National College of Ireland

A Reinforcement Learning Based Load Balancing Approach for Energy Efficiency and QoS Optimization in Live Cloud Streaming

MSc Research Project
MSc in Cloud Computing

Ganesh Ram Shanmugam
Student ID: 23244275

School of Computing
National College of Ireland

Supervisor: Aqeel Kazmi

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Ganesh Ram Shanmugam
Student ID:	23244275
Programme:	MSc in Cloud Computing **Year:** 2025
Module:	MSc Research Project
Supervisor:	Aqeel Kazmi
Submission Due Date:	15-09-2025
Project Title:	A Reinforcement Learning Based Load Balancing Approach for Energy Efficiency and QoS Optimization in Live Cloud Streaming
Word Count:	7349 **Page Count:** 20

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: Ganesh Ram Shanmugam

Date: 15-09-2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	Yes
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	Yes
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.	Yes

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):	

A Reinforcement Learning Based Load Balancing Approach for Energy Efficiency and QoS Optimization in Live Cloud Streaming

Ganesh Ram Shanmugam
23244275

Abstract

Traditional load balancing methods work with fixed rules or set limits, which makes it difficult to handle unpredictable workloads in real time. This results in high latency, resource wastage, and a poor user experience. To address these challenges, this research focused on using Reinforcement Learning (RL) to make smarter autoscaling decisions in live cloud streaming environments. Two RL methods—Proximal Policy Optimization (PPO) and Deep Q-Network (DQN)—were used to dynamically allocate virtual machine (VM) resources based on live workload metrics. The live cloud streaming simulation scenario was developed using the CloudSim Plus tool with Java. Three different types of video streaming scenarios (light, medium, and heavy workloads) were simulated to collect the dataset. The standard Gym API was used to design a compatible environment in Python for training RL agents with the synthetic dataset generated from the simulation. The trained model was integrated with a prediction server to make real-time decisions by selecting the best VM configurations during workload execution. For evaluation, three types of workloads were simulated, and the RL agents' results were compared with a random autoscaling model. The results showed that both RL models, PPO and DQN, significantly outperformed the random baseline, reducing latency by 74%, saving energy by 67%, and increasing throughput by 88%. RL mainly improved response speed and data delivery, while only minor improvements were observed in jitter. These results demonstrate the potential of RL-based load balancing as an energy-efficient strategy for live cloud streaming without affecting the QoS for users.

Keywords: Cloud computing, Reinforcement Learning, PPO, DQN, CloudSim Plus, Quality of Service, Live video streaming.

1 Introduction

1.1 Background and Motivation

Nowadays, cloud computing is the most widely used technology for delivering scalable and on demand services to advanced applications (Cisco, 2022). Among all, live cloud streaming platforms use this technology more, where Quality of Service (QoS) is much important (Cisco, 2022). These services will always receive unpredictable workloads, different numbers of viewers and require strict service level agreements (SLAs) to ensure low latency, high throughput, minimal jitter, and low energy use (Li et al., 2020). Traditional load balancing methods use simple rules or fixed limits like checking the CPU use or setting the time

between scaling. Even though these methods are simple to implement, they will react slowly to real time changes which will lead to not enough resource allocation during peak loads and waste of energy while the demand is low (Kaur and Singh, 2022). Due to these disadvantages, it affects the user experience and increases the cost than necessary.

1.2 Reinforcement Learning as a Solution

The recent growth in machine learning, specifically in reinforcement learning (RL), provides a smarter way for cloud resource management. RL uses a learning agent to learn from past results and make decisions as per the requirements, which is energy efficiency in this case (Sutton and Barto, 2018). Unlike traditional methods, RL can adjust to sudden changes in traffic and support different types of goals, such as reducing latency and saving energy while maintaining the Quality of Service (QoS).

1.3 Research Problem

Still, the usage of RL for load balancing in live cloud streaming is limited only. Most of the past research work focused on general cloud applications or on a big workload where QoS were not required to maintain. But in the field of live cloud streaming, QoS cannot be compromised. This creates a research gap, as the traditional methods are struggling to make a required real time decision. This leads to the research question below.

Research Question: How can a reinforcement learning based load balancing model minimize energy usage without affecting the QoS in live cloud streaming services?

1.4 Research Aim and Objectives

The main goals of this research project are listed as follows:

- To minimize energy usage while load balancing with RL models without affecting the QoS metrics like latency, jitter, and throughput to ensure the best experience for the user for live cloud streaming applications.
- To compare the PPO and DQN models with a non intelligent random load balancing model to showcase the advantages of using RL models.

To achieve the above aims, the following objectives were defined:

- The system needs to use two widely used RL algorithms, such as Proximal Policy Optimization (PPO) and Deep Q-Network (DQN) to learn the best Virtual Machine (VM) combinations for different types of traffic workloads.
- A custom simulation scenario needs to be developed using the CloudSim Plus tool with Java code.
- Three different types of video streaming scenarios, like light, medium, and heavy workloads should be simulated to collect the dataset.
- The dataset should include key features such as VM MIPS, allocated memory, bandwidth, cloudlet length, and video quality.
- This dataset needs to be used to train the machine learning model in custom RL environments developed in Python, using the standard API Gym to learn the best VM

combinations, and the trained models should be tested in a simulated environment to evaluate how well they perform load balancing under different scenarios.

1.5 Evaluation Summary

Then the performance of the PPO and DQN models was compared with a random load balancing model with different workload scenarios. Latency, energy consumption, jitter, and throughput were used as valuation metrics which are important in live cloud streaming where QoS is important. The results showed that both RL models, PPO and DQN, performed much better than random baseline in terms of latency, energy efficiency, and throughput, by reducing the latency by 74%, energy savings by 67%**,** and increasing the throughput by 88%. Only jitter showed less amount of improvement across all the models. These results showed the potential of RL based load balancing as an energy efficient strategy for live cloud streaming without affecting the QoS for the users.

1.6 Report Structure

The remainder of this report is structured in the following manner. Section 2 contains related work. Section 3 contains the research methodology. Section 4 contains the system design specification. Section 5 contains the implementation details. Section 6 contains the evaluation of the experimental results. Section 7 contains the conclusion and future work.

2 Related Work

2.1 Reinforcement Learning in Cloud Resource Management

Reinforcement learning is being used a lot, for its ability to make decisions well in unpredicted workload conditions. For an example, in Lahande (2023), research on Q-learning algorithms to manage virtual machines and task allocation under different workloads showed better usage of resources compared to the static methods. A similar paper by Esther et al. (2024) provided another approach to implementing an adaptive load balancer using Deep Q-Networks (DQN) trained on Google Cluster data. The results provided in the paper mention that the system responded well to managing resources in sudden workload spikes compared to the traditional load balancing methods. Another study by Zhou et al. (2024) mentions that Deep RL for cloud scheduling works well in handling complex data by showing the performance of non-linear decision making in managing complex state space. A combined DQN model was introduced by Kumar et al. (2023) for managing big data at the enterprise level. Then recently, Chawla (2024) came with an RL based adaptive load balancing system that results in more efficient distribution of workloads in a cloud environment where workloads will change over time.

These recent papers collectively show that RL can provide better performance in dynamic load balancing compared to the traditional methods in cloud environments. These papers focused on big data processing or general cloud-based workloads but did not address anything about the real time live streaming scenarios, which will require not only the efficient

use of resources but also low latency and high throughput for a better user experience. This research will be focussed on addressing these research gaps by developing an RL based load balancer for live cloud streaming workloads without affecting the QoS.

2.2 Load Balancing in Real-Time Cloud Streaming

Some of the traditional load balancing methods like Round Robin, Max-Min, and Min-Min were widely used in cloud computing in the past and currently because they are simple to implement and easy to use compared to the machine learning based techniques (Kaur & Singh, 2022). These methods can be used effectively for stable workloads and cannot be used to handle unpredictable workloads like in a live cloud streaming environment. The traditional load balancing methods were used by Kaur and Kaur (2019) for load balancing in video streaming, but these methods struggled to perform to meet the unpredictable workload. The research papers like Alibaba Cloud (2023) and Yang (2022) gave the ideas and stated the need for scalable and adaptive load balancing in video streaming, but they didn't integrate any RL based models for load balancing, only architecture ideas were presented in the paper. A hybrid reinforcement learning based model was used by Bishoy (2024) for load balancing the unpredicted workloads at the network edge. That setup provided the best performance compared to the traditional methods.
These recent studies show a lot of advancements in load balancing. Still, there is a gap in live cloud streaming load balancing for energy efficiency without affecting the key metrics like latency, throughput and jitter. This research will aim to address this gap.

2.3 Energy Efficiency and QoS Metrics

The efficient usage of energy is always a concern as it can affect the operational cost and environmental impact. A static load balancing method was used by Beloglazov and Buyya (2012), that saved energy by combining the VMs and turning off the unused servers, which helped to achieve up to 30% energy savings. Then more research moved to using RL based methods for resource management for the effective use of energy. An advanced load balancing method using reinforcement learning was designed by Zaman and Grosu (2013), where the RL model is trained to turn off the virtual machines which are not in use without affecting performance much. Then the recent studies like Kahil et al. (2025) and Javed et al. (2024) used RL based models for improving energy usage in AI-driven data centers. Then the other research by Nawaz et al. (2025) explored RL to reduce power consumption by combining VMs.
These recent studies showed how to reduce energy usage but did not mention anything like how QoS metrics such as latency, jitter and throughput will affect while optimizing energy consumption, which are very important metrics for live video streaming. This research will aim to address this gap by designing an RL-based load balancing model for the effective use of energy without affecting the QoS metrics.

2.4 Simulation Tools for RL-Based Cloud Resource Management

After developing the load balancing model, it is important to test the model in simulated environments before using the model in real cloud environments. It is important to choose a simulation tool for evaluating the model in a simulated environment. According to Calheiros et al. (2011), CloudSim is widely used for simulating processes like VM allocation, scheduling, and load balancing because it is flexible to use and can be extended as per need. After that, GreenCloud, another simulation tool developed by Kliazovich et al. (2013), was

used to make an energy analysis on hardware level but cannot be used for real time workload simulation. So, it cannot be used for this research. Another research from Wilk et al. (2018) explained the RL based cloud experiments using CloudSim Plus. OpenAI Gym was integrated with CloudSim in a research paper by Agos Jawaddi and Ismail (2024) to train the RL agents in energy aware cloud environments.
Due to the modern design and proven compatibility with RL models, CloudSim Plus was chosen for this research to simulate the live streaming workloads and for generating the synthetic datasets, and for evaluating the final trained RL models under different types of workloads.

2.5 Comparative Summary of Related Work

Table 1 shows a summary of all the key papers related to this research to make understanding the comparison easier. This table shows the author's name, the method they followed in their study, the type of RL model, if they used any, the evaluation metrics they used in that research and the results and difference of those papers to this research. Most of the reviewed papers used Q learning or DQN and some recent research used PPO also. But these papers mainly focused on general workloads or any other big data workloads. Also, many papers did not include jitter or throughput in their evaluation metrics. This comparison clearly shows that there is a still research gap and the next section will explain those gaps in detail.

Table 1: Comparative Summary of Reviewed Research Papers

Author(s) & Year	Method/Technique	Target Environment	Metrics Considered	Findings
Lahande, (2023)	Q-Learning	VM allocation under changing workloads	Resource usage, execution time	Performed better than fixed rules in unpredictable workloads; lacks real-time streaming focus
Esther et al., (2024)	Deep Q-Network (DQN)	Google Cluster-based task autoscaling	Throughput, resource efficiency	Managed sudden workload spikes better than traditional methods; jitter not considered
Chawla, (2024)	RL Adaptive Load Balancer	Dynamic cloud workloads	Latency, energy usage	Achieved improved workload distribution; QoS metrics like throughput were not included
Bishoy, (2024)	Hybrid RL Model	Edge-based unpredictable workloads	Latency, reliability	Showed better performance than traditional methods; not focused on energy-QoS balance
Zaman and	RL Power	VM energy	Power	Trained models to

Grosu, (2013)	Optimization	management	consumption, SLA	turn off idle VMs; didn't analyze latency or throughput
Agos Jawaddi and Ismail, (2024)	CloudSim + Gym Integration	Simulated cloud workloads for RL training	Power usage, VM allocation	Used RL training in simulated environments; not tested for video streaming

2.6 Research Gap

From seeing the related work from above, it clearly shows that there is significant progress in applying RL models to cloud resource management and energy optimization. But still there is a gap that remains as follows:

- There is not much research made on RL based load balancing for live video streaming workloads, where latency, jitter, and throughput are much important.
- Only limited research made on energy efficient optimization while maintaining or without affecting the Quality of Services (QoS) with the use of RL based load balancing models.
- Lack of design for simulation environments integrated with RL training models mainly for fluctuating workloads.

This research aims to address these gaps by developing a custom RL based load balancing model integrated with CloudSim Plus, for handling the unpredictable streaming workloads with energy efficiency and without affecting the QoS. The RL models like PPO and DQN will be evaluated against a Random baseline to provide the evidence for showing the advantages of using the RL models in this domain.

3 Research Methodology

3.1 Research Process

This research study used an experimental approach, where different types of experiments were carried out in controlled environments to compare the RL based load balancing model to a random auto scaling model as a baseline. The research process was combined into three parts. Simulation-based experiment: To generate repeatable and control different types of workload scenarios. Machine learning model training: Using PPO and DQN models to make the smarter decisions for scaling. Evaluation: Comparing the RL models against a random scaling policy which will choose random actions to see how well the RL models perform against that.

3.2 Dataset Generation

A dataset is needed to train the RL models effectively. To create this dataset, a simulation environment was created, simulating different types of live video streaming workloads using the simulation tool called CloudSim Plus using Java code. In each simulation iteration, one or more Virtual Machines (VMs) will be created with different types of computing power (MIPS), RAM (memory), and bandwidth (internet speed). To simulate the real-world video

processing scenario, cloudlets were used to replicate the streaming tasks with different types of lengths and output sizes. For every iteration, the important metrics in live video streaming scenarios, such as latency, energy consumption, jitter, throughput, VM configuration parameters, and video quality levels were logged in a csv file. The final dataset will have a total of 30,000 different types of workload and resource combinations with 10,000 total iterations.

3.3 RL Environment Design

A custom gym environment was created to help the RL models training. This Python environment consists of three important spaces, such as State space, each row from the dataset like normalized values such as VM compute power, RAM, bandwidth, incoming task length, and video quality level will act as input variables (OpenAI, 2025). In action space, the different types of action the RL can take are based on the values of the input features, such as choosing the VM combinations to run a task (OpenAI, 2025). Three types of actions were used here as low, medium and high resources of VM combination. The reward function indicates the weightage that will balance improving performance and reducing energy use (OpenAI, 2025). Higher weight is given to the latency because it's the most important metric in real time streaming.
This Python environment will be used to interact with the RL models to the generated dataset for training the models to learn how to make smart scaling decisions through trial and error and improve performance across multiple goals like energy efficient without affecting the QoS.

3.4 Model Training

The RL models were trained using Python Stable Baselines3. The two RL models used were PPO (Proximal Policy Optimization) and DQN (Deep Q-Network). PPO is a policy based model which means it will learn a policy directly of taking each action (Schulman et al., 2017). It is good for continuous action spaces (Schulman et al., 2017). DQN is value based model which means it will estimate the future expected action based on the action space, and it will pick the best one (Mnih et al., 2015). It is good for smaller action spaces (Mnih et al., 2015). Each RL model is trained with 20 different random combinations of parameters using the generated dataset in a loop. In each loop, the Gym environment will select the VM combinations based on the predicted actions, and it will get rewards based on the reward function set earlier. After getting the rewards for each training loop, which is the values of QoS metrics like latency, energy and jitter. The best configuration of training values is used, to train the model with more timesteps for both PPO and DQN for better learning of the model. After the training, both models were saved as a zip file for validating the model's performance in the future.

3.5 Real Time Evaluation Procedure

The trained RL model's performance was validated by using the models to make decisions in the CloudSim Plus simulated environment. The predictor server was created using Python for integrating models with a simulated environment to make real time decision making. The simulation setup was created using Java. This simulation setup will randomly select VM combinations as the input states and send the values to the Python predictor server. Then the RL model will make the decision based on the inputs and send back the action to the simulation set up. Then the workload will be executed based on the predicted action. Then

the measured metrics like latency, energy use, jitter, and throughput for the chosen VM combinations will be logged for over 100 simulation runs. Latency will be calculated using the average execution time for each streaming task (ITU-T, 2008). Energy consumption will be calculated by estimated power usage in kWh (ITU-T, 2008). Jitter will be calculated by the variation in response times (ITU-T, 2008). Throughput will be calculated by data delivered per second (Mbps) (ITU-T, 2008). This evaluation was performed for three models, such as the PPO model, DQN model and a random baseline model. Each model was executed with different workload scenario such as light, medium and heavy to collect the comparative results.

4 Design Specification

4.1 System Overview

The created system design is used to support the project objective of RL based load balancing for live cloud streaming workloads without affecting the QoS metrics. This system is designed to simulate the real time resources management of live cloud streaming, where the RL model will choose the best VM based on the incoming workload. Figure 1 shows the architecture diagram of the designed system with two phases, including CloudSim Plus for workload simulation, a Python Gym environment for RL training using PPO and DQN models and a prediction server for real time integration with a simulation environment for PPO and DQN evaluation.

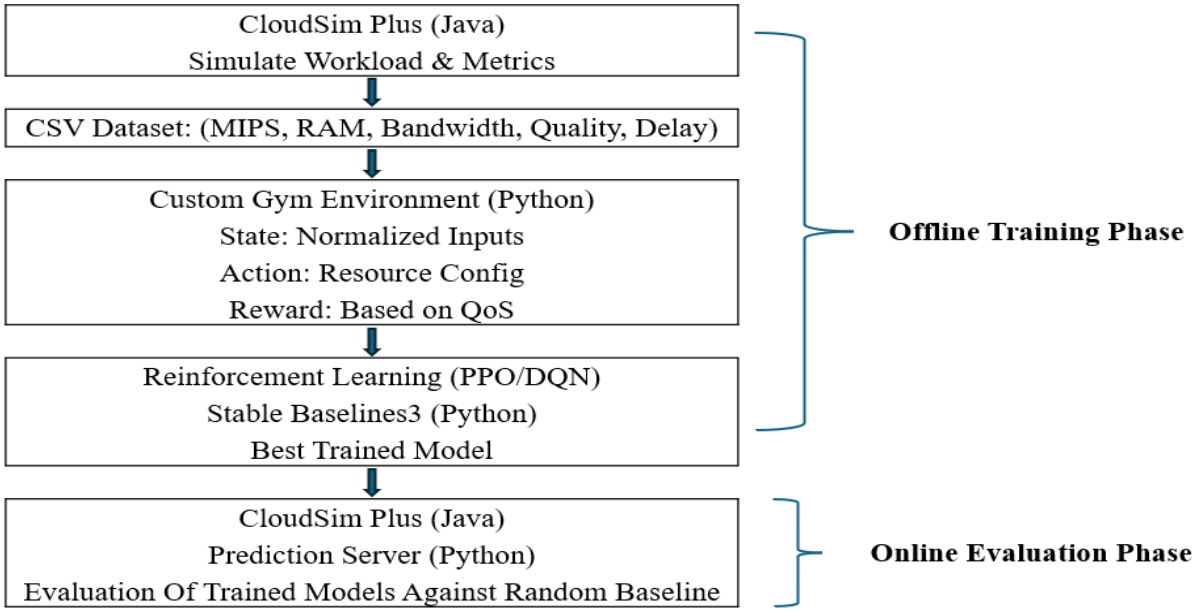

Figure 1: System Architecture for RL-Based Load Balancing in Live Cloud Streaming

4.2 Component Breakdown

The main parts of the system consist of the following:

1. **CloudSim Plus Simulation (Java)**

The CloudSim Plus based simulation environment is the main part of the system. The java file is used to create simulation scripts using Java for creating different types of streaming workloads with different combinations of VM configurations by simulating the real time variability in cloud environments. In each simulation iteration, the cloudlet is processed using the three VMs with low, medium, and high MIPS values. The calculated QoS metrics such as latency, jitter, energy consumption, and throughput will be recorded for each iteration. The final dataset will be generated and saved as a CSV file. This generated dataset will be used for training the machine learning model.

2. Reinforcement Learning Environment (Python Gym)

The next part of the system design is about a custom Gym compatible python environment to train the RL models by integrating with the generated dataset. It will consist of three important spaces (OpenAI, 2025). The state space consists of input to the environment such as called observation with a set of 5 values and the action space, which is simple which is that the RL model can choose between the three possible actions based on the VM configurations for each iteration and the reward function used to tell the model how to perform well based on the formula (OpenAI, 2025). Then the two RL models, PPO and DQN, will be trained using Stable-Baselines3 with different combinations of parameters. Then the best trained model based on the minimum average latency and energy in validation will be saved as a zip file.

3. Prediction Server (Python) and Testing Simulation

A Python prediction server will be developed to make a connection between the Java simulation environment and the trained RL models. For testing the model's performance, another Java file will be used to send the input variables such as observation data to the prediction server as a JSON file and receive the action from the RL models prediction which will be applied to the simulation loop. This architecture is used to simulate how the real world integration will happen between the RL model and real time deployment scenario.

4. Evaluation and Metrics Logging

To check the performance of the model, the same Java script will be used to calculate the average latency, jitter, energy consumption, and throughput for three tested models, such as PPO, DQN, and RANDOM across 300+ iterations on different types of workloads. Each result will be stored in the csv file for comparing and analyzing the performance between the three models.

4.3 Data Flow Design

In this system design, the data will flow in two stages:

1. Offline Training Phase

This phase is called offline training phase as the RL models were trained using pre-generated data without interacting with a simulated environment. In this phase the CloudSim Plus Java simulation will run 10,000 iterations with different streaming workloads and collected QoS metrics will be saved in a CSV file. Then the generated dataset will be used by the Python RL environment file for training the PPO and DQN models to learn the best VM combination based on the workload. Then the trained model will be saved as a zip file for future use.

2. Online Evaluation Phase

This phase is called the online evaluation phase as the trained RL models will be used in real time for making decisions in a live cloud streaming simulation environment. In this phase, the CloudSim Plus Java simulation will create new cloudlet workloads. These workloads will be sent to the Python prediction server file by converting the workload details into a normalized input vector as a JSON file. The server loads the trained model and predicts the best VM type and sends back the decision to the Java file, then it will allocate the VM configurations to run

the task, and log the performance metrics like latency, energy, jitter, and throughput into a CSV file for future performance analysis.

4.4 Design Strategy and Justification

The system design was created to achieve some key goals. To simulate the real-world workloads environment, CloudSim Plus will be used to simulate the three workload types by offering flexible and detailed control over virtual machines (CloudSim Plus, 2025). To make smarter decisions in load balancing, the system will use the PPO and DQN models by training with Stable-Baselines3 within a Python Gym environment. These two RL models were widely used methods for working effectively in optimizing resource usage applications (Sutton and Barto, 2018). The system uses JSON based interaction between Java and Python to keep the lightweight communication, which will help to run the processes asynchronously (ECMA International, 2017). For analysing the performance of the models, all the collected outputs were logged into a CSV files, which will be helpful to analyse the results later using tools like Python. The system also ensures the use of widely supported compatible tools like OpenAI Gym, Stable-Baselines3, and CloudSim Plus for easier integration with different platforms like Kubernetes in the future. Finally, this modular system encourages both real time and training usage at the same time, allowing each part of the system to remain separate and to be scalable or upgradable for future use.

5 Implementation

5.1 Simulation Setup in CloudSim Plus (Java)

The first simulation environment was implemented using the Java simulation tool for simulating and evaluating the cloud-based environment called CloudSim Plus 6.3.0. The WorkloadSimulation.java file is used to create the simulation script using Java for creating a single datacenter with one host and three different combinations of VM configurations such as MIPS, RAM and Bandwidth parameters with different types of streaming workloads by simulating the real time variability in cloud environments. The simulation code is designed to run for 10,000 iterations with different variations in cloudlet length and VM configurations for simulating three real world live cloud streaming workloads. In each simulation iteration, a cloudlet is processed using the three VMs with low, medium, and high MIPS values and three random cloudlet lengths will be assigned to the VMs respectively. The calculated QoS metrics for each VM and cloudlet combination, such as latency, jitter, energy consumption, and throughput were recorded for each iteration and saved in TrainingDataset.csv file. The final dataset had a total of 30,000 different types of workload and resource combinations. This generated dataset will be used for training the machine learning model.

5.2 Dataset Generation

The final dataset generated from 10,000 simulation runs has a total of 30,000 different types of workload and resource combinations. The saved dataset had several features, like the following, vmMips which is the CPU processing power (in MIPS) for the chosen VM, cloudletLength represents the total instruction length for the cloudlet (task), normRam represents the normalized amount of RAM needed, similarly normBw represents the normalized bandwidth requirement, videoQuality means the different types of the streaming

resolution like 480p, 720p, or 1080p, latency which is an important QoS metric means the total time it takes to complete the task (in milliseconds), energyConsumption reflects the amount of power used during the every execution of the tasks, jitter represents the variation in delay during the execution, throughput represents the number of tasks completed per second (ITU-T, 2008). This data was saved in a TrainingDataset.csv file which will be used to train the RL models in Python.

5.3 RL Environment Setup in Python

A custom Gym compatible Python environment was implemented using the streaming_env_multi_vm.py file to train the RL models by integrating with the generated dataset. A Python class called StreamingEnv was created for setting up the streaming workloads environment and interacting with the RL models. It will consist of three important spaces. The state space consists of input to the environment such as called observation, with a set of five values, including vmMips, which is normalized by 4000 MIPS, cloudletLength, which is normalized by 20,000, normRam, normBw and encoded values of videoQuality. The action space is created by the three possible actions where each action corresponds to selecting one of the VM configurations: 0 (Low), 1 (Medium), or 2 (High) based on the VM configurations for each iteration. The reward function is designed to tell the model how to perform well based on the formula such as

- **Reward = − (2 × latency + 0.2 × energy + 0.3 × jitter)**

It is designed to penalize high latency, energy use, and jitter and all the values are normalized. To maximize the performance of the model, a bonus of +1 will be rewarded to the model if all the three performance metrics fall below 30% of their maximum values. After finishing the action in each iteration, the episode will automatically move to the next iteration group. This Python environment will be used for the RL models to learn how to make smart scaling decisions about balancing resources in live cloud streaming workloads, by considering both the VM configurations and performance results.

5.4 Training PPO and DQN Models

The two RL models, PPO and DQN, were trained using the Python library called Stable-Baselines3. The generated dataset from the CloudSim Plus simulation is used to train the RL models. The training was done using the StreamingEnv environment (from streaming_env_multi_vm.py), which was wrapped with DummyVecEnv for compatibility with Stable-Baselines3 using Jupyter Python notebook cells. The training for both the models was performed in two stages. In the first stage, the PPO model was trained for 20 loop iterations with different combinations of hyperparameters such as learning_rates, gammas and batch_sizes at 10k timesteps using the PPOLoop.ipynb file. The results of each hyperparameter set with the calculated performance in terms of latency, energy, and jitter were shown in Figure 2.

From Figure 2, the best performing hyperparameter combinations were chosen as learning_rate=0.0005, gamma=0.99, n_steps=1024, batch_size=32, ent_coef=0.01, verbose=2 for the second stage of training. In this phase, the selected hyperparameter combinations were trained for longer like 150k timesteps for PPO, to ensure better learning across different workload scenarios using the PPOFinal.ipynb file. Similarly, in the first stage, the DQN model was trained for 20 loop iterations with different combinations of hyperparameters such as learning_rates, gammas and buffer_sizes at 10k timesteps using the DQNLoop.ipynb file. The results of each hyperparameter set with the calculated performance in terms of latency, energy, and jitter were shown in Figure 3.

Model	learning_rate	gamma	batch_size	latency	energy	jitter
PPO-01	0.0001	0.950	16	1.423423	0.000083	0.083908
PPO-02	0.0001	0.950	32	1.444927	0.000084	0.089107
PPO-03	0.0001	0.950	64	1.391125	0.000081	0.088171
PPO-04	0.0001	0.950	128	1.430112	0.000083	0.088021
PPO-05	0.0001	0.970	16	1.459681	0.000085	0.087157
PPO-06	0.0001	0.970	32	1.557639	0.000091	0.092874
PPO-07	0.0001	0.970	64	1.474356	0.000086	0.088131
PPO-08	0.0001	0.970	128	1.490129	0.000087	0.089905
PPO-09	0.0001	0.990	16	1.450588	0.000085	0.100520
PPO-10	0.0001	0.990	32	1.458353	0.000085	0.098566
PPO-11	0.0001	0.990	64	1.446386	0.000084	0.092473
PPO-12	0.0001	0.990	128	1.479403	0.000086	0.085038
PPO-13	0.0001	0.995	16	1.465688	0.000085	0.089820
PPO-14	0.0001	0.995	32	1.431862	0.000084	0.084807
PPO-15	0.0001	0.995	64	1.577307	0.000092	0.090694
PPO-16	0.0001	0.995	128	1.444346	0.000084	0.094027
PPO-17	0.0003	0.950	16	1.459817	0.000085	0.090338
PPO-18	0.0003	0.950	32	1.436777	0.000084	0.092622
PPO-19	0.0003	0.950	64	1.276519	0.000074	0.092216
PPO-20	0.0003	0.950	128	1.539236	0.000090	0.096506

Figure 2: PPO Hyperparameter Tuning Results

From Figure 3, the best performing hyperparameter combinations were chosen as learning_rate=0.0007, gamma=0.99, buffer_size=50000, learning_starts=500, batch_size=32 for the second stage of training. In this phase, the selected hyperparameter combinations were trained for longer like 200k timesteps for DQN, to ensure better learning across different workload scenarios using the DQNFinal.ipynb file. After the training, both models were saved using the model.save() method as a zip file for future use like real time decision making during the evaluation phase.

Model	learning_rate	gamma	buffer_size	latency	energy	jitter
DQN-01	0.0001	0.950	10000	1.483495	0.000087	0.087375
DQN-02	0.0001	0.950	20000	1.470811	0.000086	0.087554
DQN-03	0.0001	0.950	30000	1.392548	0.000081	0.093838
DQN-04	0.0001	0.970	10000	1.469754	0.000086	0.091311
DQN-05	0.0001	0.970	20000	1.415978	0.000083	0.095414
DQN-06	0.0001	0.970	30000	1.360776	0.000079	0.083558
DQN-07	0.0001	0.990	10000	1.443372	0.000084	0.094080
DQN-08	0.0001	0.990	20000	1.455968	0.000085	0.090974
DQN-09	0.0001	0.990	30000	1.345085	0.000078	0.095040
DQN-10	0.0001	0.995	10000	1.538754	0.000090	0.094326
DQN-11	0.0001	0.995	20000	1.480860	0.000086	0.080439
DQN-12	0.0001	0.995	30000	1.381013	0.000081	0.090997
DQN-13	0.0003	0.950	10000	1.451176	0.000085	0.089485
DQN-14	0.0003	0.950	20000	1.441500	0.000084	0.089633
DQN-15	0.0003	0.950	30000	1.331087	0.000078	0.094001
DQN-16	0.0003	0.970	10000	1.362149	0.000079	0.093265
DQN-17	0.0003	0.970	20000	1.552297	0.000091	0.088237
DQN-18	0.0003	0.970	30000	1.500018	0.000088	0.094402
DQN-19	0.0003	0.990	10000	1.390335	0.000081	0.085494
DQN-20	0.0003	0.990	20000	1.476710	0.000086	0.087660

Figure 3: DQN Hyperparameter Tuning Results

5.5 Real Time Validation Using Predictor Integration Between Java and Python

The saved RL models will be used to make decisions in the CloudSim Plus simulated environment. The prediction server called predict_server.py was created using Python for integrating models with simulated environments to make real time decision making. The simulation setup was created using java with the file called TestingTrainedModel.java. This java file sends the input variables such as observation data to the prediction server as a JSON file and the prediction server will load the RL model and send back the predicted action to the Java. Based on the action received, the Java simulation script will allocate the corresponding VM and run the cloudlet.

5.6 Evaluation Output and Logging

The final evaluation is implemented by the same setup of integrating trained PPO and DQN reinforcement learning models with a CloudSim Plus based simulation file, TestingTrainedModel.java. This Java code was developed to simulate how the real world integration would happen between the RL model and real time deployment scenario. The simulation script is used to run 100 iterations. For each iteration, a single cloudlet is created with different types of workload scenarios such as light, medium and heavy. These VMs values differ randomly in MIPS, RAM, and bandwidth. These values are sent as a JSON formatted input values to a Python script (predict_server.py) via standard input. Then the prediction server loads the RL model and sends back the predicted action that corresponds to one of the three VM configurations to Java. Then the workload will be executed based on the predicted action. This evaluation was performed for three models, such as the PPO model, DQN model and a random baseline model. Each model was executed with a different workload scenario, such as light, medium and heavy. Random mode chose the VM combinations randomly without loading any RL model from the prediction server. After executing the predicted action, the metrics like latency, energy use, jitter, and throughput for the chosen VM combinations will be calculated and logged for over 100 simulation runs. All the simulation results are logged into a CSV file named according to the different models, such as PPO, DQN and Random. These CSV files were analyzed to compare the performance of each model by graphs and plots created by Python using the Final Results.ipynb file.

6 Evaluation

6.1 Evaluation Setup

The final evaluation is implemented by the setup of integrating trained PPO and DQN reinforcement learning models with a CloudSim Plus based simulation file, TestingTrainedModel3.java. The evaluation process consists of three types of streaming workloads, such as light, medium and heavy. The performance of the model compared with key QoS metrics such as Latency, Energy Consumption, Jitter, and Throughput by also showing percentage improvements across all the models, such as PPO, DQN and Random.

6.2 Average Performance Metrics

Table 2 presents the average values of latency, energy, jitter, and throughput achieved by all the load balancing models, such as PPO, DQN, and Random across all scenarios, such as light, heavy and medium load conditions.

Table 2: Average Metrics Table

Row	Scenario	Latency	Energy	Jitter	Throughput	Model
0	Heavy	7.0000	0.000408	0.097380	342.857140	PPO
1	Light	1.5805	0.000091	0.096120	1558.585209	PPO
2	Medium	4.0100	0.000234	0.101412	598.503740	PPO
3	Heavy	7.0000	0.000408	0.099780	342.857140	DQN
4	Light	1.6932	0.000096	0.101647	1500.604502	DQN
5	Medium	4.0100	0.000234	0.103914	598.503740	DQN
6	Heavy	21.9864	0.001033	0.099597	200.950365	Random
7	Light	5.3131	0.000247	0.099104	840.348755	Random
8	Medium	15.5300	0.000714	0.095362	318.123989	Random

The results show that both the PPO and DQN models performed better than the Random baseline model in terms of latency, energy and throughput. For example, in the heavy workload comparison, latency dropped from 21.98s in Random model to 7.0s in both PPO and DQN, which is a 68% improvement. Similarly, energy usage decreased by about 60%, and throughput increased by 70% or more with the RL based models compared to the Random model. Jitter didn't show any significant improvements and stayed the same across all models.

6.3 Percentage Improvements Over Baseline

Table 3 explains the percentage improvements of PPO and DQN compared to Random load balancing model.

Table 3: Percentage Improvement Table

Scenario	Metric	Model	Improvement (%)
Heavy	Latency	PPO	68.162137
Heavy	Energy	PPO	60.463933
Heavy	Jitter	PPO	2.226562
Heavy	Throughput	PPO	70.617824
Light	Latency	PPO	70.252771
Light	Energy	PPO	63.162886
Light	Jitter	PPO	3.010874
Light	Throughput	PPO	85.468854
Medium	Latency	PPO	74.179008
Medium	Energy	PPO	67.230780
Medium	Jitter	PPO	-6.343709
Medium	Throughput	PPO	88.135369
Heavy	Latency	DQN	68.162137
Heavy	Energy	DQN	60.463933
Heavy	Jitter	DQN	-0.183037
Heavy	Throughput	DQN	70.617824
Light	Latency	DQN	68.131599
Light	Energy	DQN	61.233901
Light	Jitter	DQN	-2.565787
Light	Throughput	DQN	78.569254
Medium	Latency	DQN	74.179008
Medium	Energy	DQN	67.230780
Medium	Jitter	DQN	-8.967809
Medium	Throughput	DQN	88.135369

The results show that RL models learnt the scaling policies well, which helps to reduce resource usage without affecting the user experience. Significant improvements can be seen in throughput by up to 88% in the medium workload scenario for both PPO and DQN, which means they allocated the compute resources efficiently based on the streaming demands. Overall, energy improved by up to 67% from both RL models. These improvements will help to reduce the overall resource costs. Latency remains a consistent improvement of above 68–74% across all the workloads for both the RL models compared to the Random model.

6.4 Visual Analysis of Metrics

The comparison of average latency across all the three workloads is shown in Figure 4. The PPO and DQN models consistently showed better performance compared to the Random model in all types of workload conditions. For example, under heavy workloads, RL-based models achieved a 66–70% reduction in latency compared to Random model.

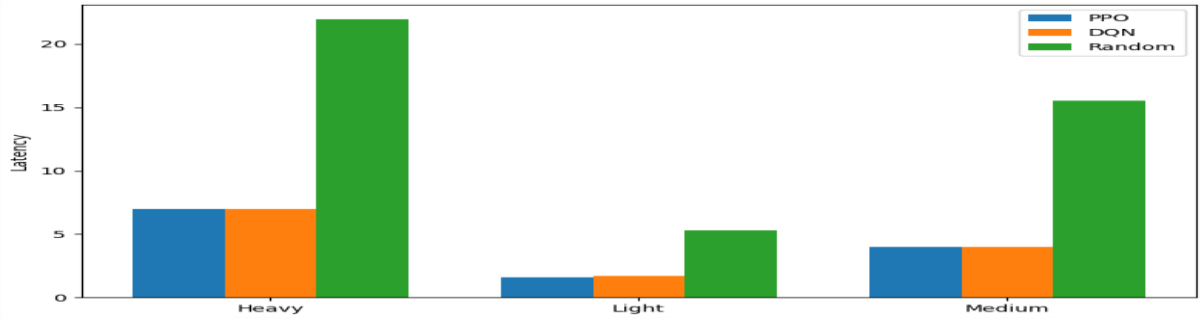

Figure 4: Latency Comparison.

The comparison of average energy across all the three workloads showed in Figure 5. Both PPO and DQN models showed better energy efficiency than the Random model. For example, up to 67% of energy is saved using PPO during heavy workloads.

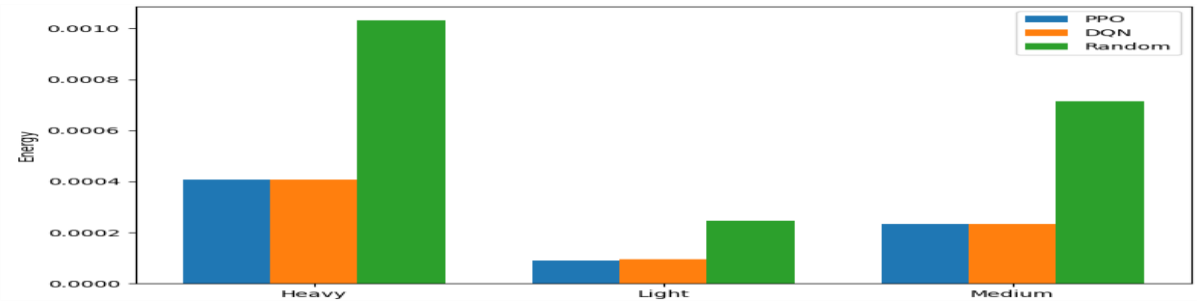

Figure 5: Energy Comparison.

The comparison of average Jitter across all the three workloads showed in Figure 6. As mentioned earlier, Jitter didn't provide any significant improvements across all three models as shown in Figure 6.

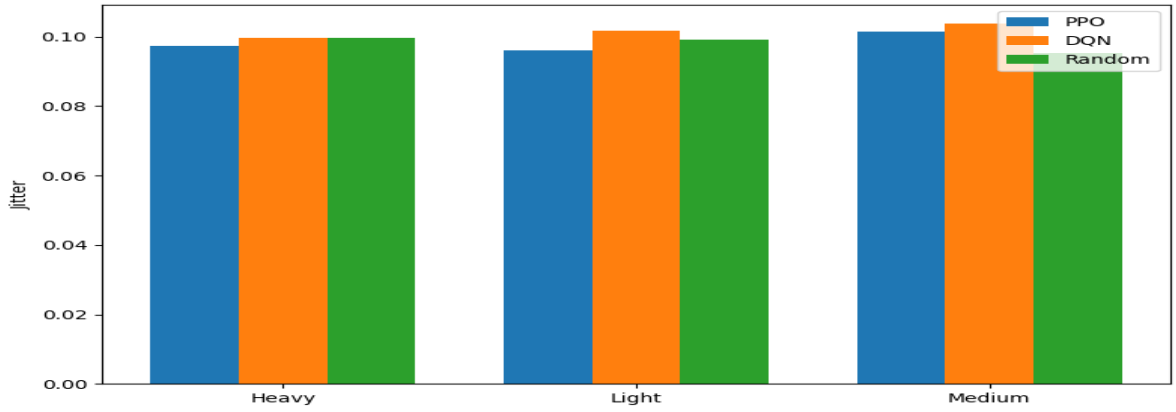

Figure 6: Jitter Comparison.

The comparison of average throughput across all the three workloads showed in Figure 7.PPO achieved the highest task completion rates compared to the DQN and Random models

as shown in Figure 7. Under light workloads, PPO and DQN reached over 1500 units, which is nearly double that of the Random model.

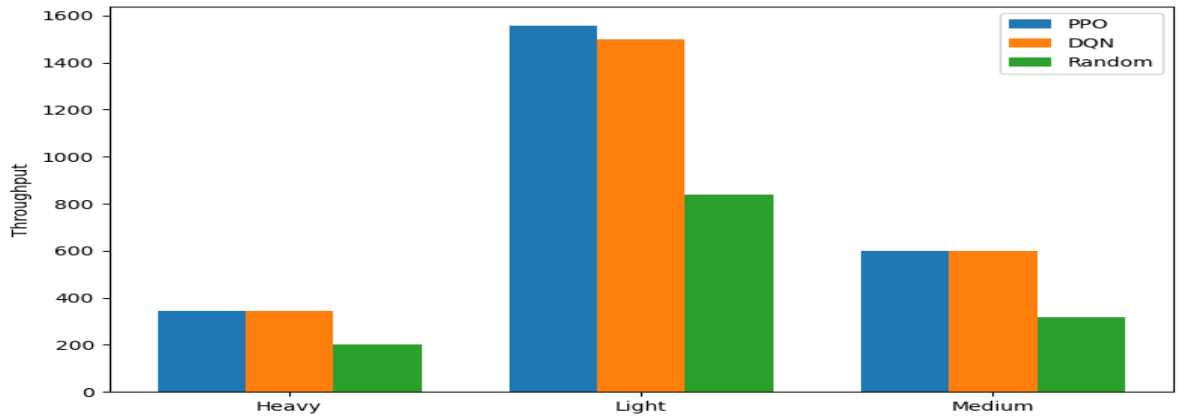

Figure 7: Throughput Comparison.

6.5 Discussion

The above evaluation showed the nine experimental results based on the different combinations of workloads, such as light, medium and heavy and load balancing methods such as PPO, DQN and Random models. The main goal of this evaluation section is to see how well the RL model's load balancing decision reduces energy consumption without affecting the QoS in live cloud streaming by measuring latency, energy use, throughput, and jitter. From seeing the results, both PPO and DQN models performed better than the Random model in most of the cases. For example, PPO model reduced the latency by up to 74% and saved 67% more energy and improved 88% of throughput in medium workloads. DQN also provided a similar output but slightly less improvements compared to the PPO. These results support the earlier research of Lahande (2023) and Chawla (2024) which uses RL for managing cloud resources effectively. However, some limitations were there in this experimental set up like each virtual machine was never tested with simulating real-world challenges like network delays and queuing delays. Because of this, the calculated metrics results like latency and jitter may not be accurate on heavy workloads. Then the results of jitter did not show any better consistent values. In some cases, Random showed better jitter performance compared to the PPO and DQN models like in medium workloads. This may be caused due to the basic simulation set up without simulating packet level variation, which is important for streaming workloads. The training of the RL models included with different types of workload data which will help balance learning. The results in RL models showed better performance in light and medium workloads compared to the heavy workloads. Even though the PPO and DQN performed well than Random in latency and throughput, the margin was smaller. If the training focused more on challenging cases like heavy workload conditions, it may learn better to handle heavy workload conditions. Even with those limitations, The RL based load balancing is still flexible, energy efficient and effective. Without using the fixed rule-based load balancing like Kaur and Singh (2022), RL based load balancing methods adjusts to change the different types of workload conditions automatically. These results also support research by Wilk et al. (2018) and Agos Jawaddi and Ismail (2024) which provided simulation based RL models. In summary, the PPO and DQN models showed some great potential and can be used for future research with more realistic simulation by integrating better models of the network, real-time learning and exploring more varied workload patterns.

7 Conclusion and Future Work

This research focussed on whether reinforcement learning based models can provide energy efficient load balancing without affecting the QoS services in live cloud streaming environments. The research is based on whether it is possible to perform better than traditional or random auto scaling methods by training the RL models to make smarter decisions in dynamic cloud environments. To find this out, the main goal is to design a cloud simulation environment and RL training by generating a dataset from the simulated environment and training the models using that dataset and validating the models and comparing their performance with key QoS metrics like latency, jitter, throughput and energy against a random model. This design goal is implemented using CloudSim Plus for workload simulation, Python based Gym environments for RL model training and a real time validation of RL models using Python for integrating models with simulated environments to make real time decision making. The two RL models, PPO and DQN, were trained and evaluated using different types of workloads simulating light, medium and heavy workload scenarios. The evaluation results of these models are compared with a random scaling model within the simulation environment. The results showed that both RL models showed better performance compared to the random model across all the key metrics. For example, PPO reduced the latency by up to 74%, reduced the energy by up to 67% and improved the throughput by up to 88%. The DQN model also showed some good performance but less compared to the PPO. From these findings, the PPO and DQN models showed some great potential and can be used for load balancing with efficient use of energy and without affecting the QoS services in live cloud streaming environments. Despite these positive results, there were some limitations to look up. The system was tested only using the simulation environment, not directly with the real-world cloud streaming set up, which limits how well the system will work in real world use. The training of the RL model is limited to a fixed number of VMs and trained using a fixed pattern of workloads. These set up results in RL models showed better performance in light and medium workloads compared to the heavy workloads. This modular system with simulation, training and prediction layers allows each part of the system to remain separate and to be scalable or upgradable for future use. To move forward, there are several meaningful ways to continue this research in the future, like validating the trained RL models in live Kubernetes environments with Prometheus as an agent to collect real-time metrics and Horizontal Pod Autoscaling (HPA) integration. Another idea for extending this research in future is to develop spot pricing in reward functions to make the system more cost aware. Some other useful upgrades of this research would be to expand the RL model's decision with more options, like choosing the type of resources, such as CPU or GPU, and using multiple machine learning models to balance the loads in the same environment. In conclusion, this research showed and helped to understand that the process of reinforcement learning based load balancing for efficient use of energy without affecting the performance metrics of QoS in live cloud streaming environments is possible and beneficial. It achieves the main research goals and creates a strong base for future work by providing academic knowledge and practical value for cloud-based environments.

References

Alibaba Cloud (2023) ‘Load balancing for video streaming services’, *Alibaba Cloud Tech News*. Available at: https://www.alibabacloud.com/tech-news/a/load_balancer/gu0idw26l4-load-balancing-for-video-streaming-services (Accessed: 9 August 2025).

Agos Jawaddi, S.N. and Ismail, A. (2024) ‘Integrating OpenAI Gym and CloudSim Plus: A simulation environment for DRL agent training in energy-driven cloud scaling’, *Simulation Modelling Practice and Theory*, 130, 102858. doi:10.1016/j.simpat.2023.102858.

Beloglazov, A. and Buyya, R. (2012) ‘Energy efficient resource management in virtualized cloud data centers’, *Future Generation Computer Systems*, 28(5), pp. 755–768. doi:10.1016/j.future.2011.04.017.

Bishoy, S. (2024) ‘Self-optimized agent for load balancing and energy efficiency: A reinforcement learning framework with hybrid action space’, *IEEE Open Journal of the Communications Society*, 5, pp. 4902–4919. doi:10.1109/OJCOMS.2024.3429284.

Calheiros, R.N., Ranjan, R., Beloglazov, A., De Rose, C.A.F. and Buyya, R. (2011) ‘CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms’, *Software: Practice and Experience*, 41(1), pp. 23–50. doi:10.1002/spe.995.

Chawla, K. (2024) ‘Reinforcement learning-based adaptive load balancing for dynamic cloud environments’, *arXiv preprint*. Available at: https://arxiv.org/abs/2409.04896 (Accessed: 9 August 2025).

Cisco (2022) ‘Cisco visual networking index: Forecast and trends, 2022–2027’. Available at: https://www.cisco.com/site/us/en/solutions/annual-internet-report/index.html (Accessed: 9 August 2025).

CloudSim Plus (2025) *CloudSim Plus Documentation*. Available at: https://cloudsimplus.org (Accessed: 9 August 2025).

Esther, J.V., Alugoju, S. and Indraja, K. (2024) ‘Adaptive cloud load balancing with reinforcement learning: Leveraging Google cluster data’, in *Proceedings of the 2024 5th International Conference on Electronics and Sustainable Communication Systems (ICESC)*. IEEE. doi:10.1109/ICESC60852.2024.10689973.

ECMA International (2017) *ECMA-404: The JSON data interchange syntax*. 2nd edn. Geneva: ECMA International. Available at: https://www.ecma-international.org/publications-and-standards/standards/ecma-404/ (Accessed: 9 August 2025).

ITU-T (2008) *E.800: Definitions of terms related to quality of service (QoS)*. Geneva: International Telecommunication Union. Available at: https://www.itu.int/rec/t-rec-e.800-200809-i (Accessed: 9 August 2025).

Javed, F., Khan, A. and Nawaz, R. (2024) ‘Using reinforcement learning to improve energy efficiency in AI-based data centers’. Available at: https://www.researchgate.net/publication/386992658 (Accessed: 9 August 2025).

Kaur, J. and Singh, A. (2022) 'Review of load balancing techniques in cloud computing', *International Journal of Computer Applications*. Available at: https://www.researchgate.net/publication/367935844 (Accessed: 11 April 2025).

Kaur, A. and Kaur, B. (2019) 'Load balancing optimization based on hybrid heuristic–metaheuristic techniques in cloud environment', *Journal of King Saud University – Computer and Information Sciences*, 34(3), pp. 813–824. doi:10.1016/j.jksuci.2019.02.010.

Kahil, A., Sharma, T., Välisuo, P. and Elmusrati, M. (2025) 'Reinforcement learning for data center energy efficiency optimization: A systematic literature review and research roadmap', *Applied Energy*, 396, 123456. doi:10.1016/j.apenergy.2025.123456.

Kliazovich, D., Bouvry, P. and Khan, S.U. (2013) 'GreenCloud: A packet-level simulator of energy-aware cloud computing data centers', *The Journal of Supercomputing*, 62(3), pp. 1263–1283. doi:10.1007/s11227-010-0504-1.

Kumar, R., Verma, S. and Sharma, N. (2023) 'Hybrid deep Q-learning model for resource management in cloud', *The Bioscan*. Available at: https://thebioscan.com/index.php/pub/article/download/3076/2629/5508 (Accessed: 9 August 2025).

Lahande, P.V. (2023) 'Reinforcement learning approach for optimizing cloud resource utilization with load balancing', *IEEE Access*, 11, pp. 127567–127577. doi:10.1109/ACCESS.2023.3329557.

Li, X., Darwich, M., Bayoumi, M. & Amini Salehi, M. (2020) *Cloud-Based Video Streaming Services: A Survey*, arXiv preprint. Available at: https://arxiv.org/abs/2011.14976 (Accessed: 9 August 2025).

Mnih, V., Kavukcuoglu, K., Silver, D., et al. (2015) 'Human-level control through deep reinforcement learning', *Nature*, 518(7540), pp. 529–533. doi:10.1038/nature14236. Available at: https://www.nature.com/articles/nature14236 (Accessed: 9 August 2025).

Nawaz, M., Ali, H. and Tanveer, M. (2025) 'Energy-efficient cloud computing through reinforcement learning-based workload scheduling', *International Journal of Advanced Computer Science and Applications*, 16(4). Available at: https://thesai.org/Downloads/Volume16No4/Paper_64-Energy_Efficient_Cloud_Computing.pdf (Accessed: 9 August 2025).

OpenAI (2025) *OpenAI Gym Documentation*. Available at: https://github.com/openai/gym (Accessed: 9 August 2025).

Schulman, J., Wolski, F., Dhariwal, P., Radford, A. and Klimov, O. (2017) 'Proximal policy optimization algorithms', *arXiv preprint* arXiv:1707.06347. doi:10.48550/arXiv.1707.06347. Available at: https://arxiv.org/abs/1707.06347 (Accessed: 9 August 2025).

Sutton, R.S. and Barto, A.G. (2018) *Reinforcement Learning: An Introduction*. 2nd edn. Cambridge, MA: MIT Press. Available at: https://mitpress.mit.edu/9780262039246/reinforcement-learning/ (Accessed: 9 August 2025).

Wilk, P., Furmankiewicz, M. and Kołodziej, J. (2018) 'Repeatable experiments in cloud resources management using Deep-Q-Learning and CloudSim Plus', *Cracow Grid Workshop (CGW 2018)*, presentation. Available at: https://www.cyfronet.pl/cgw18/presentations/S5-02_CGW2018_WF_PK_JK.pdf (Accessed: 9 August 2025).

Yang, X. (2022) 'Load balancing streaming servers', *SRS Blog*. Available at: https://ossrs.net/lts/en-us/blog/load-balancing-streaming-servers (Accessed: 9 August 2025).

Zaman, S. and Grosu, D. (2013) 'Combinatorial auction-based allocation of virtual machine instances in clouds', *Journal of Parallel and Distributed Computing*, 73(4), pp. 495–508. doi:10.1016/j.jpdc.2012.12.006.

Zhou, G., Tian, W., Buyya, R., Xue, R. and Song, L. (2024) 'Deep reinforcement learning-based methods for resource scheduling in cloud computing: A review and future directions', *Artificial Intelligence Review*, 57, pp. 1–49. doi:10.1007/s10462-024-10756-9.