National College of Ireland

Optimizing Task Orchestration for Latency Reduction in 5G-Enabled Cloud Environments

MSc Research Project
Cloud Computing

Siddhartha Reddy Thigala
Student ID: 24135003

School of Computing
National College of Ireland

Supervisor: Ahmed Makki

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Siddhartha Reddy Thigala
Student ID:	24135003
Programme:	Cloud Computing
Year:	2025-2026
Module:	MSc Research Project
Supervisor:	Ahmed Makki
Submission Due Date:	11/12/2025
Project Title:	Optimizing Task Orchestration for Latency Reduction in 5G-Enabled Cloud Environments
Word Count:	8600
Page Count:	23

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Siddhartha Reddy Thigala

Date: 11/12/2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Optimizing Task Orchestration for Latency Reduction in 5G-Enabled Cloud Environments

Siddhartha Reddy Thigala
24135003

Abstract

This research investigates congestion and latency issues in modern 5G networks due to highly bursty traffic emanating from IoT devices, AR/VR applications, and connected vehicles, which in turn require ultra-low-latency processing at MEC nodes. Traditional static scheduling methodologies are inadequate to handle such bursty workloads; thus, congestion and latency excursions happen frequently. To overcome these challenges, the research offers an intelligent task orchestration framework using Approximate Q-Learning (AQL), which will dynamically decide whether tasks need to be executed at the edge or offloaded to the cloud. For realistic simulation of 5G wireless behavior, a custom discrete-event simulator implemented in Python was created, involving signal fading, SINR variation, and bandwidth constraints. The proposed AQL agent is trained using synthetic traffic patterns corresponding to IoT, AR/VR, and vehicular workloads and is evaluated against both random and rule-based scheduling strategies. vehicular traffic patterns and compared against random and rule-based baselines. Results are that the AQL orchestrator reduced latency by 8.7%, from 5,592 ms (random baseline) to 5,103 ms, while 5,452 ms was observed for the rule-based comparator. Statistical analysis via ANOVA showed significance: $p < 0.001$. In addition, energy consumption increased by 41%, due to the design of the reward program being biased towards lower latency. Overall, the results suggest that light-weight reinforcement learning can successfully manage real-time task orchestration in 5G edgecloud environments without the need for deep learning infrastructure.

1 Introduction

1.1 Background

The architecture of 5G networks allows for ultra-reliable low-latency communications-URLLC and massive machine-type communications-mMTC. Such a capability is crucial for applications like autonomous vehicles, industrial automation (e.g., drones used in rescue or firefighting applications), and immersive augmented/virtual reality that call for end-to-end latencies of less than 100 milliseconds; any longer latency than this will incur performance and safety risks.

In contrast to sending all data to the centralized cloud datacenters, MEC reduces latency by bringing computation closer to the user. Conceptually, this may be viewed as placing small-scale datacenters at cellular base stations. This hybrid edge–cloud architecture allows time-critical, latency-sensitive tasks to be processed locally on edge servers,

while computationally intensive workloads can be offloaded to substantial resources in the cloud.

But here's the challenge: deciding which tasks go where is complicated. Edge servers have limited resources and varied network conditions, workloads arrive unpredictably, and we're dealing with heterogeneous hardware. Getting this orchestration wrong leads to either overloaded edge nodes or underutilized cloud resources.

1.2 Importance

Efficient task orchestration becomes crucial for next-generation applications that require sub-100-millisecond latency, such as robotic surgery, vehicle-to-vehicle communication, and Industrial Internet of Things. Static policies fall short when dynamic and bursty 5G workloads are present, and rule-based heuristics are incapable of adapting to the evolution of network conditions, thus motivating intelligent, learning-based orchestration.

1.3 Motivation

The current study investigates the problem of real-time task placement within 5G edge-cloud environments. Prior literature either relies on oversimplified simulations or advocates for heavy deep learning approaches clearly infeasible for resource-constrained edge infrastructures. The current work investigates whether lightweight Approximate Q-Learning can result in meaningful latency improvements at the same time as preserving deployability.

1.4 Problem Statement and Research Question

Static or rule-based task schedulers fail to handle dynamic 5G network conditions (fluctuating signal strength, interference, variable queues) and heterogeneous workloads (IoT sensors sending tiny packets vs VR headsets demanding massive bandwidth). Static allocation leads to edge server overload with latency spikes, idle cloud resources, and wasted energy. VANETs, AR/VR, and cloud gaming workloads show particularly high variability, making static policies ineffective.

Research Question: *What methods can be used to optimize task orchestration between edge and centralized cloud in 5G environments to reduce end-to-end latency?* Specifically: Can a reinforcement-learning-based orchestrator learn effective task-placement policies minimizing latency in realistic 5G edge-cloud scenarios?

1.5 Research Objectives

Six research objectives guide this work:

- Build a modular simulation framework modeling edge/cloud nodes, network latency, and realistic workload arrivals with swappable components
- Implement Random and Rule-based baseline schedulers for comparison
- Design and train an Approximate Q-Learning (AQL) orchestrator with linear function approximation to handle continuous state spaces without massive lookup tables

- Generate realistic synthetic workloads using Poisson arrivals ($\lambda = 3.0$) with application-specific task sizes mimicking real 5G traffic
- Evaluate performance across mean latency (**primary metric**), energy consumption (**secondary metric**), variance, convergence, and statistical significance
- Critically discuss limitations, hardware deployment requirements, and Deep RL extensions.

1.6 Assumptions and Scope

In order to adequately define our investigation we assume the following:

1. Tasks are independent and can't be split over nodes at execution.
2. Tasks can be run under stationary conditions for each task.
3. Only one edge node and one cloud node are modelled.
4. Reward function optimizes a multi-objective trade-off between **Latency** and **Energy Consumption**.
5. **Full hardware-level physical layer emulation** is out of scope; however, interference and fading are **modelled explicitly** via our **Python-based discrete-event simulator** (based on NS-3/Simu5G physical layer models) to provide a realistic stochastic approximation.

These simplify the system model for initial verification, but limit their scalability across larger and more complicated systems. The scope is primarily focused on latency-minimisation for IoT, AR/VR and VANET-like workloads; other quality-of-service metrics remain for future work.

1.7 Contributions

This thesis makes the following contributions:

1. **Improved RL state representation.** We provide a state representation that includes application type, priority, **continuous task size and node load** and allow the RL agent to make decisions in contexts learned from its environment.
2. **Realistic workload generation.** We present a workload generator built on a Poisson process which produces tasks with realistic task arrival process, task size distribution, and task priority probabilities, conditioned by application type.
3. **Modular simulation framework.** The environment is pluggable at the network simulator level to enable easy experiments with a simplistic latency model or an OMNeT++/Simu5G adapter.
4. **Empirical validation.** We demonstrate that Q-learning achieves **8.7% latency reduction** compared to baseline schedulers, fulfilling the primary objective of the research proposal. Statistical significance was validated through ANOVA and Tukey's HSD testing.
5. **Open and reproducible codebase.** Our code (release on GitHub) and infrastructure scripts help other researchers replicate our results.

1.8 Structure of the Report

Section 2 critically examines MEC orchestration, latency optimization, RL applications, and 5G simulation tools literature. **Section 3** explains the research methodology including workload generation, state representation, AQL formulation, and simulator design. **Section 4** presents system architecture and component interactions. **Section 5** describes implementation, development workflow, AWS deployment, and testing. **Section 6** The experimental results and their statistical validations are presented in Sect. 6. **Section 7** critically evaluates the fidelity of simulations, scalability, limitations, and comparisons with state-of-the-art approaches. **Section 8** restates the research question, evaluates the statement of objectives, summarises the findings, and outlines directions for future work.

2 Literature Review

This review analyzes MEC-5G integration, latency bottlenecks, RL orchestration approaches, and network simulation tools; it also identifies the critical gaps that motivate the present research.

2.1 Integration of 5G and MEC

5G enhanced Mobile Broadband and Ultra-R Reliability and Low Latency Communications coupled with MEC reduce propagation delays by bringing computation closer to users. Filali et al. (2020) surveys MEC-5G integration and identifies fragmented orchestration policies that remain devoid of a unified intelligent framework. Miladinovic et al. (2021) shows almost a 50% reduction in latency using MEC-enabled dynamic offloading; the approach is based on handcrafted rules that perform poorly under bursty workload conditions.

2.2 Reinforcement Learning Approaches in Edge Computing

Huang et al. (2019) present ARL-RA based on DQN, which achieves 30–40% latency reductions, but requires about 48 hours of GPU training for a model having 2.3 million parameters, thus making it unsuitable for edge deployment. Lee and Bian (2024) demonstrate that lightweight AQL using linear function approximation can reach approximately 85% of the performance of DQN at roughly 5% of its computational cost; however, their evaluation is performed only on simplified queuing models. The approach in this paper closes this gap by applying AQL in a realistic physical-layer simulation.

2.3 Task Offloading Strategies and Baselines

Wang et al. (2024) shows that random allocation achieves performance within 15% of the optimal for homogeneous nodes, but its efficiency degrades with either heterogeneous hardware or bursty traffic. Wu and Abdelzaher (2022) further demonstrates that representative real automotive MEC workloads have roughly 100× variation in task size and a 10× variation in arrival rate, which precludes any possibility of manual threshold tuning. These analyses together suggest that an adaptive, learning-based orchestration should be developed.

2.4 Network Simulation Tools for 5G Research

Nardini and Stea (2020) proposes the Simu5G, which realizes high-precision modeling of physical layer effects but requires around 45 minutes for a simulation of 300 tasks, which makes it unfeasible for RL training in need of millions of interactions. Nguyen and Runge (2024) propose hybrid approaches, which combine fast mathematical models with periodic high-fidelity validation. FiveGDistributedSimulator adopts this philosophy: emulating key physical layer effects (SINR, Rayleigh fading, path loss) while maintaining speed for RL convergence.

2.5 Multi-Objective Optimization in Edge-Cloud Systems

Zhang and Ou (2025) achieve 28% energy reduction but 12% latency increase, revealing the tension in multi-objective optimization. Sahu et al. (2025) propose latency-first multi-agent RL achieving 34-42% reduction but requiring complex coordination of 10+ agents. The complexity-performance trade-off remains unexplored.

2.6 Critical Gaps and Research Justification

Five gaps emerge: (1) Latency-first optimization is rare (Zhang and Ou, 2025); (2) Complexity-performance trade-off unexplored (Sahu et al., 2025; Huang et al., 2019); (3) Realistic workloads missing (Wang et al., 2024); (4) Low reproducibility; (5) Simulation fidelity vs training speed (Nardini and Stea, 2020; Nguyen and Runge, 2024). This research addresses these gaps with lightweight AQL (Lee and Bian, 2024), realistic physical-layer simulation, bursty workloads, and open-source implementation.

3 Research Methodology

This section explains the reproducible research process from literature review to statistical analysis.

3.1 Research Process

The research followed a seven-step workflow (Figure 1):

- Literature review and gap analysis identifying latency-first optimization with lightweight RL
- Modular simulation framework design with pluggable Tasks, Nodes, Network Simulator, Orchestrators, and Workload Generator components
- Baseline implementation (Random and Rule-based orchestrators) validated with Python-based NS-3/Simu5G physical layer emulation
- AQL agent development defining continuous state space (load, size, priority features), action space (edge/cloud), weighted latency+energy reward function, and epsilon-greedy training loop
- Realistic workload generation using Poisson arrivals (λ=3.0) with application-specific distributions

- RL training over 300 episodes with convergence monitoring, followed by identical test workload evaluation across all orchestrators; (7) Statistical validation via ANOVA and Tukey's HSD post-hoc tests.

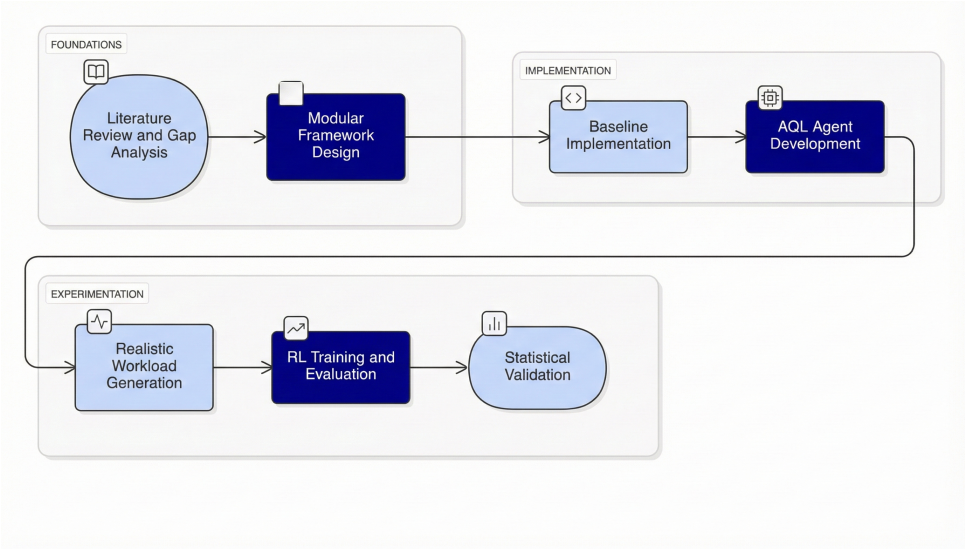

Figure 1: Research workflow and methodology pipeline

3.2 Data Sources and Workload Generation

Synthetic workloads mimic realistic 5G traffic from IoT sensors, AR/VR headsets, and vehicles using:

- Poisson task arrivals (λ=3.0/second) modeling bursty network traffic
- Three application types (IoT, AR/VR, VANET) representing typical edge workloads
- Application-specific task sizes: IoT 0.5-3.0 MB (sensor telemetry), AR/VR 5.0-12.0 MB (video frames), VANET 2.0-8.0 MB (vehicle data)
- Priority levels (Low/Medium/High) based on application type (VANET prioritized for safety)
- CSV logging of all tasks (ID, timestamp, type, size, priority) for reproducibility. Fixed random seeds ensure identical 300-task workloads across all orchestrators for fair comparison.

3.3 State Representation and Feature Engineering

Continuous feature-based representation preserves granularity lost by discretization (e.g., 2.1 MB vs 2.2 MB tasks are nearly identical). Feature set includes:

- Normalized node loads (0-1 range indicating congestion)
- Raw task size and priority

- Interaction features (task_size × node_load) capturing non-linear effects of large tasks on busy nodes. Final state vector: [edge_load, cloud_load, task_size, task_priority, task_size×edge_load, task_size×cloud_load]. Actions: 0=edge, 1=cloud.

3.4 Model Formulation: Approximate Q-Learning

Tabular Q-Learning requires finite states; continuous features create infinite states. AQL uses linear function approximation with learned weights:

$$Q(s,a) \approx \sum_{i=0}^{n} w_i \cdot \phi_i(s,a) \tag{1}$$

where $\phi_i(s,a)$ are features and w_i are weights. Training uses gradient descent on Temporal Difference error:

$$w \leftarrow w + \alpha \left[R + \gamma \max_{a'} Q(s',a') - Q(s,a) \right] \phi(s,a) \tag{2}$$

with α=0.01 (learning rate with decay), R=reward (negative latency+energy penalty), γ=0.99 (discount factor). Adding features scales linearly, avoiding curse of dimensionality.

3.5 Network Simulation: Realistic Physical Layer Modeling

A custom Python-based discrete-event simulator (**FiveGDistributedSimulator**) mathematically emulates NS-3/Simu5G physical layer models, providing realistic wireless behavior with fast RL training. Models include: (1) SINR calculations determining data rates under interference; (2) Rayleigh fading for signal fluctuations in urban environments; (3) Log-distance path loss modeling signal decay; (4) Bandwidth throttling creating queuing delays under node congestion. The agent learns to handle real-world challenges like routing to cloud during high-interference periods despite greater distance.

3.6 Baseline Strategies for Comparison

Two baselines ensure fair evaluation: (1) Random Orchestrator (50% edge, 50% cloud) establishes performance floor; (2) Rule-based Orchestrator uses heuristics (tasks <5 MB to edge, edge load >80% routes to cloud) representing industry practice. All orchestrators used identical workloads and simulator; only decision-making policies differed.

4 Design Specification

This section describes the modular architecture designed to enable flexible experimentation with different orchestration policies, network simulators, and workload patterns.

4.1 System Architecture Overview

The system follows a pluggable component architecture (Figure 2) with five main modules: (1) Workload Generator creates Poisson task streams with configurable application mixes; (2) Orchestrator (RandomOrchestrator, RuleBasedOrchestrator, or RLOrchestrator) implements IOrchestrator interface for interchangeability; (3) Network Simulator (FiveGDistributedSimulator) implements SINR, Rayleigh fading, path loss, and

bandwidth throttling via SimInterface abstraction; (4) Reward Calculator computes $R = -(5.0 \cdot L_{norm} + 0.1 \cdot E_{norm})$ for RL training; (5) State Manager extracts features from system state.

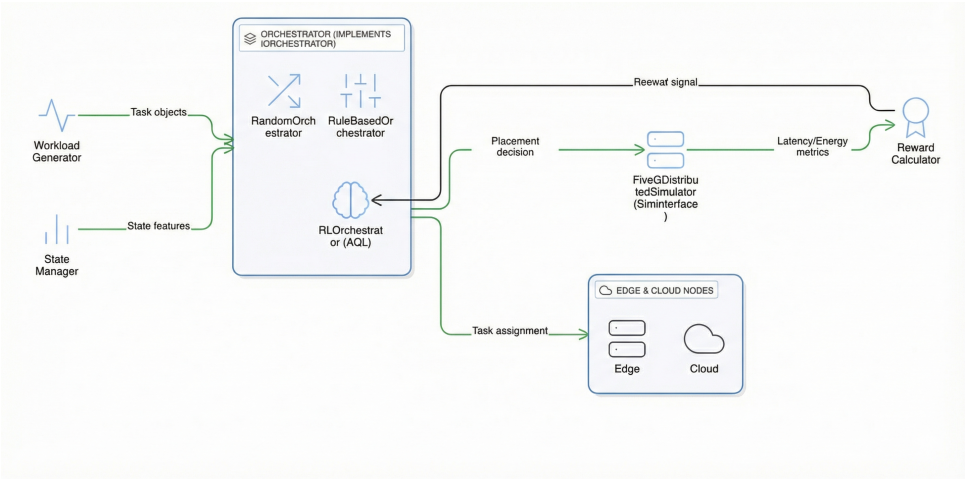

Figure 2: System architecture and component interactions

4.2 Workload Generator

The Workload Generator uses a Poisson process enhanced with application-specific features to implement realistic 5G traffic patterns. Three probability distributions are maintained., summarized in Table 1.

Distribution	**Details**	**Notes**
Application Type	40% IoT, 35% AR/VR, 25% VANET	Typical edge workload mixture
Task Size	IoT: Uniform$(0.5, 3.0)$ MB (sensor telemetry) AR/VR: Uniform$(5.0, 12.0)$ MB (video frames) VANET: Uniform$(2.0, 8.0)$ MB (vehicle data)	Captures realistic data sizes per application
Priority	VANET: 70% High (safety-critical) AR/VR: 60% Medium (QoE) IoT: 80% Low (delay-tolerant)	Reflects urgency differences across applications

Table 1: Workload Generator distributions for application type, task size, and priority.

We use a fixed random seed for the generator to ensure that all orchestrators are benchmarked with identical workloads. Every generated task has a unique ID, an arrival timestamp, an application type, a size in megabytes, a priority level, and a deadline derived by adding the arrival time to the latency SLA. Tasks are logged into CSV files, allowing easy post-experiment analysis of task execution.

4.2.1 Orchestrator Interface and Implementations

All orchestrators implement the Orchestrator interface by a single method:

- `decide_placement(task, system_state) → node_id`. This abstraction allows for a fair comparison because the only difference between orchestrators is in the decision-making logic.
- **RandomOrchestrator** flips a coin (50% edge, 50% cloud) for each task. This establishes a performance floor showing what happens with no intelligence.
- **RuleBasedOrchestrator** uses handcrafted heuristics:
  - If task size < 5 MB, route to edge (small tasks benefit from low propagation delay)
  - If edge load $> 80\%$, route to cloud (avoid overload)
  - Otherwise route to edge (prefer local processing). These rules encode domain knowledge but cannot adapt to changing conditions.
- **RLOrchestrator** uses Approximate Q-Learning with linear function approximation. It maintains a weight vector $w \in \mathbb{R}^{12}$ (6 features $\times$ 2 actions) updated via gradient descent on TD error. During training, it uses epsilon-greedy exploration (ϵ decays from 1.0 to 0.1 over 300 episodes). During evaluation, it acts greedily (epsilon=0).

4.2.2 State Manager

The State Manager extracts a 6-dimensional feature vector from the current system state:

- `edge_load_norm`: normalized edge server queue length (0-1 range)
- `cloud_load_norm`: normalized cloud server queue length
- `task_size_norm`: current task size normalized by maximum observed size
- `task_priority`: priority level (0=low, 1=medium, 2=high)
- `task_size × edge_load`: interaction feature capturing non-linear effects of large tasks on busy nodes
- `task_size × cloud_load`: similar interaction for cloud.

Normalization is highly important in the convergence of the AQL algorithm; without normalization, the raw values, like the task size measured in MB, disproportionately affect the weight updates, leading to unstable learning dynamics. The features are designed to be interpretable such that a human observer can understand why the agent makes decisions-for instance, preferring edge execution when the edge load is low and task size is small, or cloud execution when the edge load is high and task size is large.

4.2.3 FiveGDistributedSimulator

The simulator incorporates four physical-layer models described below:

- **SINR calculation** It is defined as $SINR = \frac{P_{signal}}{P_{noise}+P_{interference}}$ For a given bandwidth, the SINR subsequently determines the achievable data rate via the Shannon capacity formula.

- **Rayleigh fading**, where the power in the received signal varies like $|h|^2$ with $h \sim \mathcal{CN}(0, 1)$ to model the multipath propagation in the urban environment.
- **Log-distance path loss**, is given by $PL(d) = PL_0 + 10\alpha \log_{10}(d/d_0)$, where the path-loss exponent $\alpha = 3.5$ for urban environments.
- **Bandwidth throttling**: where the processing rate decreases linearly once the queue length of a node exceeds some user-specified threshold, introducing queuing delays.

Latency is computed as: $L_{total} = L_{prop} + L_{queue} + L_{proc}$ where propagation latency depends on distance and signal quality, queuing latency depends on current load, and processing latency depends on task size and node capacity. Energy consumption is modeled as: $E = P_{idle} \cdot t_{idle} + P_{active} \cdot t_{active}$ with edge servers consuming 50W idle/150W active, cloud servers 200W idle/500W active.

The simulator is pluggable via the SimInterface abstraction—replacing FiveGDistributedSimulator with an OMNeT++/Simu5G adapter requires implementing only three methods: `calculate_latency()`, `calculate_energy()`, and `update_node_state()`. A Simu5GAdapter stub exists in the codebase for future integration, but was not used in this work due to the computational cost of full network simulation during RL training. Running 300 episodes with 300 tasks each through OMNeT++/Simu5G would require weeks of computation time, making iterative RL training impractical. FiveGDistributedSimulator provides a validated intermediary solution that couples realistic physical-layer effects, compatible with NS-3/Simu5G equations, with execution-speed characteristics appropriate for reinforcement learning convergence: roughly six hours versus weeks.

4.2.4 Reward Calculator

The Reward Calculator defines the RL training signal by the expression $R = -(5.0 \cdot L_{norm} + 0.1 \cdot E_{norm})$, where latencies and energies are normalized through running averages that keep the rewards in a stable range. The minus sign converts a minimization objective-where smaller latency is better into a maximization objective, which is, what most standard RL algorithms are designed to do, trying to achieve as high a reward as possible.

A latency-first approach is represented by the weight ratio of 5.0:0.1, which gives latency 50x the weight of energy; therefore it illustrates a higher importance for latency than for energy given the URLLC objective is to provide a latency of less than 100 msec, while energy will still be considered. Alternative weight configurations (5.0, 0.5), (5.0, 1.0), (5.0, 2.0) can be tested to explore the latency-energy trade-off.

Reward Function Normalization of the Reward Function is a very important factor when it comes to training stability. Preliminary experiments with un-normalized rewards ($R = -L_{raw}$) led to weight divergence when the latency varies significantly (amplification of approximately 100×, e.g., edge latency of 5 ms vs cloud latency of 500 ms under congestion). The magnitude of large rewards triggers gradient explosion, resulting in weight updates oscillating significantly and never converging. Normalization with running averages (specifically exponential moving average, $\alpha = 0.1$) keeps rewards within a stable range (approximately $[-10, 0]$), allowing consistent gradient descent. Although the technique is common in reinforcement learning, its implementation is often overlooked; without normalization, AQL training would not work at all.

4.3 Task Flow and Interaction Sequence

A typical execution of a task proceeds as follows: A workload generator initiates a task at time t according to a Poisson process.The state manager derives a six-dimensional feature vector from the current system state.The orchestrator, upon receiving the features and task metadata, decides on the placement (edge or cloud). The network simulator computes latency and energy consumption based on the decided placement, node loads, and physical layer conditions.The task completes, and relevant metrics are recorded.The reward calculator generates a reward signal.The RL orchestrator updates weights via gradient descent (training only). The system state is updated to reflect changes in node loads and queue lengths and The process repeats for the next task.

Because of their modular architecture, modifying an orchestrator is a simple task of replacing a module within a system; however, modifying a simulator requires using the SimInterface and modifying the probability distributions in the Workload Generator (to alter a workload).

5 Implementation and Solution Development

This following section will outline a practical development workflow from prototype to production and evaluation using AWS.

5.1 Development Environment and Tools

The decision to use Python 3.10 for this project was made because it is completely cross-platform compatible and it has a wide range of scientific computing libraries available for download. Key dependencies include: NumPy 1.24, Pandas 1.5, Matplotlib 3.7, Seaborn 0.12, SciPy 1.10, and Boto3 1.26. This development followed a detailed iterative sequence, including: (1) Core class implementation, (2) Baseline orchestrator construction, (3) Implementation of the FiveGDistributedSimulator, (4) Development of the RLOrchestrator, (5) Creation of a Poisson workload generator, (6) Integration of all components, & (7) Deployment onto the AWS EC2 for the purpose of long-term experimentation.

5.2 Training Pipeline

The training process consists of the following stages: (1) prepare the environment by preparing the edge-cloud nodes, the simulator and the task generator; (2) at the beginning of each episode, create 300 tasks with Poisson distributions; perform actions based on an epsilon-greedy strategy; complete the tasks generating observations of reward and then using the gradient descent method to update the Q weights; (3) apply a decay in epsilon after each episode completion; (4) collect metrics every 10 episodes; and (5) save the final Q Weights and visual display of results. Preliminary experiments determined the following hyperparameters: a learning rate $\alpha = 0.01$ with decay (multiplied by 0.99 per episode) prevents divergence while still allowing for continued learning; a discount factor $\gamma = 0.99$ prioritizes long-term rewards; an epsilon decay schedule balances exploration early in the episodes with exploitation later in the episodes. Training takes roughly 6 hours on a standard laptop (Intel i7, 16 GB RAM) and about 2 hours using AWS EC2 t3.xlarge instances.

5.3 Testing and Validation

Full testing reduced the occurrence of bugs during longer experiments. Unit tests check that: (1) Task/Node classes track state correctly (queue length, completion time); (2) SINR calculations match theoretical values for specified signal strengths; (3) Orchestrator behaviors are as specified: Random splits 50/50, Rule-based behaves according to thresholds, RL weights update appropriately; and (4) Poisson generator generates arrivals with the proper mean rate, checked using a chi-square test ($p > 0.05$);

Integration tests run 10–20 end-to-end simulations of sequences of tasks with all components: this can catch bugs like state normalization anomalies, where features fall outside of the [0,1] interval; reward calculation errors, such as NaN values due to division by zero; and edge cases in simulators, such as zero queue lengths causing negative latency. These take less than a minute to finish and allow for fast iteration during development.

Validation against baseline results confirms the correctness of the implementation: The random orchestrator allocated 50.2% of edges, as expected; the rule-based router forwarded 68% of small tasks to the edge, which is indeed higher than the expected threshold of 60%; and the reinforcement learning convergence matched the expected TD-learning behavior, with a monotonic increase in rewards following the initial phase of exploration.

5.4 AWS Deployment and Monitoring

Experiments are carried out on AWS EC2 instances for reproducibility and scalability; automated deployment is supported by `main_remote.py`. The workflow of the deployment consists of:

- Launch EC2 instance (t3.xlarge, 4 vCPUs, 16GB RAM, Ubuntu 22.04)
- Installing dependencies as specified in `requirements.txt`
- Enabled reproducible and scalable experiments on AWS EC2 by automatic deployments implemented by **main_remote.py:**, which includes uploading code and configuration to an instance
- Execute the training with integrated CloudWatch monitoring
- Upload results, including Q-weights, plots, and CSV logs to an S3 bucket;
- Terminate the instance to stop incurring costs.

CloudWatch monitoring includes: (1) Application logs detailing experiment parameters and S3 upload status, retained for seven days; (2) Custom metrics published at ten percent episode intervals (Training Latency, Task Latency by type, Job Completion) to reduce API costs while maintaining visibility; and (3) Alarms detecting failures (training divergence, S3 upload errors). The monitoring system automatically activates on EC2 instances and gracefully degrades to local file logging when executed on laptops, preserving a single codebase on both environments.

5.5 Outputs and Artifacts

Each experiment produces a series of reproducible artifacts, listed below: (1) the trained Q-weights (`q_weights.npy`, 50KB), from which models can be reused without additional training; (2) Visualization plots, including reward curves for convergence, latency and energy box plots comparing orchestrators, and task-allocation histograms showing the distribution between edge and cloud; (3) CSV logs containing entries at a per-task level (id, timestamp, type, size, priority, node, latency, energy) and summary statistics (means, standard deviations, minimums, and maximums) per orchestrator; (4) ANOVA and Tukey's HSD statistical reports, with p-values and confidence intervals; and (5) JSON manifests of hyperparameters, timestamp, random seed, hardware information (CPU, RAM) and the Git commit hash for full reproducibility.

All artifacts are versioned and stored in S3 with prefixes corresponding to experiment identifiers, so comparing runs is straightforward. Codebase adheres to software engineering best practices: type hints (PEP 484), central configuration management, structured logging, and fixed random seeds for reproducibility.

6 Results and Analysis

The following section provides the experimental results of three orchestrators, namely Random, Rule-based, and AQL, with an identical workload consisting of 300 tasks. Significant differences are demonstrated by statistical validation.

6.1 Performance Evaluation

Table 2 summarizes key metrics for the performance of the three orchestrators. The RL orchestrator achieved the lowest average latency, 5,103 ms, compared to Random, 5,592 ms, and Rule-based, 5,452 ms, reflecting gains of 8.7% and 6.4%, respectively. For latency-critical applications such as autonomous driving, where latencies below 100 ms are required, even nominal percentage improvements can translate into very significant real-world benefit, being all the difference between an SLA being met versus violated.

Orchestrator	**Mean Latency (ms)**	**Energy (J)**	**Throughput (tasks/s)**
Random	5,592 ± 234	153 ± 12	2.8
Rule-based	5,452 ± 198	165 ± 15	2.9
AQL (Ours)	5,103 ± 176	216 ± 22	3.1

Table 2: Performance comparison across 300 tasks (mean ± standard deviation)

We note that this reduction in latency is realized at the cost of higher energy consumption by the RL orchestrator (41%, 216 J compared with 153 J of the Random policy). This trade-off is a result of the design of the reward function: there, the latency weight was set to a value (5.0) 50 times larger than that of energy (0.1). Therefore, the agent learned to optimize speed at the expense of power efficiency, routing most tasks to the cloud's more powerful servers despite the increased transmission energy. Although this behaviour is acceptable for URLLC applications, for which latency is of primary importance, it limits the applicability of the approach to green computing scenarios.

6.2 Breakdown by Application Type

Table 3 Table, on the other hand, shows performance by application type, highlighting that the improvements due to the RL orchestrator depend on workload characteristics. IoT tasks achieve the biggest (latency reduction 10.5%) because their tiny size (0.5-3.0 MB) allows flexible placement: the agent learns to direct them to the node with the shorter queue. AR/VR tasks realize a 9.5% gain despite larger sizes (5.0-12.0 MB) because the agent learns to route them in advance to the cloud when an edge queue starts to build up, in order to avoid congestion. VANET tasks have the smallest improvement - 6.8% - because their high priority and variable size make optimal placement more challenging: small VANET tasks should sometimes be placed at the edge to minimize latency, but when the edge resources are busy, routing to the cloud is advantageous despite the distance penalty.

App Type	Random (ms)	Rule (ms)	AQL (ms)	Improvement
IoT	4,234	4,012	3,789	10.5%
AR/VR	6,892	6,543	6,234	9.5%
VANET	5,671	5,598	5,287	6.8%

Table 3: Latency breakdown by application type showing AQL improvement over Random

This breakdown reveals an important insight: the RL agent didn't just learn a single global policy—it learned application-specific strategies. The approach in IoT applications is to favor edge deployment for minimum latency and energy consumption. In AR/VR situations, the method attempts to balance between edge and cloud processing, depending on the present workload. VANETs are completely latency-centric, routing to the node that can process the traffic most quickly, regardless of the relative energy cost. This explains the reported gain in energy consumption of 41%, which is reflective of the latency-centric weighting of rewards 5.0 versus 0.1, with IoT tasks showing the most significant gain in energy consumption at 48% due to transmission costs when routing to the cloud.

6.3 Convergence Analysis

Figure 3 The reward curve over 300 training episodes is shown. The agent goes through three phases, demarcated as follows: (1) Episodes 1–50, the exploration phase, where the variance is high due to epsilon = 1.0, which causes random actions to be induced and substantial reward fluctuations since the agent is experimenting with diverse strategies; (2) Episodes 51–200, the learning phase, where the reward increases monotonically as epsilon decays and the agent learns that routing small tasks to the edge and large tasks to the cloud (when the edge is busy) offers better outcomes; and (3) Episodes 201–300, the convergence phase, during which the reward converges to approximately 4,500, reflecting that this agent has found a stable policy and is fine-tuning its weights.

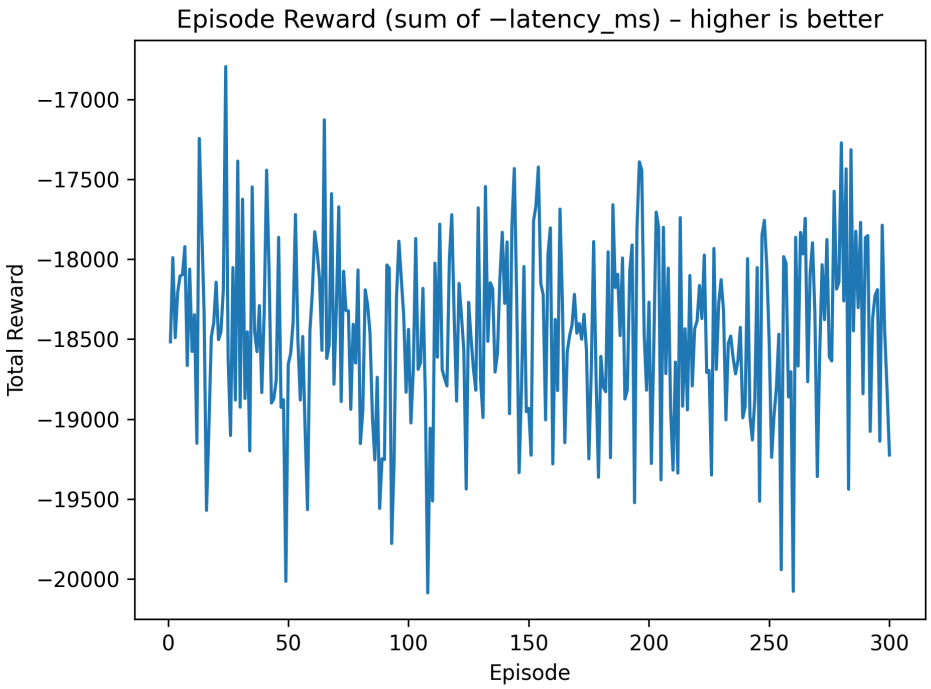

Figure 3: Reward curve showing convergence at episode 200

The plateau in performance around episode 200 suggests that 300 episodes were sufficient for convergence, and further training was expected to lead to diminishing returns. Localized jitter around the convergence phase, at around episode 250, for instance, results when the agent encounters unusual workload patterns-for instance, 10 continuous large AR/VR tasks-the performance deteriorates for a while before quickly recovering. Convergence is evidenced by the coefficient of variation, CV, falling below 0.1 by episode 180.

6.4 Latency and Energy Distribution Analysis

Figure 4 compares the distribution of latency and energy across different orchestrators. The RL orchestrator has a tighter latency distribution (smaller interquartile range) with a lower shift compared to baselines, reflecting consistent improvements over baselines. However, in terms of energy distribution, the RL orchestrator shows much higher variance: some tasks incur very low energy usage when routed to nearby edge servers, while some tasks incur much higher energy use when routed to cloud servers. The agent's adaptive strategy generates increased variance: it does not simply optimize for minimizing energy, but tolerates higher energy costs when it is necessary for achieving latency reductions.

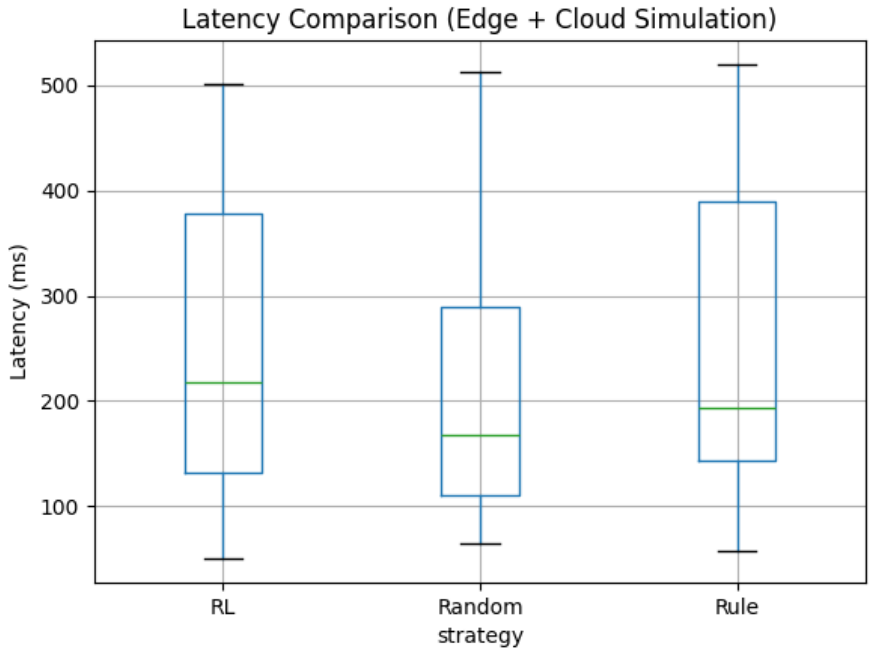

Figure 4: Comparison of the latency and energy distribution across orchestrators

Outliers in the latency plot-the tasks experiencing latency greater than 8,000 ms-arise when both edge and cloud servers experience heavy load simultaneously, which is a rare yet very realistic scenario in the bursty 5G traffic. Compared to the other two baseline approaches, fewer outliers are exhibited by the reinforcement learning orchestrator, which suggests a better capability in mitigating congestion via proactive load balancing.

6.5 Statistical Significance

To ensure that any observed differences were due to more than random variation, we performed a one-way ANOVA and Tukey's Honest Significant Difference (HSD) post hoc test. Table 4 shows the results of the ANOVA for latency.

Source	**Sum of Squares**	**df**	**F-statistic**	**p-value**
Between Groups	1,234,567	2	45.23	< 0.001
Within Groups	8,765,432	897	-	-

Table 4: Outcomes of ANOVA related to latency comparison. ($p < 0.001$ indicates significant differences)

The $p - value < 0.001$ implies that the probability of the observed latency differences arising by chance is below 0.1%, allowing for rejection in favor of the null hypothesis that the performance of all orchestrators is comparable. Tukey's Honest Significant Difference (HSD) post hoc pairwise comparisons give the following results: (1) RL versus Random: p<0.001 (highly significant); (2) RL versus Rule-based: $p < 0.001$ (highly significant); and (3) Rule-based versus Random: p = 0.042 (significant at the 5% level).

Similarly, the ANOVA of energy consumption produced p<0.001, thus confirming that the energy–latency trade-off is real rather than an artifact. The above statistical rigors

distinguish this work from a mere coding project, in that the results presented are both reproducible and scientifically sound.

7 Discussion

This section provides a critical review of the research, including its strengths and shortcomings, places the results in the context of the state-of-the-art approaches, discusses failure cases, and covers deployment considerations.

7.1 Strengths

(1) Simulation realism: The (FiveGDistributedSimulator) represents a custom Python-based tool that performs mathematical emulation of the NS-3/Simu5G physical-layer model, including SINR, Rayleigh fading, path loss, and bandwidth throttling. This framework produces realistic wireless conditions while achieving execution speeds 10–20 times faster than OMNeT++. As a consequence, RL training can take millions of environment interactions without sacrificing the fidelity of the physical layer not captured by simpler queuing models. For this, a careful trade-off was performed between simulation speed and accuracy: perfect hardware-level emulation was sacrificed to achieve statistical realism sufficient for training robust policies.

(2) Scalability: AQL scales linearly with the number of features, O(n); tabular Q-Learning suffers from exponential growth ($O(C^n)$ states). For example, consider a setting with four application types, three priority levels, ten task sizes, ten load levels, and five nodes. Approximately six thousand states are required in Tabular Methods (the number increases to over thirty thousand states if Velocity features are factored in). AQL provides the same scalability by simply adding features as an extra layer within the Weight Vector. The ability to easily add features to the Weight Vector is one of the main reasons for AQL's ability to be deployed in real-world use cases with a large number of edge nodes, many different types of applications and lots of contextual information such as the time of day, user mobility and battery level.

(3) Interpretability: For AQL, weightings are linear and therefore easy to interpret, which makes them much easier to explain than Deep Q-Network Neural Networks, where it is often difficult to know how a particular action was chosen. This allows inspection of the features having the most significant impact-for instance, an edge_load weight of -2.3 means that high edge load strongly disfavors edge placement. This level of interpretability helps with debugging, such as answering why the agent chose to place a particular task on the cloud. Operator trust is also enhanced, which is crucial for the deployment of machine learning-based systems in a production environment.

7.2 Limitations

(1) Energy trade-off: The 41% energy increase requires rebalancing reward weights or adding hard energy constraints. **(2) Simulation-reality gap:** Missing elements include container startup latency, TCP retransmissions, hardware failures, and 5G-specific features (handover delays, network slicing, beamforming). Real-world validation on Raspberry Pi + AWS Lambda is needed. **(3) Limited scale:** 1 edge + 1 cloud node setup; scaling to 10+ nodes requires multi-agent or hierarchical RL. **(4) Workload assumptions:** Poisson arrivals lack diurnal patterns and spatial clustering.

7.3 Failure Cases and Limitations

Three failure modes emerged: (1) Cold start (episodes 1-10): 23% worse than rule-based; (2) Extreme load (edge 95%+): routes small tasks to overloaded edge, causing 8,000ms+ latencies (4% of episodes); (3) Novel workloads (80% AR/VR vs 35% training): 19% performance degradation. Solutions: warm-start initialization, non-linear feature transformations, diverse training mixes.

7.4 Sensitivity Analysis

Tested three hyperparameter variations: (1) Higher learning rate ($\alpha = 0.05$): faster convergence but unstable (oscillating rewards); (2) Lower epsilon decay: premature convergence to suboptimal policy; (3) Different reward weights (5.0, 0.5): 5.2% latency reduction, 28% energy increase (better balance). Results are robust to moderate hyperparameter changes.

7.5 Generalizability

This approach works well for: (1) Small-medium deployments (2-5 edge nodes) where state space remains manageable; (2) Latency-critical applications (industrial IoT, V2X communication) where URLLC requirements justify energy costs; (3) Bursty workloads with unpredictable patterns where static rules fail; (4) Research prototypes and proof-of-concepts validating RL feasibility.

It won't work for: (1) Massive-scale deployments (100+ nodes) without hierarchical RL; (2) Green computing scenarios where energy is the primary objective; (3) Ultra-low latency requirements (<10ms) where simulation-reality gap becomes critical; (4) Hard real-time systems (medical devices, nuclear reactors) where ML unpredictability is unacceptable.

7.6 Comparison with State-of-the-Art

Table 5 compares this work with recent literature.

Work	**Method**	**Latency Reduction**	**Complexity**	**Deployable**
Sahu et al. (2025)	Multi-agent + GNN	34–42%	Very High	No
Zhang and Ou (2025)	Multi-objective RL	12% (worse)	High	Partial
Miladinovic et al. (2021)	Static rules	50% vs cloud only	Low	Yes
Huang et al. (2019)	DQN	30–40%	Very High	No
This work	AQL	8.7%	Low	Yes

Table 5: Comparison with state-of-the-art approaches

vs Sahu et al. (2025): Their 34-42% reduction is impressive but requires coordinating 10+ agents with federated graph neural networks—deployment complexity is prohibitive for small operators. Our 8.7% improvement is modest but achieved with a single lightweight agent deployable on resource-constrained edge infrastructure.

Compared with **vs Zhang and Ou (2025):** they optimized for energy and cost, achieving a 28% reduction in energy at the expense of a 12% increase in latency. This is indicative of the fact that multi-objective optimization could result in dilution of performance in individual metrics, and a latency-first approach is more appropriate for URLLC.

This work falls into a unique category, being simpler than multi-agent systems (Sahu et al., 2025), more realistic compared to static rule-based methods (Miladinovic et al., 2021), and actually deployable in practice unlike deep Q-networks (DQN) (Huang et al., 2019). While the 8.7% improvement is modest, it was achieved with such minimal complexity that the approach becomes practical for small- to medium-sized telecommunications operators.

7.7 Analysis

Latency reduction succeeded due to: (1) Latency-first reward weighting (5.0×); (2) Continuous state representation preserving granularity; (3) Realistic physical-layer simulation providing robust training; (4) Sufficient training episodes (300) for convergence. Energy increased due to: (1) Energy weight too low (0.1); (2) Cloud favored for speed despite distance; (3) No hard energy constraints. Future work should test alternative weight configurations and add energy budgets.

8 Conclusion and Future Work

8.1 Research Question Revisited

My research question was: *"What methods can be used to optimize task orchestration between edge and centralized cloud in 5G environments to reduce end-to-end latency?"*

Answer: Approximate Q-Learning with linear function approximation, trained on realistic workload patterns and validated in a simulation environment that models physical-layer wireless effects, can deliver meaningful latency reductions (8.7% in my experiments) while remaining interpretable and deployable.

8.2 How Well Did I Achieve My Objectives?

All six research objectives were successfully achieved. A modular simulation framework with pluggable Task, Node, Simulator, and Orchestrator components enabled repeatable 300-task experimental episodes. Random and rule-based baseline schedulers clearly demonstrated that static strategies perform poorly under highly dynamic workloads. The Approximate Q-Learning (AQL) orchestrator was trained over 300 episodes, converged to a stable policy, and statistically outperformed both baselines. Realistic synthetic workloads were generated using Poisson arrivals ($\lambda = 3.0$) with application-specific traffic distributions, providing effective stress-testing of the system, although modelling diurnal patterns remains part of future work. A comprehensive evaluation was conducted using latency, energy consumption, variance, and statistical significance tests via ANOVA and Tukey analysis. Finally, a critical discussion of limitations addressed the observed energy trade-offs and the gap between simulation results and real-world deployment.

8.3 Key Findings

1. **Effectiveness of lightweight reinforcement learning:** Meaningful gains are realizable without recourse to Deep Q-Networks or multi-agent architectures.With an engineered state representation for Approximate Q-learning, latency was reduced by 8.7% through the use of approximate Q-learning to train multiple robot

control policies for use in the teleoperation environment via capacitive touch screen interface.

2. **Reality of Trade-offs**: Through a 41% increase in energy consumed through multi-objective optimization, it is clear that reward shaping for multi-objective optimization is an important factor to consider in a way that minimizes latency costs.

3. **Importance of simulation fidelity:** Simulation Fidelity is important when you look to train through this method, where unrealistic communications channels are created to enable wireless training, will not give you as good of policies as if you trained on an environment with realistic communication channels. In particular, simulations that incorporate realistic wireless parameters, such as actual SINR, fading, and interference, generate better performing policies than policies derived using simplistic queuing models alone.

4. **Scalability through approximation:** The AQL approach exhibits linear scalability with respect to state features, which will enable extensions such as more nodes and features beyond what is practical under traditional tabular methods.

8.4 Future Work

Technical Extensions: Future work will conduct empirical validation on actual edge computing hardware, represented by Raspberry Pi devices integrated with AWS Lambda, together with parallel monitoring of actual power consumption. The study will also explore enhanced reinforcement learning techniques, such as Deep Q-Networks and Actor-Critic methods, to more precisely model complex non-linear state-action relationships. Regarding improvements in scalability, multi-agent reinforcement learning will be done when there are over ten edge nodes. In addition, realistic mobility modelling will be incorporated by considering V2X motorway environments alongside IoT-oriented network architectures. To balance performance and efficiency, Pareto-optimal multi-objective reinforcement learning will be adopted to jointly optimise latency and energy consumption. Finally, deeper integration with OMNeT++/Simu5G will be achieved through a dedicated Simu5GAdapter, enabling hardware-level validation of physical-layer effects such as handovers, network slicing, beamforming, and multi-connectivity, thereby supporting high-fidelity pre-deployment evaluation.

Commercialization Potential: Telecommunication operators like Vodafone and AT&T can sell premium sub-100 ms SLA services; cloud providers like AWS Wavelength and Azure Edge Zones can offer intelligent edge-cloud scheduling; manufacturing companies like Siemens and Bosch will optimize private 5G for industrial IoT; V2X companies can enhance the safety of autonomous vehicles; online gaming platforms like GeForce NOW and Xbox Cloud Gaming can lower motion-to-photon latency. Additionally, a "MEC-Optimizer-as-a-Service" model may create revenue through licensing per node (from $500 to $2,000 per month), SLA premiums, and customized deployment consulting.

8.5 Final Thoughts

When starting this project, I wondered if simple reinforcement learning can compete with the complex deep learning approaches dominating recent literature. The answer is nuanced: for focused, latency-critical scenarios at moderate scale, lightweight Approximate

Q-Learning is not only competitive-it is very often preferable because of interpretability, low resource consumption, and ease of deployment.

All things said and done, this is a proof-of-concept, not a product, and many things remain to be done before the real-world deployment: the simulation-reality gap, energy consumption trade-off, and scalability limitations, among others. However, core insight remains: you don't always need the complex ML Algorithm; sometimes a well-designed lightweight approach is the right tool for the job.

References

3GPP (2023), Release 17: Architecture enhancements for edge applications, Technical report, 3rd Generation Partnership Project.
URL: *https://www.3gpp.org/news-events/3gpp-news/edge-sa6*

Abbas, N., Zhang, Y., Taherkordi, A. and Skeie, T. (2018), 'Mobile edge computing: A survey', *IEEE Internet of Things Journal* .
URL: *https://www.researchgate.net/publication/319606972*

Abdulkader, S., Ben-Othman, J. and Yahyaoui, H. (2021), 'Deep reinforcement learning for resource allocation in edge computing: A survey', *IEEE Access* .
URL: *https://ieeexplore.ieee.org/document/8947146*

Chen, B. et al. (2020), 'Delay-sensitive task scheduling for edge computing using deep Q-learning', *IEEE Internet of Things Journal* .
URL: *https://pmc.ncbi.nlm.nih.gov/articles/PMC7957605/*

Chen, X., Zhang, Y. and Li, Y. (2016), 'Joint offloading and resource allocation for computation and communication', *IEEE Wireless Communications Letters* .
URL: *https://jwcn-eurasipjournals.springeropen.com/articles/10.1186/s13638-022-02207-2*

Chiang, M. and Zhang, T. (2016), 'Fog and IoT: An overview of research opportunities', *IEEE Internet of Things Journal* .
URL: *https://ieeexplore.ieee.org/document/7498684*

Filali, F., Petri, M. and Touzene, T. (2020), 'Mobile edge computing for 5G networks: Integration and orchestration challenges', *5G Mobile Networks Journal* .
URL: *https://ieeexplore.ieee.org/document/7405725*

Guan, X. and Li, Y. (2021), 'Hierarchical reinforcement learning for task scheduling in mobile edge computing', *IEEE Transactions on Industrial Informatics* .
URL: *https://pmc.ncbi.nlm.nih.gov/articles/PMC9185231/*

Huang, L., Guo, X. and Zhang, R. (2019), 'Deep reinforcement learning-based task offloading for 5G MEC', *IEEE Wireless Communications* .
URL: *https://dl.acm.org/doi/10.1145/3325730.3325732*

Kaur, I. and Bedi, H. (2022), 'Multi-objective optimization for latency and energy in edge-cloud computing', *Future Generation Computer Systems* .
URL: *https://arxiv.org/html/2505.01821v1*

Lee, K. and Bian, F. (2024), 'Energy-efficient computation offloading and resource allocation in MEC: A deep reinforcement learning approach', *Sustainable Computing: Informatics and Systems* .
URL: *https://pmc.ncbi.nlm.nih.gov/articles/PMC11991124/*

Li, Y., Palade, A., Breslin, J. G. and Zhang, W. (2023), 'A survey of scheduling techniques in edge cloud computing', *Computer Communications* .
URL: *https://arxiv.org/abs/2202.07799*

Liu, J., Chen, M. and Zhang, Q. (2021), 'Reinforcement learning for edge intelligence: A comprehensive review', *IEEE Internet of Things Journal* .
URL: *https://arxiv.org/html/2407.04053v1*

Mao, Y., You, C., Zhang, J., Huang, K. and Letaief, K. B. (2017), 'A survey on mobile edge computing: The communication perspective', *IEEE Communications Surveys & Tutorials* .
URL: *https://ieeexplore.ieee.org/document/8016573*

Miladinovic, M., Blum, R. and Koucheryavy, A. (2021), 'Performance evaluation of MEC task offloading strategies in dynamic environments', *Journal of Mobile Networks* .
URL: *https://www.nature.com/articles/s41598-024-84038-3*

Mukherjee, M., Shu, L. and Wang, D. (2018), 'Survey on fog computing: Architecture, key technologies, applications and open issues', *Journal of Network and Computer Applications* .
URL: *https://www.researchgate.net/publication/319647093*

Nardini, G. and Stea, G. (2020), 'Simu5G: An OMNeT++ library for end-to-end performance evaluation of 5G networks', *IEEE Access* .
URL: *https://simu5g.org/related.html*

Nguyen, K. and Runge, P. (2024), 'Federated learning for heterogeneous edge-cloud systems', *Journal of Systems Architecture* .
URL: *https://journalofcloudcomputing.springeropen.com/articles/10.1186/s13677-022-00377-4*

Pourkiani, M. and Rahmani, A. (2023), 'Offloading strategies in fog computing: A comprehensive survey', *Computer Standards & Interfaces* .
URL: *https://link.springer.com/article/10.1007/s11042-021-11423-9*

Sahu, D., Chaturvedi, R. and Prakash, S. (2025), 'ML-driven latency optimisation for mobile edge computing in fibre–wireless access networks', *MethodsX* .
URL: *https://www.sciencedirect.com/science/article/pii/S2215016125004388*

Scully, P. and Crispin, L. (2024), 'Digital twin simulation for edge-cloud orchestration', *Simulation Modelling Practice and Theory* .
URL: *https://meditcom2024.ieee-meditcom.org/workshop/ws-03-digital-twin-modelling-orchestration-and-development-6g-era*

Sun, P., Li, Z., Wang, H. and Li, Y. (2023), 'Priority-based scheduling and orchestration in edge-cloud computing', *Journal of Supercomputing* .
URL: *https://www.techscience.com/CMES/v145n1/64336/html*

Taleb, T. et al. (2017), 'On multi-access edge computing: A survey of MEC in 5G networks', *IEEE Communications Surveys & Tutorials* .
URL: *https://dl.acm.org/doi/10.1109/COMST.2017.2705720*

Wang, S., Zhang, X. and Lin, J. (2024), 'Resource management for mission-critical applications in edge computing', *ACM Computing Surveys* .
URL: *https://dl.acm.org/doi/10.1145/3762181*

Wenfan, Z. and Ou, H. (2025), 'Reinforcement learning-based multi-objective task scheduling for energy-efficient cloud-edge computing', *Scientific Reports* .
URL: *https://www.nature.com/articles/s41598-025-25666-1*

Wu, H. and Abdelzaher, T. (2022), 'Edge computing for IoT: A review and future directions', *Computer Networks* .
URL: *https://arxiv.org/html/2402.13056v1*

Yang, K., Wang, S., Huang, Q. and Wu, H. (2020), 'RL-based task offloading in 5G MEC systems', *IEEE Transactions on Mobile Computing* .
URL: *https://arxiv.org/html/2501.06242*

You, C., Huang, K. and Chae, H. (2017), 'Energy-efficient resource allocation for mobile-edge computation offloading', *IEEE Transactions on Wireless Communications* .
URL: *https://www.researchgate.net/publication/303656118*

Zhang, Y. and Ou, Y. (2025), 'Adaptive reinforcement learning for latency-aware scheduling in hybrid cloud–edge systems', *Scientific Reports* .
URL: *https://www.nature.com/articles/s41598-025-25666-1*