	0.2414
4	[5, 4, 1, 6, 2, 3]	28	0.0357	0.2668