National College of Ireland

Energy-Latency Trade-Off Aware Load Balancing in Fog Computing Using Pareto-DQN Algorithm

MSc Research Project
Cloud Computing

Myungjee Koh
Student ID: x23268409

School of Computing
National College of Ireland

Supervisor: Shaguna Gupta

National College of Ireland
Project Submission Sheet
School of Computing

National College of Ireland

Student Name:	Myungjee Koh
Student ID:	x23268409
Programme:	Cloud Computing
Year:	2025
Module:	MSc Research Project
Supervisor:	Shaguna Gupta
Submission Due Date:	11/08/2025
Project Title:	Energy-Latency Trade-Off Aware Load Balancing in Fog Computing Using Pareto-DQN Algorithm
Word Count:	7,689
Page Count:	26

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:	
Date:	15th September 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):	

Energy-Latency Trade-Off Aware Load Balancing in Fog Computing Using Pareto-DQN Algorithm

Myungjee Koh
x23268409

Abstract

Fog computing (FC) has emerged as a key paradigm for low-latency application. In contrast to the centralize computing model, a decentralized, heterogeneous FC acts as the intermediate node between the IoT devices and cloud server. Meanwhile, reducing energy consumption in computational infrastructure is directed toward is now of utmost priority. Although Fog computing offers an opportunity to reduce energy consumption by enabling localized processing, it still remains a conflicting nature of two objectives - latency and energy consumption. To address this issue, this paper proposes a dynamic load balancing strategy using Pareto Deep Q-Networks (PDQN) algorithm by focusing on the energy-latency trade-off. The Pareto front is a set of all non-dominate solutions, which directly captures the trade-offs between the objectives. PDQN algorithm integrates a deep reinforcement learning framework with Pareto front guided selection. This novelty lies in its ability to jointly optimize latency and energy objectives dynamically adapting to heterogeneous fog environments. The study contributes not only to achieve optimize among the fog nodes but also to supplement by cloud computing.

1 Introduction

1.1 Background and Motivation

In recent years, the widespread growth of Internet of Things (IoT) devices has resulted in an explosion of data generation. IoT has become an essential role in our daily life with billions of connected devices generating vast amounts of data. IoT is growing continuously to proliferate across diverse sectors. However, the traditional cloud technology is becoming latency-prone and less suitable for time-critical next-generation IoT applications such as intelligent transportation systems (ITS), smart cities, smart healthcare, smart agriculture, manufacturing, construction, and mining due to long transmission delay and congestion of the network from sensing devices to the cloud data centres. [13]
To address these problems, Fog computing (FC) has emerged as a key paradigm for low-latency application. Even though, efficient resource management, task scheduling, and load balancing pose challenges in fog environments. [3] In contrast to the centralize computing model, a decentralized, heterogeneous fog computing paradigm act as the intermediate node between the IoT devices and cloud server and has been used to process time-sensitive application that supports all the cloud-like services by providing computing, storage, and communication. [4]
On the other hand, as the urgency of climate change grows, reducing energy consumption

in computational infrastructure is now of utmost priority. According to recent statistics, data centers currently consume one fifth of Ireland's electricity and projected to use one-third in the next few years. [14] This rising demand is directed toward the energy-efficient designs that must weigh climate priorities and facilitate the digital economy. Fog computing offers an opportunity to reduce energy consumption by enabling localized processing, but it also requires intelligent resource management.
In this context, the study aims to develop a dynamic load balancing strategy that can optimize both latency and energy consumption. Although the conflicting nature of these two objectives, the proposed approach focuses on the energy-latency trade-off.

1.2 Challenges in Load Balancing

Load balancing (LB) in fog computing is a critical method to offer system responsiveness, resource efficiency, and reliability. Load balancing involves spreading the workload over the multiple fog nodes to avoid overload and utilize the resource optimally. However, designing an effective load balancing strategy in fog computing is particularly difficult due to the following reasons:
Firstly, there is an inherent trade-off among decreasing latency and decreasing energy consumption. Offloading the task to the closest node minimizes latency but may lead to inefficient energy consumption. Conversely, transmitting tasks to energy-efficient nodes can achieve maximum transmission delay. This two-objective trade-off complicates decision-making.
Second, fog environments are highly heterogeneous and dynamic. Nodes vary in terms of CPU power, memory, network bandwidth, and available energy. Moreover, task arrival rates and user demands vary unpredictably with time. All these attributes necessitate a load balancing strategy that is adaptive, context-aware, and scalable.
Third, network conditions vary between regions, and connectivity may be intermittent or unreliable. Fog nodes might fail or get disconnected at any time, so fault-tolerant mechanisms are required. [9]
To tackle these challenges, a dynamic and flexible approach such as Deep reinforcement learning (DRL) is extremely well positioned. DRL is capable of learning optimal task distribution policies from experience by exploring the world through trial and error, enabling autonomous decision-making in dynamic environments.

1.3 Research Questions

This paper investigates the feasibility and effectiveness of Pareto-DQN based dynamic load balancing between fog nodes. Specifically, it explores how to balance a trade-off between latency and energy consumption cost with a multi-objective optimization framework. The following research questions are addressed.

- RQ1: Can a Pareto DQN-based load balancer learn an optimal policy that effectively balances the trade-off between latency and energy consumption?
- RQ2: How does the Pareto DQN-based strategy perform compared to traditional load balancing methods such as Round-robin (RR) and Weighted Round-robin (WRR)?

In order to enhance the trade-off performance further, the system uses Pareto-DQN optimization techniques on the reward estimation that enables to optimize among multiple goals without requiring fixed weights.

1.4 Limitations

Despite the potential benefits of the Pareto Deep Q-Network (PDQN)-based load balancing strategy in fog computing, this project has several limitations. First, it relies on simulation-based evaluation. All experiments are conducted in the YAFS (Yet Another Fog Simulator) environment. While this setup enables flexible experimentation, it does not fully capture real-world variability such as network congestion, hardware faults, or sensor inaccuracies.
In addition, the Pareto-DQN agent is trained and evaluated in a static and predefined simulation environment. Its application to unseen topologies, other models of application, or highly dynamic workloads has not been examined.
Also, since PDQN has an offline learning paradigm, the agent does not interact with the environment when it is training. While this reduces the need for exploration-exploitation trade-offs, it augments sensitivity to the quality and diversity of the offline training data. In case the dataset dose not contain representative state-action transitions or edge-cases, the policy being learned can learn not to generalize properly in deployment.
Lastly, the energy model used in this work is simplified. Energy consumption is estimated based on instruction counts and an IPT-based constant multiplier. A more accurate model would incorporate hardware-specific power characteristics, idle states, and communication energy consumption for various node types.

Paper Organization: The structure of this paper is organized as follows, Section 2 reviews the relative work. Sections 3 and 4 discuss problem formulation, energy consumption estimation, the proposed system flow and structure, and used data. Section 5 introduces the Pareto-DQN algorithm in fog computing, and YAFS simulator. The performance evaluation of the proposed algorithms is discussed in Section 6. Finally, Section 7 summarizes conclusions based on the finding of this project including future research.

2 Related Work

In fog computing environment, there are lots of existing dynamic load balancing algorithms. This section discusses load balancing strategies followed by main categories are recognized: heuristic, meta-heuristic, machine learning, fundamental, hybrid and software-defined networking (SDN).

2.1 Heuristic Algorithms

Ala'anzy, M.A. et al. (2024) [1] proposed an Optimized Load Balancing (OLB) algorithm to allocate workloads effectively across fog nodes and to reduce communication and computational delays in IoT-driven healthcare environments. The system architecture is a three-tier framework, which consists of IoT layer, Fog layer and Cloud layer. By simultaneously addressing traffic demands and computing power, the approach enhances latency, network usage and execution time comparing another load balancing techniques. The

OLB algorithm maintains an updated array of total latencies, recalculating only for the specific device being assigned, which allows the system to reduce redundant computations and improve response times. Although it shown significant result of a reduction in latency and improvement in network usage, it still has a limitation about integration with ML technique.
M. Ebrahim and A. Hafid (2023) [10] provided a Multi-Criteria Decision Analysis (MCDA) approach using ELECTRE to efficiently balance the load in Fog computing.
A.Ashgar et al. (2021) [5] proposed a fog-based health monitoring system and a new Load Balancing Scheme (LBS) that addresses two types of latencies, which is communication latency caused by traffic load and computing latency.
S. Chang and H. Lutfiyya (2024) [8] proposed a value-iteration load balancing algorithm to optimize the task execution time. They built a real computing cluster using multiple Raspberry Pis to form fog nodes.
R. Beraldi et al. (2020) [7] introduced two random walk-based, fully distributed load balancing algorithms for fog computing – Sequential Forwarding and Adaptive Sequential Forwarding. It considers both a simplified scenario, namely simplified scenario that models the fog nodes as M/M/1/Q queuing systems. The other is the realistic scenario, which is based on smart city scenarios. While the adaptive algorithm adjusts thresholds with hops, it does not dynamically adjust based on actual node metrics such as CPU usage, energy constraints and wireless channel state.

2.2 Meta-Heuristic Algorithms

Sing R et al. (2023) [24] developed using Bald Eagle Search algorithm for optimally allocating resource through load balance. The algorithm focused on reducing the energy consumption and enhancing makespan and resource utilization. It is compared with other algorithms: Dynamic Resource Allocation Method (DRAM), Load Balanced Service Scheduling Approach (LBSSA) and Whale Optimization algorithm and bat algorithm (WOA-BAT). The result states that the algorithm is relatively better than theses three algorithm in terms of makespan, energy consumption and percentage of resource utilization.
Y. Seraj et al. (2024) [23] introduced Load-balanced Offloading with MAPE (Monitor-Analyze-Plan-Execute) and Particle Swarm Optimization in Mobile Fog Networks (LIMO). LIMO is a mobility-aware autonomous task migration policy by employing the POS meta-heuristic algorithm.
A. Nayak et al. (2024) [19] suggested a Particle Swarm Optimization (PSO)-based load balancing architecture. It shows the result by reducing response time, processing time, and costs comparted to Round Robin, Throttled Algorithm in fog-cloud environments. However, It primarily tests under synthetic, uniform request sizes (100 bytes/request) and rates (60 requests/user), which might not effective burstiness or heterogeneous workload patterns typical in IoT environments.

2.3 Machine Learning Algorithms

H. Kaur et al. (2024) [16] proposes Enhanced K-means clustering of tasks and VM for LB at fog layer (EKCLB) within smart city scenarios. Tasks are clustered based on priority and burst time and it combines to Greedy Task-to-VM Assignment and Particle Swarm optimization (PSO) for load balancing refinement. The results show improving makespan

and resource utilization but lacks energy analysis.
N.T Pouya et al. (2021) [25] introduce novel load balancing method using the Double-Q-learning algorithm to find a low-load fog node. This reinforcement learning approach aims to minimize processing delay and overload probability by learning the best load balancing policies over time without requiring neighbors state information.

2.4 Fundamental Algorithms

S. Thangam et al. (2024) [22] provided a analysis of three load balancing algorithms i.e. round robin, weighted round robin and least loaded with measure metrics such as cost of execution, network usage and application loop delay. However, it revealed subtle variations in result.
S. Ali and R. Alubady (2023) [2] introduced the Remind Weighted Round Robin (RWRR) load balancing algorithm to enhance the QoS metrices such as latency and reaction time. RWRR takes into consideration the overall capabilities of each VM and the length of a task for each request, which can assign the task to suitable VMs.

2.5 Hybrid Algorithms

N.E. Belkout et al. (2022) [6] proposed a load balancing strategy using Deep Reinforcement Learning (DRL) combined with a link-state rousting protocol based on the Dijkstra algorithm. Furthermore, the authors introduced a load balancer smart controller (LBSC) which is responsible for distributing the tasks among the nodes and routing them to the destination. The DRL agent is charge of load balancing strategy using a neural network. The evaluation results show that the proposed algorithm achieves a lower processing delay compared to classic load balancing algorithms.
N.J. Kumar et al. (2025) [17] suggested a hybrid load balancing framework using K-Means and Firefly Algorithm in fog-based healthcare environments. K-Means clustering groups tasks based on workload, burst time, and criticality score. And Firefly Algorithm optimizes task assignments within each. Although results show effectively improve processing speed and energy efficiency, the specific simulation tool is not mentioned.

2.6 Software-Defined Networking (SDN)

A.M.Jasim and Al-Raweshidy, H. (2024) [15] introduced a load-balancing framework for healthcare systems in urban areas by integrating static and SDN-based load-balancing algorithms. The results shown the novel and effectiveness of the proposed method in system latency and deployment cost compared to their previous studies.
L. Pani et al. (2022) [20] suggested an SDN-enabled, energy-efficient fog computing architecture for healthcare IoT data. This approach features to balance energy saving with maintaining QoS for different traffic classes. While real-time data are processed immediately, delay-tolerant data are processed in batches. However its evaluation is conducted purely through analytical modelling, and no simulations or real-world testbed validations are provided.

Table 1: A comparative analysis of load balancing in fog computing

Author	Method	Approach	Optimization Objectives	Evaluated Against	Simulation Tool
Ala'anzy, M.A. et al. (2024)	Heuristic	Optimized Load Balancing (OLB) algorithm	To reduce communication and computational delays in IoT-driven healthcare environments	LBS, FNPA, LAB, MEC	iFogSim
Ebrahim, M. et al. (2023)		Multi-Criteria Decision Analysis (MCDA)	To make service selection decisions including compute and network load information	Random, DRR, Nearest and Fastest	YAFS
Asghar, A. et al. (2021)		New Load Balancing Scheme (LBS)	To address communication latency and computing latency	Cloud-only, FNPA, LAB	iFogSim
Chang, S. et al. (2024)		Greedy + Online Learning	To optimize task execution time	Weighted Round Robin, Min-Min, Active Monitoring Algorithm	Raspberry Pi Testbed
Beraldi, R. et al. (2020)		Sequential and Adaptive Forwarding	To provide fair load sharing in heterogeneous workload	No Load Balancing Case	Matlab, OMNeT++
Sing, R. et al. (2023)	Meta-heuristic	Bald Eagle Search (BES) Optimization	To reduce energy consumption, enhance makespan, and resource utilization	DRAM, LBSSA, WOA-BAT	Python
Seraj, Y. et al. (2024)		Particle Swarm Optimization (PSO)	To enhance system efficiency with user mobility	MAMF offloading	MobFogSim
Nayak, A. et al. (2024)		PSO-based method	To reduce processing delays and boost performance	Round Robin, Throttled	CloudSim, CloudAnalyst
Kaur, H. et al. (2024)	ML Algorithms	Enhanced K-means Clustering	To reduce makespan and enhance resource utilization	Non-clustering approach	iFogSim2
Pouya, N.T. et al. (2021)		Double-Q-learning	To minimize processing time and overload	SSLB, Proportional, Random	iFogSim
Thangam, S. et al. (2024)	Fundamental	RR, LL, WRR	To reduce execution cost, network usage, and loop delay	RR, LL, WRR	iFogSim
Ali, S. et al. (2023)		Enhanced WRR	To improve QoS (latency, reaction time)	WRR, Least Connection, RHO	Visual Studio (C#), Arduino
Belkout, N.E. et al. (2022)	Hybrid Algorithms	DRL + Dijkstra heuristic	To minimize task processing and communication delay	Random, RR, Weighted RR	YAFS
Kumar, N.J. et al. (2025)		K-Means + Firefly Algorithm	To handle dynamic task scheduling complexity	Hybrid models (LBMACO, FAPSO, FGAACO)	Simulation-based (unspecified)
Jasim, A.M et al. (2024)	SDN-based	LB-OESP and SDN-GH	To optimize resource utilization, reduce response time	OESP, HOSSC	MATLAB
Pani, L. et al. (2022)		SDN + MPSS-based processing	To design energy-efficient SDN for healthcare data	MSS, MPSS, MPSSI	–

2.7 Research Gaps and Limitations

The recent advances in fog computing have provided a wide range of load balancing techniques, each accompanied with its own assumptions and compromises. Heuristic and meta-heuristic techniques such as OLB, BES, and PSO have shown promise to enhance response time or energy consumption but are usually sensitive to tuning parameters and may not scale up in real-time, dynamic fog networks. Alternatively, rule-based or basic algorithms like Weighted Round Robin (WRR) are easy to implement but possess no adaptability and decision capability when encountering changing network states. Machine learning-based methods, especially reinforcement learning, have also been explored more recently for fog computing. For example, a Double Q-learning-based method for minimizing delay and overload proposed by Pouya et al. (2021). [25] Such methods focus primarily on optimizing a single objective such as delay or load minimization and do not address directly the trade-offs between conflicting goals like energy consumption and latency.

A key inspiration for this research comes from Belkout et al. (2022), [6] who proposed a hybrid load balancing framework that used DRL and a heuristic routing protocol (Dijkstra algorithm). Their work introduced a smart load balancer controller (LBSC) in a fog computing setting and demonstrated better performance than traditional schedulers. This work provides the foundation for my contribution in using DRL for load balancing. But Belkout et al. (2022) [6] were mainly focused on minimizing computation and communication delay. Energy us was not included in the reward signal or optimization goal. This creates a limitation for applications in environments where energy efficiency is equally or more critical. Because in edge and fog systems powered by limited-capacity batteries or renewable sources.

To address this, the study applys a multi-objective reinforcement learning framework from A.B. Esan et al. (2023), [11] where Pareto Q-learning was utilized to successfully balance between operation cost and load loss probability in solar microgrids. Motivated by their results, which showed that Pareto-based Q-learning can effectively manage trade-offs between conflicting objectives.

Furthermore, Pareto-Q-Learning (PQL) can only perform in and environment with a small-sized state-space. M.Reymond and A.Nowé (2019) [21] proposed a novel algorithm: Pareto-DQN, that will estimate the Pareto front of complex environment, with a high-dimensional state-space.

Thus, the project primarily focuses on developing a load balancing strategy that can simultaneously optimize two conflicting objects: latency and energy consumption. The proposed method is combined the Pareto front approach with Reinforcement Learning (RL) would provide a promising solution. Unlike nature inspired algorithms, which typically need to be re-designed or re-tuned when the environment changes, reinforcement learning can dynamically adapt to the continuously changing condition of a fog environment.

Specifically Deep Q-Network (DQN) was chosen because it enables the sequential learning of dynamic load balancing decisions in fog computing. DQN is an extension of the traditional Q-Learning algorithm, where instead of maintaining a large Q-table that becomes impractical in high-dimensional state spaces, a deep neural network is used to approximate the Q-function. This allows the algorithm to handle complex and continuous state spaces, such as those arising from heterogeneous fog nodes with dynamic CPU utilization,

energy consumption, etc. By using DQN, the model can learn from past experiences, generalize to unseen system states, and gradually improve its policy over time. While nature inspired algorithms such as PSO or ACO are effective in global optimization, they often require repeated initialization and optimization for each new scenario. This adaptability makes DQN particularly suitable for fog environments are highly dynamic and unpredictable.
To sum up, the project's major contribution is generalizing DQN-based load balancing to the multi-objective case, proposing a Pareto-based reward design that capture both latency and energy trade-offs, and implementing the algorithm in a simulated fog environment using YAFS, allowing real-time task allocation and routing decisions.

3 Methodology

This section outlines the methodology adopted for developing a Pareto-DQN load balancing strategy in fog computing environments. The primary goal is to optimize the trade-off between energy consumption and latency in task scheduling among fog nodes. The methodology consists of three main parts: Problem Formulation, energy Consumption Estimation in YAFS, and system flow chart.

3.1 Problem Formulation

The proposed Pareto-DQN algorithm can be modeled as a Multi-objective Markov Decision Process(MOMDP). This problem formulation is made based on C.F Hayes et al. (2022) [12] The formulation is described as follows:

3.1.1 Multi-Objective Markov Decision Process (MOMDP)

The load balancing problem in the system is formalized as a Multi-Objective Markov Decision Process (MOMDP). This formulation captures the trade-offs between latency and energy consumption when distributing tasks among fog nodes. The crucial difference between a single-objective MDP and a MOMDP is the vector-valued reward function R, which means that the length of the reward vector is equal to the number of objectives. [12]

A MOMDP is defined as:

$$\langle S, A, T, \gamma, \mu, R \rangle \tag{1}$$

- S : The state space $S \in \mathbb{R}^{N \times d}$ captures the current system state, where $N = 10$ fog nodes and $d = 8$ features per node. It is represented as:

$$S = [s_1, s_2, \ldots, s_N]^T \tag{2}$$

where

$$s_i = \begin{bmatrix} CPU_i,\ Queue_i,\ Energy_i,\ Latency_i, \\ MA\_Latency_i,\ SuccessRate_i,\ Activity_i,\ Efficiency_i \end{bmatrix}, \quad \forall i \in \{1, 2, \ldots, N\}. \tag{3}$$

Here:

  - CPU_i: Current CPU utilization of fog node i (percentage).

- $Queue_i$: Proportion of queue occupancy at fog node i.
- $Energy_i$: Cumulative energy consumed by fog node i (Joules).
- $Latency_i$: Current measured latency for task completion at fog node i (milliseconds).
- $MA\_Latency_i$: Moving average latency over recent tasks for fog node i.
- $SuccessRate_i$: Task success rate (%) at fog node i.
- $Activity_i$: Normalized measure of node activity (e.g., queue length ratio).
- $Efficiency_i$: Computed processing efficiency metric for fog node i.

"Efficiency" is not a single metric representing CPU, storage, or network individually, but rather a comprehensive performance indicator that combines multiple metrics to evaluate the overall effectiveness of fog nodes. The efficiency metric is calculated using a weighted combination approach that takes into account three primary factors. The first component is CPU utilization, which represents how efficiently the node is using its computational resources. This accounts for 40% of the total efficiency score. The second component is success rate, which measures the percentage of tasks that are successfully processed by the node This also contributes 40% to the overall efficiency calculation. The third one is activity metric, which indicates how actively the node is processing tasks and represents 20% of the efficiency score. The calculation formula for efficiency follows this patterns:

$$efficiency = (cpu\_normalized \times 0.4 + success\_normalized \times 0.4 + activity\_metric \times 0.2) \times 100.0. \quad (4)$$

This approach ensures that the efficiency value is normalized to a 0-100% range, making it easier to compare different nodes and understand their relative performance levels. Additionally, there's an alternative real-time calculation method that uses a different formula:

$$efficiency = (success\_rate \times activity\_metric) \,/\, CPU\ utilization. \quad (5)$$

This formula emphasizes the relationship between successful task completion and resource consumption, where higher efficiency indicates that a node is achieving better results with lower resource usage.

In practical terms, a high efficiency score indicates that a fog node is performing well with minimal resource waste, successfully processing tasks, and maintaining active operation. Conversely, a low efficiency score suggests that a node might be consuming excessive resources while delivering poor results or experiencing operational issues. This efficiency metric essentially represents the overall "health status" of each fog node, enabling the system to identify which nodes are most suitable.

all these eight dimensions are normalized to the [0,1] range. CPU utilization is converted from percentage to [0,1] by dividing by 100, queue length is normalized

relative to a maximum of 10 messages, energy consumption is scaled relative to a maximum of 10J, and efficiency is converted from percentage to [0,1] by dividing by 100. This comprehensive normalization approach ensures that all parameters contribute equally to the learning process regardless of their original scales and units. Therefore, the state vector for the Pareto-DQN algorithm provides crucial information for intelligent node selection, which captures real-time performance, reliability, and efficiency metrics across N fog nodes, enabling the load balancing agent to make informed routing decisions.

- A : At each decision making, the agent selects a fog node to which the incoming request is dispatched:

$$A_t = \{FogNode_1,\ FogNode_2,\ \ldots,\ FogNode_n\} \tag{6}$$

  where $N \in [10, 30]$. Each action corresponds to routing the request to a specific fog node, thereby influencing its workload and energy usage.

- $T : S \times A \times S \to [0, 1]$ is the transition function, defining the probability of moving from one state to another.

- $\gamma \in [0, 1)$ is the discount factor, balancing immediate versus long-term objectives.

- $\mu : S \to [0, 1]$ represents the initial state distribution.

- $R : S \times A \times S \to \mathbb{R}^2$ is the reward function, defined as a two-dimensional vector.

$$R = [-Latency, -Energy] \tag{7}$$

  where:

$$Latency = (time\_reception - time\_emit)\ \times\ 1000(milliseconds) \tag{8}$$

$$Energy = WATT \times service\_time(J) \tag{9}$$

  By applying the negative sign, both objectives are naturally aligned with the reinforcement learning maximization framework, since lower latency and lower energy correspond to higher rewards.
  The other parameters listed in state, such as CPU utilization, queue occupancy or success rate, are not directly optimized within the reward function but instead included as part of the state representation. These characteristics enable the agent to learn system behavior and make more proactive load balancing decisions. In this way, proportional and inversely proportional measurements are assumed to be distinct. The performance objectives are converted into negative reward where they should be minimized, while descriptive properties remain in the state without direct conversion in the reward function.
  Energy consumption estimation will discuss next section. 3.2

3.1.2 Policies and Value Functions

A policy π defines the load balancing strategy as:

$$\pi : S \times A \to [0, 1]. \tag{10}$$

The value function under policy π is given by:

$$V^{\pi} = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{k+1} \;\middle|\; \pi, \mu\right], \tag{11}$$

where $r_{k+1} \in \mathbb{R}^2$ denotes the reward vector for latency and energy at time step $k+1$.

Unlike single-objective RL, this value function is vector-valued, producing a partial order among policies. This requires reasoning about trade-offs instead of a single optimal solution.

3.1.3 Solution Set and Pareto Front

Because improving one objective often degrades another, the solution is defined as a set of non-dominated policies, known as the Pareto Front.

Definition (Pareto Front): The Pareto Front $PF(\Pi)$ is defined as:

$$PF(\Pi) = \{\pi \in \Pi \mid \nexists \pi' \in \Pi : V^{\pi'} \succ_P V^{\pi}\}, \tag{12}$$

with Pareto dominance $\succ_P$ given by:

$$V^{\pi'} \succ_P V^{\pi} \iff \left(\forall i : V_i^{\pi'} \geq V_i^{\pi}\right) \wedge \left(\exists j : V_j^{\pi'} > V_j^{\pi}\right). \tag{13}$$

3.2 Energy Consumption Estimation in YAFS

Because the simulation environment (YAFS) does not natively provide energy consumption metrics, I implemented a custom calculation procedure based on each fog node's power consumption parameter $WATT$ and the task's service time. For a task executed on node i, the energy consumption is computed as:

$$E_i = P_i \times T_i \tag{14}$$

where:

- P_i: Power rating (Watt) of node i, defined in the topology configuration.
- T_i: Service time (in seconds) recorded for the task at node i.

The average energy consumption across recent tasks is then given by:

$$E_{avg} = \frac{\sum_{i=1}^{N} E_i}{N} \tag{15}$$

where N is the number of tasks processed in the considered time window.

Topology Configuration. Each fog node is assigned a power rating according to the system topology. For example:

- Fog nodes 1–10: power ratings between 31W and 60W.
- IoT devices: 5W (low power).
- Cloud server: 20W (medium power).

These values are specified in the topology configuration file (`topologyDefinition.json`) as shown below:

```
{
  "entity": [
    { "id": 1, "model": "Fog-server", "WATT": 33 },
    { "id": 2, "model": "Fog-server", "WATT": 43 },
    ...
    { "id": 10, "model": "Fog-server", "WATT": 60 }
  ]
}
```

Implementation Details. To ensure reliable energy estimation, I applied the following measures:

- **Service time filtering**: Only tasks with service time $T_i < 1.0$ seconds are considered, excluding abnormal values.
- **Default values**: If a node lacks a defined $WATT$ parameter, a default of 40W is used.
- **Minimum bound**: A minimum value of 0.1 Joules per task is enforced to avoid unrealistically low values.

Example. Consider a fog node with $P = 33$W and an average service time of $T = 0.003$ seconds. The energy consumption per task is:

$$E = 33 \times 0.003 = 0.099 \text{ J}. \tag{16}$$

If 10 tasks are processed with similar service times, the total consumption is 0.99 J, corresponding to an average of 0.099 J per task.

3.3 System Flow Chart

Figure 1 depicts an workflow of the proposed system. It begins the main simulation script (main.py) in the YAFS with JSON format configuration files. Once initialized, it prompts the selection of a load balancing algorithm. The proposed Pareto-based Deep Q-network (PDQN) algorithm compares to two traditional approach - Round Robin (RR), Weighted Round Robin (WRR). During the simulation, RR and WRR selection process is deterministic and cycles through the available fog nodes. Whereas, PDQN is observed and transformed into a real-time state representation, which is passed to the neural network for inference. The PDQN agent then selects an action (fog node assignment) based on its

learned policy and Pareto front optimization between latency and energy consumption. At the end of the simulation, result analysis is performed. This includes YAFS simulation's default link and results CSV, and the Pareto front to visualize the trade-off between latency and energy consumption.

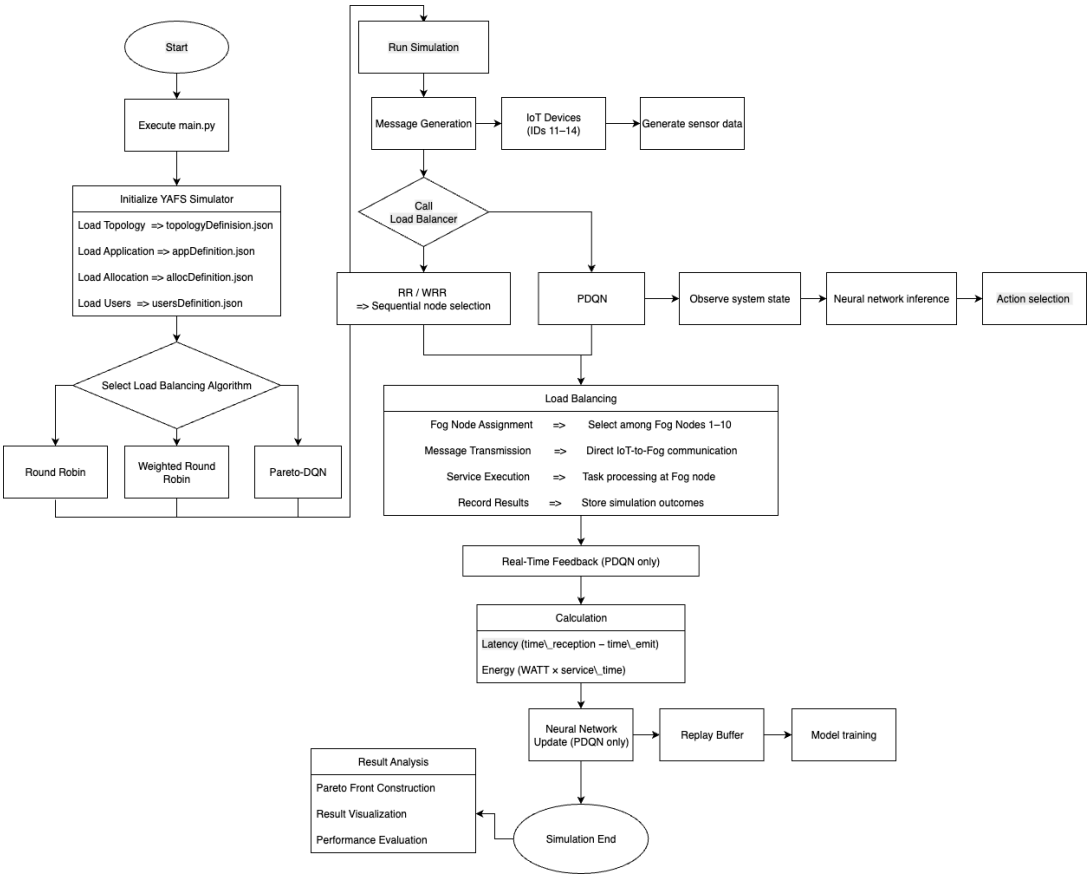

Figure 1: System Flow chart

4 Design Specification

4.1 System Architecture

The system is a three-layered hierarchical architecture, which consists of cloud, fog, IoT device layers. This three-tier architecture has widely adopted in the IoT system, because it can meet a need to real-time or time-sensitive environment. Unlikely the cloud-only implementation, integrating with cloud-fog layer acts a complementary role in the IoT section. Fog computing brings the resources near the edge of the network, so that the latency can significantly decrease. In the other word, cloud computing offers resources only for large computation power needs workload. However, Fog computing is the extended version of cloud servers. Thus the cloud layer cannot be replaced entirely with fog. It must co-exist to assist each other whenever necessary. [5] The different elements of the architecture are shown in Figure 2

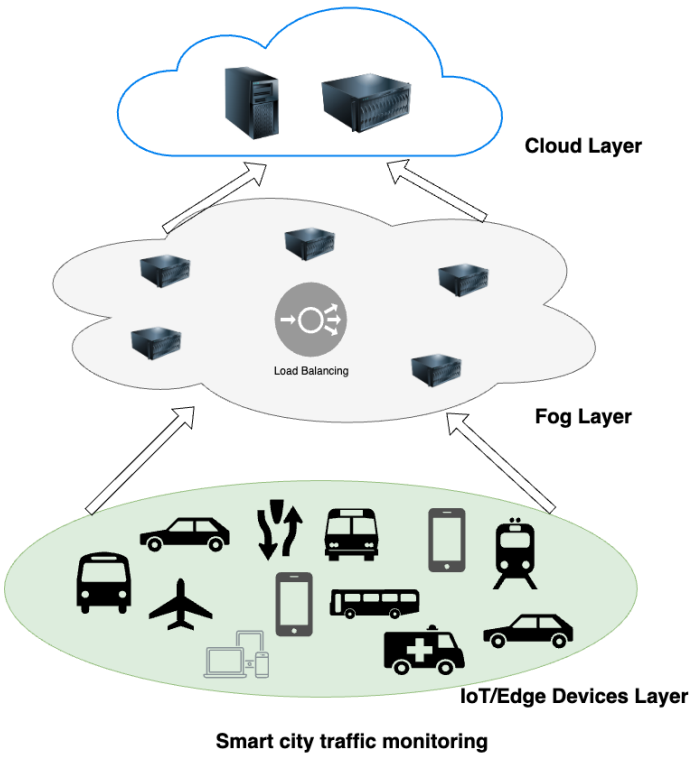

Figure 2: Architecture Diagram

1) The IoT/Edge layer: It represents IoT devices and edge of the network and is comprised primarily of intelligent devices with sensing properties along with the end-users who ultimately consume the applications. The devices generate data that initiates the processing workflow. Latency upon users receiving the result messages is considered in this work to be negligible because the work does not focus on upstream resource handling and processing but rather on end-user delivery latency. All the data that is created is directed upwards to the upper layers by the Fog layer to be calculated. Especially, the paper assumes smart city traffic monitoring system, which means these IoT devices relative to traffic data such as train, bus and etc.

2) The Fog layer: It consists of a fog node network that is randomly distributed and interconnected. The nodes in the network are equipped with computing hardware such as gateways, routers, and switches. Even though the fog nodes offer an equivalent computation and storage to cloud infrastructure, their hardware functionality is largely limited. Nevertheless, their proximity to sources of data makes them apt for applications involving latency-sensitive processing. The proposed Pareto-DQN load balancing approach is developed in the fog layer to distribute task dynamically.

3) The Cloud layer: It is emulated as a centralized data center providing enormous computing and storage capabilities for running large-scale IoT applications. Communication between the cloud and other system parts is facilitated through a gateway that connects it to the fog layer. Operations that cannot be supported efficiently in the fog environment because of constrained resources are offloaded to this cloud infrastructure.

4.2 Simulation Data

The simulation is based on a Smart City Traffic Monitoring use case that processes heterogeneous traffic sensor data in real-time. The dataset is defined in JSON format include computational modules, messages, and transmission mappings between IoT devices and fog nodes. Although its dataset is not real-world data but simulator generated data, it is represented to traffic monitoring data.

The workload consists of four distinct message types :
M.SensorDataFast, M.SensorDataMedium, M.SensorDataSlow, and M.TrafficAnalysis
These simulate traffic monitoring tasks show varying computational size and network demands. These different type of messages stand for diverse sensor data. Each messages has a different payload sizes ranging from 1,024 bytes to 5,120 bytes and instruction counts from 2 million to 20 million instructions, which means diverse processing requirements from high-frequency sensor reading to computationally intensive traffic analysis tasks. The heterogeneity in task sizes is critical for evaluating the efficiency of the proposed load balancing strategy, so that it could reflect realistic fluctuations in IoT-generated workloads.

There are tree distinct study cases, small, medium, and large scale, to comprehensively evaluate the scalability and adaptability of the proposed load balancing approach. Each study case is defined externally in a JSON file (topologyDefinition.json) and included the number of fog nodes, CPU processing capability, and network bandwidth. That JSON format file contains the IDs, types, and connectivity of all nodes in the simulation. The cloud server is always assigned as id:0 and is excluded from the fog node count. These three separate study cases are represented to scalability analysis by varying the number of fog nodes from 10 to 30, which means the system's performance can be assessed under low, moderate, and hight network density conditions. It is also represented to heterogeneous resource allocation and network capacity variation with the fog CPU range and the bandwidth range to emulate realistic scenarios. This design ensures that changes to the network structure or hardware specifications can be made without altering the system code, moreover, improving flexibility and reproductibility.

4.3 Dataset for PDQN training

In contrast to the simulation input dataset, the dataset for DQN training is not pre-collected but dynamically generated during the simulation. The Pareto-DQN agent continuously interacts with the YAFS environment, and each interaction generates a state transition stored in an experience replay buffer. Each state is represented as an 8-dimensional vector including CPU utilization, queue length, energy consumption, current latency, moving average latency, success rate, activity level, and efficiency.

Each action corresponds to a selected fog node, while the reward is defined as a two-dimensional vector capturing latency and energy objectives. The replay buffer has a maximum capacity of 20,000 transitions, and mini-batch updates are performed with a batch size of 32 to balance learning stability and computational efficiency.

Traffic is generated by IoT devices that submit approximately 17,500 tasks per device, in accordance with the configured λ rates. A typical simulation runs for 5,000

time units, during which the Pareto-DQN agent makes allocation decisions every 20 steps. This process dynamically produces between 15,000 and 20,000 transitions depending on the simulation length and the number of IoT devices (four in our setup). Each episode consists of 100 message decisions, and several hundred episodes (up to 700) can be generated in a complete run.

All metrics are normalized to the [0,1] range using study case-specific bounds (e.g., latency 5–200ms, energy 0.1–5.0J), ensuring consistent learning across heterogeneous metric scales. This dynamic dataset provides the basis for reinforcement learning and enables the Pareto-DQN agent to adaptively optimize both latency and energy trade-offs.

5 Implementation

5.1 Pareto-DQN Algorithm

The proposed Pareto-DQN (PDQN) agent is implemented in Python using the TensorFlow 2.15 framework. And simulation environment is built on YAFS (Yet Another Fog Simulator), where custom load balancing algorithms and energy models are integrated. The implementation consists of two primary components: the `ParetoDQN_Selection` module, which handles interaction with the environment, and the `ParetoDQN_Agent`, which performs policy training based on multi-objective DQN. The PDQN algorithm is built based on M.Reymond and A.Nowé (2019) [21]

5.1.1 Reward Normalization and Weight Bias

The agent optimizes two conflicting objectives: latency minimization and energy minimization. To balance their influence on learning, we applied objective-specific normalization:

$$\hat{L} = \frac{Latency}{Latency_{max}} \tag{17}$$

$$\hat{E} = \frac{Energy}{Energy_{max}} \tag{18}$$

The normalized reward vector is defined as:

$$r = [-\hat{L},\ -\hat{E}] \tag{19}$$

where negative signs ensure that smaller costs correspond to higher Q-values. To prevent excessive bias toward latency, an *energy bias probability* p_{energy} was introduced. During Pareto-front action selection, the agent chooses an energy-optimal action with probability p_{energy} and a latency-optimal action otherwise.

To handle various parameters with different ranges and scales, the system implements a comprehensive normalization system through the MetricsNormalizer class. This class provides study case-specific normalization ranges for different network scales. The normalization process uses min-max scaling to transform all parameters to the [0,1] range. For example, latency values are normalized using the formula (ms - lat_lo) / (lat_hi - lat_lo), where lat_lo and lat_hi are the lower and upper bounds for the specific study

case. Energy values follow the same pattern with their respective bounds. The system also implements dynamic normalization range adjustment based on the 95th percentile of actual simulation data to prevent saturation and ensure optimal learning performance. The normalization system provides several key advantages: it ensures consistent learning by maintaining all metrics in the same range, enables comparison between parameters with different units and scales, dynamically adapts to actual data distributions, and provides study case-specific optimization. This approach allows the Pareto-DQN algorithm to effectively balance multiple objectives while maintaining stable learning performance across different network scales and configurations.

5.1.2 TensorFlow/Keras Model

The proposed system is implemented the PDQN agent in TensorFlow/Keras. TensorFlow is an end-to-end open source platform for machine learning that provides stable Python and C++ APIs. Each forward pass receives a concatenated input of the flattened system state, the discrete action index, and a scalarization parameter $p \in [0, 1]$. The network regresses a *2D Q-vector* corresponding to the two objectives (latency and energy). The architecture is:

$$\text{Dense}(128, \text{ReLU}) \to \text{BatchNorm} \to \text{Dropout}(0.2) \to$$
$$\text{Dense}(64, \text{ReLU}) \to \text{BatchNorm} \to \text{Dropout}(0.2) \to$$
$$\text{Dense}(32, \text{ReLU}) \to \text{Dense}(16, \text{ReLU}) \to \text{Dense}(2, \text{Linear}).$$

It uses Huber loss and an lagacy Adam (Adaptive Moment Estimation) optimizer with gradient clipping (`clipnorm`=1.0). The learning rate follows an exponential decay schedule. To stabilize training, we maintain a target network and apply a Polyak/soft update with coefficient $\tau = 0.01$. For Apple Silicon environments, we enable GPU memory growth when available and fall back to CPU otherwise.
ReLU is used in hidden layers for computational efficiency and gradient stability, while the output layer uses linear activation to represent unbounded Q-values.

5.1.3 Experience Replay with Priority Sampling

Transitions $(s, a, \mathbf{r}, s', \text{done}, p)$ are stored in a fixed-size replay buffer. These are computed a TD error on the 2D target

$$\mathbf{y} = \mathbf{r} + \gamma\, \mathbf{q}^\star(s', p),$$

and use its ℓ_2 norm (with a small energy-sensitive factor) as a priority weight. Mini-batches are sampled according to these priorities. The target $\mathbf{q}^\star(s', p)$ is obtained by: (i) predicting the 2D Q-vector for every action at s' with the *target* network, (ii) extracting the Pareto non-dominated set among these vectors, and (iii) averaging the non-dominated vectors to yield a stable target.[1]

5.1.4 Pareto Front–Based Action Selection

The algorithm combines ϵ-greedy exploration with a Pareto front filter:

[1] Averaging the Pareto set is a simple and robust target aggregation; other set-to-point reductions are possible.

1. For each action a, it samples multiple scalarization parameters $p \sim \mathcal{U}(0,1)$ and obtains a cloud of 2D Q-points $\{\mathbf{Q}(s,a,p)\}$.

2. It computes the non-dominated (Pareto) subset over the union of points from all actions.

3. it applies an *energy-biased band* rule inside the Pareto set: with probability π_{energy}, select a candidate that is within a small band of the best energy performance; otherwise, select one favoring latency. This banded tie-break avoids collapsing onto a single objective and adapts online to recent reward trends.

When the recent energy penalty grows, we bias the p sampling and the band-selection toward energy-optimal regions to accelerate exploration on that axis.

5.1.5 Final Selection Policy

The final policy is:

1. With probability ϵ, pick a random action (exploration).

2. Otherwise (exploitation), form multi-p Q-samples for all actions, extract the global Pareto set, and choose within an energy-biased band (or, alternatively, latency-biased) according to a tunable probability π_{energy}.

$$a = \begin{cases} \text{random action}, & \text{with probability } \epsilon \\ \arg\max\limits_{a'} HV(\text{Pareto}(s,a'),r), & \text{otherwise} \end{cases} \tag{20}$$

5.1.6 Time Complexity

The proposed Pareto-DQN has higher computational complexity compared to heuristic approaches. In the inference stage, each decision requires evaluating the Q-network for all available actions and for multiple Pareto scalarization samples. If A denotes the number of fog nodes, d the state dimension per node, and m the number of scalarization samples, then one decision involves approximately $O(A \cdot m \cdot F)$ operations, where F is the cost of a forward pass through the neural network.
After these evaluations, a Pareto front must be extracted. In the current implementation, this is performed using pairwise comparisons with complexity $O((A \cdot m)^2)$, although this can be improved to $O(A \cdot m \log(A \cdot m))$ by applying a sorting-based non-dominated sorting algorithm.
During training updates, a batch of size B requires computing target Q-values for all actions, resulting in a cost of approximately $O(B \cdot (A \cdot F + A^2))$. These additional computations are necessary to capture the multi-objective trade-offs between latency and energy. This is significantly heavier than the constant-time O(1) complexity of RR and WRR, which simply rotate among nodes, and also higher than traditional meta-heuristics such as GA or PSO, which scale as $O(I \cdot P \cdot A)$ with the number of iterations I and population size P. However, meta-heuristics are typically run offline and must be re-executed whenever the environment changes, while PDQN learns online and adapts continuously to non-stationary fog conditions.
In practice, although PDQN requires more computation per decision, the overhead is manageable for fog networks up to 30 nodes, and can be reduced further by batching

predictions and using more efficient Pareto set extraction. This trade-off makes PDQN more costly than simple heuristics but far more suitable for dynamic, latency-sensitive environments where multi-objective optimization is necessary.

Algorithm 1 Pareto-DQN Based Load Balancing in Fog Computing

Input: Fog nodes $N = \{1, 2, \ldots, 10\}$, State space $S \in \mathbb{R}^{N \times 8}$, Action space $A = \{0, 1, \ldots, 9\}$
Output: Learned optimal policy π^* for latency-energy trade-off
Initialize NDt-Network θ_Q, Target Network θ'_Q
Initialize replay buffer $\mathcal{B}$, $\epsilon \leftarrow 0.5$, $\epsilon_{\text{min}} \leftarrow 0.05$, $\epsilon_{\text{decay}} \leftarrow 0.985$
$step \leftarrow 0$
while simulation is running **do**
    $S_t \leftarrow$ Collect node states $(CPU_i, Queue_i, Energy_i, Latency_i, MA\_Latency_i, SuccessRate_i, Activity_i, Efficiency_i)_{i=1}^{n}$
    $s \leftarrow Flatten(S_t) \in \mathbb{R}^{n*8}$
    Action Selection
    if random() $< \epsilon$ **then**
        $a \leftarrow$ Random action from A
    else
        $a \leftarrow \arg\max_{a'} R(s, a')$
    end if
    Route message to fog node a
    Reward Calculation
    $Latency \leftarrow (time_{recv} - time_{emit}) \times 1000$
    $Energy \leftarrow WATT_a \times service\_time$
    $r \leftarrow [-Latency, -Energy]$
    Observe next state s'
    Store (s, a, r, s') in $\mathcal{B}$
    if $|\mathcal{B}| \geq batch\_size$ **then**
        Sample minibatch from $\mathcal{B}$
        Train θ_R and θ_Q via MSE loss
        Every C steps: update $\theta'_Q \leftarrow \theta_Q$
    end if
    $\epsilon \leftarrow \max(\epsilon_{\text{min}}, \epsilon \cdot \epsilon_{\text{decay}})$
    $step \leftarrow step + 1$
end while
return π^*

5.2 YAFS Simulator

YAFS (Yet Another Fog Simulator) is a simulation Python library for Cloud, Edge or Fog Computing ecosystems. It is designed to analyze the design of applications and incorporates strategies for placement, scheduling and routing. YAFS is built upon SimPy, a discrete-event simulator (DES) library that provides robust functionalities for defining processes such as task execution, message transmission and shared resources (e.g., queues, network links).

YAFS network allows the modeling of the communication links among machines, end-devices and users. It also allows users to customize placement, scheduling and routing

algorithms. It provide custom processes such as the movement of the workload sources, generation of network. YAFS also supports the importation of scenario definition from JSON-format files. It is more user-friendly and a better starting point to understand the simulation. Furthermore, its performs automated CSV-based logging of two type of event - workload generation and computation, and link transmissions. Theses results could be analyzed and expanded complex analyses using R, Python or another language.
YAFS is defined by six main classes: core, topology, selection, placement, population, and application. Core manages the overall simulation and integrates the other components. Topology represents the network structure using a graph-based model. Placement handles the allocation of software modules to nodes. Selection implements the policy for service discovery and routing. Population controls workload generation and source distribution. And Application defines the application logic as a graph of modules and message flows.
The Core class is responsible for running the simulation by controlling the processes of all the simulation through the lifecycle and integrating custom policies like selection, placement, population. It also gathers raw event data - message transmission, execution - for further analysis through the Stats and Metrics classes, which computes metrics including average response time, link latency, and resource utilization.
YAFS represents the infrastructure as a graph in which the nodes are fog devices, data centers, or clusters and the edges are the network links. The graph structure is implemented using the NetworkX library that allows for the use of complex network analysis as centrality, community detection, and flow algorithms. Network topologies can be imported in different formats such as JSON, GML, GraphML or even from real-world datasets such as CAIDA or BRITE. Each node requires at least three attributes: ID, Instruction per time unit (IPT), and RAM. Additional custom attributes can represent features such as storage capacity, power consumption, virtualization, or microservices. Links between nodes are defined with at least two attributes: Bandwidth (BW) and Propagation delay (PR). A simulation contains one Topology instance which contains one or more hosted applications with their respective policies for selection, placement population.
The application model in YAFS follows the Distributed Data Flow (DDF) model, similar to the one used in iFogSim. An applications expressed as a Directed Acyclic Graph (DAG). Nodes represent application modules like services. And edges represent dependencies or messages between modules. Each module acts a computation on incoming data and each message has two mandatory attributes such as instructions and bytes.
Unlike iFogSim, YAFS provides a more flexible and reusable application definition model. It uses "message" instead of dependency, and messages are treated as transferable, reusable entities across applications. Modules are only triggered by receiving a message. [18]

6 Evaluation

The study assumes a Smart City Traffic Monitoring System that collects traffic sensing data and analyzes the data. The primary goal is a dynamic load balancing among fog nodes. The system evaluates three case studies are divided by scale. Every case studies defined by JSON format, and simulated in YAFS simulator that developed in Python version 3.9.7. The simulations were performed on an Apple M1 laptop with 8 GB RAM. The simulation is performed in the small, medium, and large scale, which uses same users

and application pipeline. Each study case has different fog nodes and link and compare to baseline load balancing algorithm - RR, WRR. Though RR and WRR algorithm are straightforward, in some of the studies in fog computing and edge computing, they have been utilized as common benchmarks since they work well for task load balancing.

Table 2: Simulation Configuration

Study Case	**Fog Nodes**	**Tasks/IoT**	**Fog CPU Range**	**Bandwidth Range (Mbps)**
Small Scale	10	17,500	1.0–2.0	5–6
Medium Scale	20	17,500	2.2–4.2	7–8
Large Scale	30	17,500	5.0–7.5	9–10

6.1 Case Study 1: Small Scale

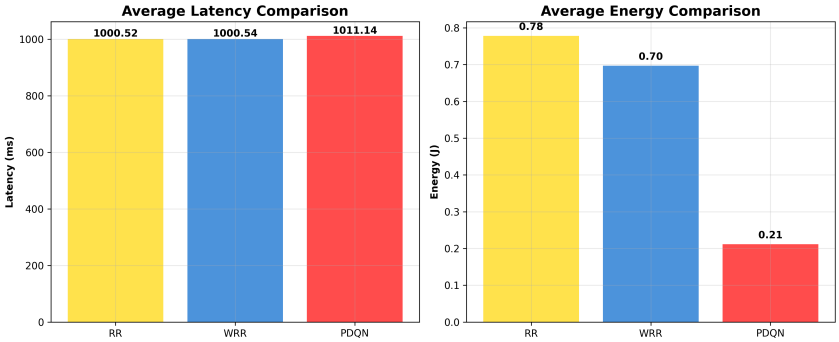

Figure 3: small scale: comparison

Results and Analysis. In the small-scale simulation, the PDQN-based load balancing algorithm achieved a substantial reduction in energy consumption, although maintaining a latency level compare to baseline approaches. An average latency of PDQN is recorded slightly higher than RR and WRR with 1011.14 ms. However, the energy consumption dropped significantly to 0.21 J, which means a 73% reduction compared to RR (0.78 J) and a 70% reduction compared to WRR (0.70 J). These results show that in low-load environment, the PDQN is capable of achieving major energy savings without a noticeable latency penalty.

6.2 Case Study 2: Medium Scale

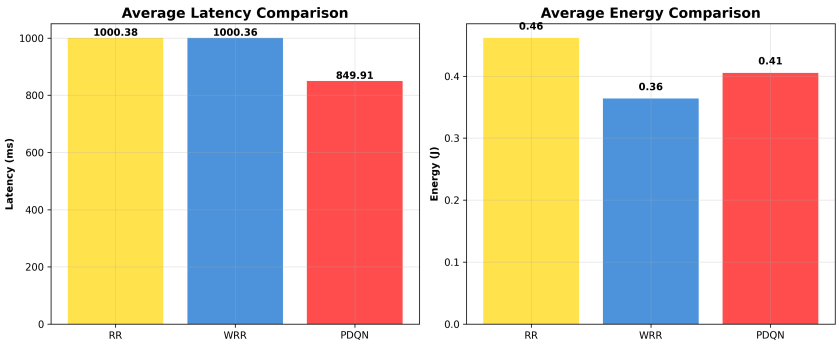

Figure 4: Medium scale: comparison

Results and Analysis. In the medium-scale, the PDQN algorithm shows that effectively balances the trade-off between latency and energy. While maintaining its advantage in latency reduction, the energy preserves competitive its efficiency. PDQN achieved an average latency of 849.91 ms that represents a 15% improvement over RR (1000.38 ms) and WRR (1000.36 ms). The energy consumption of 0.41 J, which is lower than RR (0.46 J) but a bit higher than WRR (0.36 J). These results highlight that it particularly suitable for scenarios where latency-sensitive operations are prioritized.

6.3 Case Study 3: Large Scale

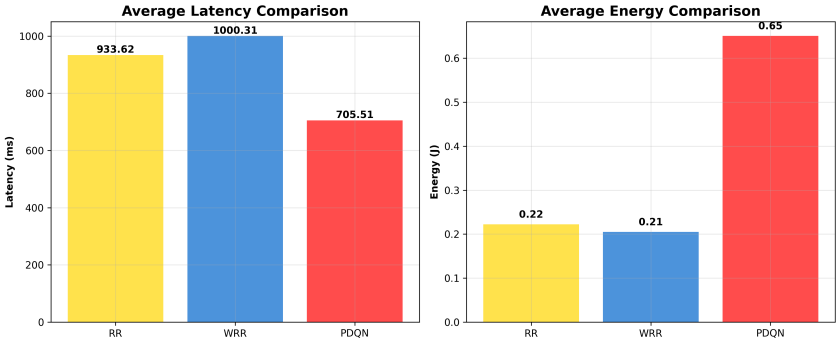

Figure 5: Large scale: comparison

Results and Analysis. In the large-scale experiments, the PDQN algorithm achieved that lowest latency, averaging 705.51 ms, which means a 24.% improvement over RR (933.62 ms) and a 29.5% improvement over WRR (1000.31 ms). Whereas, this significant latency gain came at the cost of 0.65 J in the highest energy consumption. It is a prominent figure compare to RR (0.22 J) and WRR (0.21 J). It indicates that PDQN is highly effective in minimizing latency under heavy load conditions but requires additional optimization to address its increased energy usage. This trade-off represents that, in large-scale deployments, the proposed PDQN is particularly suited for latency-critical applications, while further hyper tuning or hybrid approaches might be necessary to get balanced energy efficiency.

6.4 Discussion

Overall, the results indicate that PDQN is particularly effective for latency-critical applications, especially in medium-to-large scale system. Although the system adopted an energy-weighted reward function, the inherently small magnitude of the estimated energy values made it challenging to optimize effectively. Therefore, for scenarios where energy efficiency is equally important, further tuning or hybrid strategies that balance the two objectives may be required. These findings are in line with the nature of multi-objective optimization, where achieving an optimal balance between latency and energy consumption remains a complex but essential challenge. Optimization in one aspect would degrade the other, requiring prudent balancing strategies.

During the development process, several challenges were encountered. First, limitation in the simulation environment and the constraints of the provided framework made it difficult to calculate certain metrics with high accuracy, particularly for energy estimation. The figure of energy consumption was almost same due to simulator features. Additionally, considering realistic execution times, it was not feasible to extend DQN training indefinitely. So that the network design was scaled down from the initial plan to allow for timely execution. Nevertheless, the result is that the suggested methodology is well-suited for latency-sensitive applications. Following subsequent hyperparameter tuning, there is strong potential to achieve significant improvements in both latency and energy efficiency simultaneously.

Scalability is an another limitation. Even though the experiments covered small (10 nodes), medium (20 nodes), and large (30 nodes) scenarios, these are still relatively much smaller than usual real-world smart city deployments, which is usually 300–500 nodes. In order to facilitate rapid iteration, debugging, and tuning of PDQN within practical execution times, the smaller scale was actually intentionally selected. During development, such problems as repeated CSV access, unstable normalization, and cases where the model optimized latency at the expense of energy required the use of caching and design adjustment even at 30 nodes. So-called "large scale" is really not very large and has to be read between the line. However, it would be expected that PDQN's adaptive decision making would show even greater advantages as the scale increases.

7 Conclusion and Future Work

Fog computing has emerged as a key paradigm for low-latency applications. In other hand, reducing energy consumption in computational infrastructure is directed toward is now of utmost priority. Thus, this paper proposed a Pareto-DQN load balancing approach to find a trade-off between reducing latency and energy consumption. The system state was defined using real-time metrics collected from each fog node, including CPU utilization, queue length, latency, and energy consumption and etc, which were normalized to ensure stable learning. Instead of a scalar reward, it applies a vector reward value. The PDQN leveraged Pareto front-based action selection to identify non-dominated solutions during training. This approach enabled the agent to maintain diversity in its decision space and adapt its routing strategies in terms of multi-objective trade-offs. Evaluation results across small, medium, and large scale scenarios demonstrated that the proposed PDQN model generally outperformed traditional scheduling algorithms such as RR and

WRR, particularly in latency-sensitive environments. However, energy optimization remained challenging due to the small magnitude of measured values and the inherent conflict between latency and energy objectives.
The novelty of PDQN lies not just in applying deep reinforcement learning but also in its incorporation of Pareto-front based action selection internally within the learning process. Unlike RR and WRR, which are static rule-based allocation, PDQN continuously supports heterogeneous conditions and jointly optimizes latency and energy. This establishes a closer balance between conflicting objectives, demonstrating adaptability that is absent from baseline methods.

In future work, more accurate energy estimation methods will be explored to improve optimization effectiveness, along with extended simulation environments incorporating heterogeneous workloads, mobility, and realistic network topologies. Furthermore, advanced hyperparameter tuning and larger network architectures could enhance convergence and improve the balance between latency and energy performance. Moreover, the integration of online learning capabilities into the PDQN framework can potentially enable real-time adaption within real-world live fog computing deployments, further boosting resilience in ultra-dynamic environments. Finally, expanding the evaluation to larger-scale deployment (hundreds of fog nodes) and adding comparisons with state-of-the-art algorithms will help validate the scalability and feasibility by getting closer to real-world deployments.

References

[1] Ala'anzy, M. A., Zhanuzak, R., Akhmedov, R., Mohamed, N. and Al-Jaroodi, J. [2024]. Dynamic load balancing for enhanced network performance in iot-enabled smart healthcare with fog computing, *IEEE Access* **12**: 188957–188975.

[2] Ali, S. and Alubady, R. [2023]. Rwrr: Remind weighted rounding robin for load balancing in fog computing, *2023 7th International Symposium on Innovative Approaches in Smart Technologies (ISAS)*, pp. 1–7.

[3] Alsadie, D. [2024]. A comprehensive review of ai techniques for resource management in fog computing: Trends, challenges, and future directions, *IEEE Access* **12**: 118007–118059.

[4] Apat, H. K., Nayak, R. and Sahoo, B. [2023]. A comprehensive review on internet of things application placement in fog computing environment, *Internet of Things* **23**: 100866.
URL: *https://www.sciencedirect.com/science/article/pii/S2542660523001890*

[5] Asghar, A., Abbas, A., Khattak, H. A. and Khan, S. U. [2021]. Fog based architecture and load balancing methodology for health monitoring systems, *IEEE Access* **9**: 96189–96200.

[6] Belkout, N. E., Zeraoulia, K., Shahzad, M. N., Liu, L. and Yuan, B. [2022]. A load balancing and routing strategy in fog computing using deep reinforcement learning, *2022 International Conference on Electrical, Computer and Energy Technologies (ICECET)*, pp. 1–8.

[7] Beraldi, R., Canali, C., Lancellotti, R. and Mattia, G. P. [2020]. A random walk based load balancing algorithm for fog computing, *2020 Fifth International Conference on Fog and Mobile Edge Computing (FMEC)*, pp. 46–53.

[8] Chang, S. and Lutfiyya, H. [2024]. Adaptive agent-based load balancing in fog computing, *2024 IEEE 10th World Forum on Internet of Things (WF-IoT)*, pp. 1–7.

[9] Deepak, Upadhyay, M. K. and Alam, M. [2024]. Load balancing techniques in fog and edge computing: Issues and challenges, *2024 IEEE International Conference on Computing, Power and Communication Technologies (IC2PCT)*, Vol. 5, pp. 210–215.

[10] Ebrahim, M. and Hafid, A. [2023]. Resilience and load balancing in fog networks: A multi-criteria decision analysis approach, *Microprocessors and Microsystems* **101**: 104893.
URL: *https://www.sciencedirect.com/science/article/pii/S0141933123001370*

[11] Esan, A. B., Shareef, H. and Saeed, N. [2023]. Multi-objective energy management system for isolated solar microgrids using pareto q learning, *2023 IEEE International Conference on Energy Technologies for Future Grids (ETFG)*, pp. 1–6.

[12] Hayes, C. F., Rădulescu, R., Bargiacchi, E., Källström, J., Macfarlane, M., Reymond, M., Verstraeten, T., Zintgraf, L. M., Dazeley, R., Heintz, F., Howley, E., Irissappane, A. A., Mannion, P., Nowé, A., Ramos, G., Restelli, M., Vamplew, P. and Roijers, D. M. [2022]. A practical guide to multi-objective reinforcement learning and planning, *Autonomous Agents and Multi-Agent Systems* **36**(1).
URL: *http://dx.doi.org/10.1007/s10458-022-09552-y*

[13] Hazra, A., Rana, P., Adhikari, M. and Amgoth, T. [2023]. Fog computing for next-generation internet of things: Fundamental, state-of-the-art and research challenges, *Computer Science Review* **48**: 100549.
URL: *https://www.sciencedirect.com/science/article/pii/S1574013723000163*

[14] Houses of the Oireachtas Library & Research Service [2025]. The future of data centres in ireland, `https://www.oireachtas.ie/en/how-parliament-is-run/houses-of-the-oireachtas-service/library-and-research-service/research-matters/2025-03-20-the-future-of-data-centres-in-ireland/`. Accessed: 2025-07-22.

[15] Jasim, A. M. and Al-Raweshidy, H. [2024]. An adaptive sdn-based load balancing method for edge/fog-based real-time healthcare systems, *IEEE Systems Journal* **18**(2): 1139–1150.

[16] Kaur, H., Malik, S., Baggan, V. and Harnal, S. [2024]. Enhanced k-means clustering of tasks and virtual machines for load balancing in fog environment, *2024 13th International Conference on System Modeling Advancement in Research Trends (SMART)*, pp. 565–570.

[17] Kumar, N. J., Premkumar, R., Visuwasam, L. M. M., Arjunan, G., Yuyaraj, G. and Kumar, C. T. [2025]. Hybrid k-means and firefly algorithm-based load balancer for

dynamic task scheduling in fog computing for postoperative healthcare systems, *2025 International Conference on Advanced Computing Technologies (ICoACT)*, pp. 01–06.

[18] Lera, I., Guerrero, C. and Juiz, C. [2019]. Yafs: A simulator for iot scenarios in fog computing, *IEEE Access* **7**: 91745–91758.

[19] Nayak, A., Tripathy, S. S., Bebortta, S. and Tripathy, B. [2024]. An intelligent study towards nature-inspired load balancing framework for fog-cloud environments, *2024 IEEE International Conference for Women in Innovation, Technology Entrepreneurship (ICWITE)*, pp. 168–173.

[20] Pani, L., Singh, K., Dutta, A., Misra, C. and Roy, R. [2022]. A roadmap for energy saving using dynamic load balancing in cloud and fog architecture, *2022 10th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO)*, pp. 1–5.

[21] Reymond, M. and Nowe, A. [2019]. Pareto-dqn: Approximating the pareto front in complex multi-objective decision problems, *Proceedings of the Adaptive and Learning Agents Workshop 2019 (ALA-19) at AAMAS*. 2019 Adaptive Learning Agents (ALA) workshop: Workshop of the AAMAS conference ; Conference date: 13-05-2019 Through 14-05-2019.
URL: *https://ala2019.vub.ac.be*

[22] S, T., M, G., Sidhu, S., Sivakumar, A. and M, S. [2024]. Performance investigation of load balancing algorithms in fog computing, *2024 Third International Conference on Electrical, Electronics, Information and Communication Technologies (ICEEICT)*, pp. 1–7.

[23] Seraj, Y., Fadaei, S., Safaei, B., Javadi, A., Monazzah, A. M. H. and Hemmatyar, A. M. A. [2024]. Limo: Load-balanced offloading with mape and particle swarm optimization in mobile fog networks, *2024 5th CPSSI International Symposium on Cyber-Physical Systems (Applications and Theory) (CPSAT)*, pp. 1–8.

[24] Sing, R., Bhoi, S. K. and Panigrahi, N. [2023]. A load balancing algorithm for cloud-fog-based iot networks using bald eagle search optimization, *2023 1st International Conference on Circuits, Power and Intelligent Systems (CCPIS)*, pp. 1–6.

[25] Tahmasebi Pouya, N. and Agha Sarram, M. [2021]. Blind load-balancing algorithm using double-q- learning in the fog environment, *2021 11th International Conference on Computer Engineering and Knowledge (ICCKE)*, pp. 278–283.