National College of Ireland

Real-Time, Cost-Aware Resource Scheduling in Multi-Cloud Systems Using PPO-Based Reinforcement Learning

MSc Research Project
MSc in Cloud Computing

Priya Shanmugam
Student ID: 23273518

School of Computing
National College of Ireland

Supervisor: Shreyas Setlur Arun

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Priya Shanmugam		
Student ID:	23273518		
Programme:	MSc in Cloud Computing	**Year:**	2024-25
Module:	MSc Research Project		
Supervisor:	Shreyas Setlur Arun		
Submission Due Date:	11.08.2025		
Project Title:	Real-Time, Cost-Aware Resource Scheduling in Multi-Cloud Systems Using PPO-Based Reinforcement Learning		
Word Count:	7736	**Page Count:**	25

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: Priya Shanmugam ……

Date: 07.08.2025 ……

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):	

Real-Time, Cost-Aware Resource Scheduling in Multi-Cloud Systems Using PPO-Based Reinforcement Learning

Dashboard: https://ppo-cloud-scheduler.streamlit.app/
GitHub: https://github.com/Priya-1110/ppo-cloud-scheduler

Priya Shanmugam
23273518

Abstract

Scheduling and task placement strategies on heterogeneous platforms are most important in the age of cloud computing in order to optimize the use of cloud computing resources with the aim of achieving the Service-Level Agreements (SLAs) with the lowest possible operation costs. In this study, a real-time, intelligent scheduling system informed by Proximal Policy Optimization (PPO), a reinforcement learning algorithm is proposed in order to dynamically distribute computations in using multiple cloud providers including Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform (GCP). The system emulates the workload in the real world on fog-cloud setting using iFogSim and the crucial workload figures to create the vectors of task states, such as the CPU requirements, memory usage, execution time, and SLA level. The PPO agent is trained with lifetime reward (contingent on SLA and cost-optimality) signals on how to select the optimal choices of cloud. It is compared with traditional (Round Robin (RR), First-Come-First-Serve (FCFS)) and reinforcement learning based scheduling policies such as Deep Q-Network (DQN), and Advantage Actor Critic (A2C). The outcomes demonstrate that PPO has increased SLA satisfaction score and a lower average CPU cost in every case, which indicates its efficiency in real-time decision-making frameworks. A single Streamlit dashboard was devised to process the results of the scheduling along with the performance of the system, whereas explainability aimed at SHAP was utilized to understand the choices of PPO. The study presents an explainable intelligent and modular solution that involves task scheduling in the cloud, providing a scalable philosophy to solving resource allocation in multi-clouds.

***Index Terms*—**Cloud Optimization, Cost Prediction, AI Scheduling, Fault Tolerance, Cloud Simulation

1 Introduction

The increased use of the cloud has changed the way organizations utilize, implement, and expand their services online. Since many cloud services providers exist offering different pricing options and options to infrastructure capability such as Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform (GCP), the need to propose intelligent mechanisms of resource scheduling which dynamically selects the most appropriate cloud environment to execute a task is on the rise. Further, the generation of immense critical workloads with unpredictable variability due to the proliferation of the Internet of Things (IoT) devices and edge computing, are adding pressure to meet real-time requirements. Conventional scheduling policies such as Round Robin (RR) or First-Come-First-Serve

(FCFS) are not able to factor in the metric of load running time, task cost, SLA deadlines, or resource constraint and thus making poor selections of the cloud.

To overcome this drawback, this study introduces a reinforcement learning-based approach to cloud scheduling based on Proximal Policy Optimization (PPO) that trains to make real-time decisions on schedule based on the dynamically available task and infrastructure state. The simulator they use to create the proposed system is iFogSim to create an environment similar to that of a real-world fog and cloud computing environment. It gathers task characteristics including CPU demand, execution time, SLA constraints, and memory consumption that are used to generate state vectors input to the PPO agent. PPO scheduler is then compared with other strategies namely DQN (Deep Q-Network), A2C (Advantage Actor Critic), FCFS and RR to determine their efficiency, SLA compliance, and cost-effectiveness.

This chapter will give the background and the motivation of the research, the specific problem statement of the research questions, objectives, and the structure of the remaining chapters. It establishes the background of generating and assessing an intelligent, modular and explainable scheduling framework of multi-cloud systems via reinforcement learning techniques.

1.1 Background and Motivation

The digital transformation has come to see cloud computing play a leading role as it provides an elastic and scalable infrastructure responding to subsequent needs of computations. With organisations pursuing multi-cloud strategies using clouds such as AWS, Azure and GCP, the problem of effectively scheduling tasks in disparate environments becomes a more complex scheduling challenge. This complexity is further compounded by the fact that edge and fog computing have led to the need to dynamically distribute latency-sensitive workloads across the network layers (Mahmud et al., 2022) [1].

Conventional resource scheduling methods like Round-Robin (RR), and First-Come-First-Serve (FCFS) cannot optimize cost, SLA compliance, and resource availability and are not very effective. Conversely, reinforcement learning (RL) has been of immense promise in the management of dynamic scheduling issues as a result of experience-based learning. Specifically, Proximal Policy Optimization (PPO) has been hailed due to its stability and efficiency in on-the-fly cloud-based conditions, the kind that allows adaptive optimization decision-making without the pre-marked up data [2] (Das and Rao, 2023).

The study is an extension of these achievements, where they developed a PPO-based schedule task planner that combines with the iFogSim2 simulation framework. The system distributes tasks dynamically amid various multi-cloud providers utilizing real-time task measures and is compared with another technique such as A2C, DQN, FCFS, and RR. PPO has experimental performance better on both SLA satisfaction rate and average CPU cost than these baselines, which indicates the practicality of the method to construct an intelligent, explainable, and modular scheduler in a cloud system (Zhou et al., 2024) [3].

1.2 Problem Statement

As the use of cloud intensifies, more enterprises are taking advantage of multi-cloud usage to promote redundancy, cost optimization, and satisfy varying application

requirements. This change, however, adds a lot of complexity when it comes to scheduling computational jobs across different providers such as Amazon Web Services (AWS), Microsoft Azure and Google Cloud Platform (GCP) who all have heterogeneous pricing schema, performance metrics and service-level agreement (SLA) schema. In these settings, classical static policies such as Round Robin (RR) and First-Come-First-Serve (FCFS) cannot have the wisdom to respond in a real-time setting when the workload characteristics and availability of cloud resources change.

Moreover, the predominant known solutions do not utilize cost-awareness, SLA deadlines, and dynamics in system states in making decisions, leading to either underutilization of the resources, violating service level agreements or accruing unreasonable costs on operation. Although the usage of supervised machine learning approaches has been considered to perform tasks allocation, they are usually based on off-line training data, and they are not applicable to learning and generalization in dynamic, real-time cloud settings.

Scheduling users also still require an adaptive and intelligent solution; one that can learn the best ways to cluster clouds using live feedback and performance results. Our study helps fill in this gap by coming up with a Proximal Policy Optimization (PPO)-based cloud task scheduler meant to introduce high SLA compliance as well as low CPU cost within a simulated multi-cloud environment. This is done so that we compare our system with the state-of-the-art reinforcement learning models (A2C, DQN) and other traditional methods (FCFS, RR), to show the practical usefulness of the deep reinforcement learning in cloud resource management.

1.3 Research Questions

- **Primary Research Question:** How effective is Proximal Policy Optimization (PPO) in improving task scheduling performance in multi-cloud environments, in terms of SLA satisfaction and CPU cost, compared to traditional and reinforcement learning-based approaches?

- How does PPO compare to baseline schedulers such as Round Robin (RR), First-Come- First-Serve (FCFS), Deep Q-Network (DQN), and Advantage Actor-Critic (A2C) in terms of task execution efficiency and SLA compliance?

1.4 Proposed Solution

This study suggests a dynamic and intelligent workflow scheduling scheme based on the proximal policy optimization (PPO) algorithm using deep reinforcement learning (DRL). The system uses as a foundation an iFogSim2 simulation toolkit focused on modeling edge-fog-cloud infrastructures, and offers the possibility to test resource management approaches under realistic circumstances.

The features that the scheduler accepts as input are a set of task specific state features in terms of CPU demand, memory requirement, execution time and the SLA duration. They are encoded by means of a state vector that passes into the PPO agent. The agent will choose the most suitable cloud provider (AWS, Azure, or GCP) to use to perform every task using the learned policies so that it could get the maximum cumulative rewards that depend on the SLA compliance and reduced CPU cost.

In benchmarking PPO with four baseline schedulers, it is important to assess the proposed framework.

- Traditional: Round Robin (RR) and First- Come- First-Serve (FCFS)
- Reinforcement Learning: Deep Q-Network (DQN), and Advantage Actor-Critic (A2C)

The resultant decisions on scheduling and the performance metrics of the system are displayed in real-time with an easily made Streamlit dashboard, with explainability achieved via SHAP (SHapley Additive exPlanations) to gain transparency on agent decision-making behavior. Through this modular and explainable design, the proposed solution will be not only viable in terms of practicality, but also flexible enough to integrate with the real and dynamic multi-cloud systems.

1.5 Research Objectives

This research aims mainly to create and test smart, reinforcement learning reinforced-based planning system of scheduled tasks into multi-cloud. The scheduler will seek to optimize the task assignment regarding the question of SLA fulfillment and CPU cost minimum and be adaptable to the real-world fluctuating workloads.

In order to foster such an aim, the following specific aims are stated:

- Formulate and implement a PPO-based scheduling framework in the iFogSim2 simulator of models of fog-cloud environments as well as multi-clouds providers.
- Isolate pertinent task attributes including CPU requirements, memory load, run time and SLA deadlines with an aim to developing state vectors of reinforcement learning input.
- Train and test the agent (PPO) to generate real-time decisions about choosing clouds using the changes in the task states and system states as information.
- To compare the performance of a PPO scheduler with a baseline, compare it with traditional (RR, FCFS) and reinforcement-learned (DQN, A2C) algorithms.
- Quantitative measures to use to determine the performance results include SLA satisfaction rate, average cost of CPU, and general efficiency of scheduling.
- Add interpretability and visual feedback in the form of a Streamlit based dashboard, and SHAP explainability methods to be transparent in decision-making.

All these goals are expected to prove the competence and usefulness of PPO as a smart, sound solution to scheduling tasks in the real-world multi-cloud systems.

1.6 Limitations and Mitigation Strategies

- **Simulation Constraints**: iFogSim2 does not reflect all the realities of cloud architectures, but this was mitigated through the modeling of different workloads, the design of SLA and price models using real data garnered with AWS and Azure.
- **Training Overhead**: PPO model training is computationally heavy and thus offline training with pre-generated simulation logs were used to avoid training overhead during run-time.

- **No Live Learning**: The live training of the PPO agent was not a viable option in the simulation path and instead pre-trained model was used in real time during task scheduling.
- **Decrease Scope in Cloud**: This study covered only AWS, Azure, and GCP, though the system itself was built in a modular state so that the rest of the cloud providers could be easily integrated in further research.
- **Dashboard Responsiveness**: The dashboard design made with Streamlit slows during an update with high frequencies, and that was addressed through metric aggregation and caching mechanism.

1.7 Report Structure

Table 1: Report Structure

Chapter	Title	Description
1	Introduction	Presents the research setting, the problem, research aims, and section of the methodology.
2	Literature Review	Critically revises current literature on cloud scheduling, reinforcement learning, PPO.
3	Research Methodology	Provides information about the general strategy, the experiment design, the tools (iFogSim2) and the data collection strategy.
4	Design Specification	Describes architecture, system components, state vectors and integration design of PPO.
5	Implementation	Describes the system build, PPO teaching, Java/Python socket integration and deployment.
6	Evaluation and Results	Provides quantitative evaluation of PPO in contrast to RR, FCFS, DQN and A2C and such metrics as SLA and cost.
7	Conclusion and Future Work	Presents conclusion, addresses limitations of the research, and describes possible future expansion.

2 Related Work

The increasing complexity of cloud environments and the move to multi-cloud and edge computing has resulted in a recent explosion of work in intelligent scheduling of tasks and optimization of resources. This is especially true about reinforcement learning (RL) as a promising paradigm to dynamically assign tasks, completing the cost versus performance trade-off, and enhance the SLA (Service-Level Agreement) compliance. The following section classifies pertinent literature into three categories: **(i)** reinforcement learning in multi-cloud scheduling, **(ii)** optimization of resources using AI that involves edge and fog computing, and **(iii)** cost-aware and scheduling frameworks of cloud computing.

2.1 Reinforcement Learning for Multi-Cloud Task Scheduling

New solutions in deep reinforcement learning (DRL) have proved promising in terms of scheduling tasks when there are heterogeneous multi-cloud environments. Mangalampalli et al. (2024) [4] developed a DRL-based scheduler to respond dynamically and adapt to workload variations, which enhance SLA compliance and reduce, underutilization of resources in multi-cloud environments. Equally, Choppara and Mangalampalli (2025) [5] pushed this idea further and added fog nodes in their DRLMOTS system with better latency and performance results with cloud-fog systems.

In 2025, Sgambati et al. (2025) [6] proposed a decentralized PPO-based scheduling mechanism designed to work on a high-performance computing (HPC) system and showed that their solution can scale and reach extremely low overheads in distributed decision-making. An algorithm-level review of DRL job scheduling methods was developed by Gu et al. (2025) [7], identifying training complexity and reward sparsity as two of the most significant issues in this domain of problem solving, which is circumvented via our compressed state representation.

Zhou et al. (2024) surveyed most approaches of DRL-based cloud scheduling and found PPO, DQN, and Actor-Critic to be the best in terms of real-time decision making, which is also the rationale behind our selection of PPO against DQN A2C. MADQN-based scheduling of tasks in multi-agent cloud systems was also further elaborated by Pei et al. (2024) [8] with the direction of the need to shift towards distributed intelligence in the context of DRL scheduling.

This was proven when Pan et al. (2025) [9] came up with a two layer DQN architecture to set up a dynamic situation where the amount of work theoretically varies. They have ensured that it is important to have a multi layered DRL in a dynamic set-up of scheduling. All these studies confirm the adequacy of our interest in PPO-based real-time scheduling as compared to modern DRL methods.

2.2 AI-Driven Cloud Resource Optimization and Edge Scheduling

Some research has gone further to apply DRL-based scheduling to edge and fog computing Level environments (rather than to the centralized cloud). The research conducted by Wang et al. (2023) [10] proposed DRLIS, a common framework to handle task scheduling between the fog and cloud across levels using deep reinforcement learning, and showed a better performance in latency and resource usage in the IoT setting.

Xu et al. (2024) [11] improved scheduling of Kubernetes using deep learning and DRL that created automation of pod placement in large-scale clusters, an important design influence over our modular dashboard. The strategies of DRL in multi-access edge computing (MEC) have been reviewed by Ismail et al. (2025) [12] highlighting the growing involvement of the agent-based learning in scarce resource settings.

Li et al. (2023) [13] suggested a scheduler called GNN-based multi-agent reinforcement learning scheduler in ML workloads on edge compute devices powered by graph neural networks to learn about dependencies on ML workloads. [14] Liu et al. (2022) proposed a DRL model of load-balancing-aware resource control in edge computing to handle performance bottlenecks by intelligent, adaptive policy.

The latter works emphasize the efficiency of DRL as a way to handle complex, resource-constrained environments the direct application of which is applied in our edge-cloud simulation based on iFogSim2.

2.3 Scheduling Frameworks and Cost-Sensitive Models

In addition to conventional DRLs, a number of scholars have addressed cost-optimised scheduling in cloud computing. Blanco et al. (2024) [15] presented RELMAS, an online scheduler based on DRLs of multi-tenant DNN workloads that maximize QoS contention of shared resources in cloud conditions.

Russo et al. (2024) [16] also adopted the approach of reinforcement learning to aid QoS-aware task scheduling within multi-accelerator systems done in real-time to balance workloads between GPUs and CPUs. Sun et al. (2022) [17] have proposed DDDQN-TS, a deadline- and load-based scheduling protocol, which performed significantly better, in terms of task run-time and economic feasibility.

Collectively, these papers demonstrate the range of ways in which DRL has been used to balance performance, cost, and SLA compliance which are precisely the trade-offs our PPO-based scheduler uses with compressed state representations that are SLA-aware and real-time never-look-ahead decisions.

Table 2: Literature Review

Author/Source Title/Year	Purpose / Rationale / Aims / Question posed	Method / Sample / Study Type	Results / Findings / Conclusions	Key Ideas / Themes	Strengths / Weaknesses / Gaps	Similarities / Differences to Other Studies	Notes / My Comments
Gu et al., 2025	DRL survey in job scheduling	Algorithm-level review	Highlights PPO, DQN; identifies reward sparsity as key issue	DRL taxonomy	Strong review; lacks practical demo	Basis for PPO, A2C selection	Used in design decisions
Sgambati et al., 2025	Decentralized PPO in HPC	PPO implementation	Low overhead, scalable performance	Distributed PPO	High scalability; lacks fog focus	Unique decentralized approach	Inspires edge-side PPO
Choppara & Mangalampalli, 2025	Cloud-fog DRL scheduler	DRLMOTS, fog sim	Improves latency, performance	Fog-aware DRL	Fog-aware; no explainability	Builds on Mangalampalli et al.	Useful for edge integration
Pan et al., 2025	Dual-layer DQN for clouds	Two-stage DRL	Adapts well to dynamic loads	Multi-layer DQN	Complex; good for variable loads	Similar to A2C in idea	Dual-logic insight
Ismail et al., 2025	Survey of DRL in edge	MEC-focused review	DRL strategies growing in MEC	Edge DRL landscape	Broad scope; no solution	Supports edge-focused work	Literature base for fog sim
Xu et al., 2024	DRL + Kubernetes autoscheduling	Applied study	Automates pod placement	Cluster optimization	Realistic; cloud-native	UI-focused vs PPO logic	Supports modular design
Pei et al., 2024	Multi-agent DQN scheduling	MADQN model	Enhances task distribution	Multi-agent DRL	Good load balance; no PPO	Differs by agent focus	Not real-time
Blanco et al., 2024	DRL for DNN job scheduling	Online learning	Efficient in multi-tenant DNNs	Cost-sensitive DRL	Online-focused; app-specific	DNN-specific vs general	App-specific insight
Zhou et al., 2024	DRL in resource scheduling	Comprehensive review	PPO, DQN best for cloud	DRL scheduling map	Strong overview; no impl.	Supports our DRL use	Cited directly in Section 2
Russo et al., 2024	QoS RL in accelerator systems	Real-time sim	Meets QoS goals	Real-time RL	Hardware-specific	Similar to RELMAS	GPU-heavy focus
Mangalampalli et al., 2024	DRL scheduler in multi-cloud	PPO-based system	Boosts SLA, lowers cost	PPO for multi-cloud	Practical; no fault module	Base for our PPO model	Core reference
Wang et al., 2023	DRL for IoT edge tasks	DRLIS sim	Better latency, resource use	Fog-cloud DRL	Lacks baseline comparison	Similar to our setup	Aligned with iFogSim
Li et al., 2023	GNN multi-agent scheduler	GNN + RL model	Effective on ML tasks	GNN-enhanced RL	Complex training	Focus on edge AI	Not used in our pipeline
Sun et al., 2022	Load/deadline-aware DDDQN	Sim-based	Meets deadlines efficiently	Deadline-aware DRL	No real cloud setup	Closest to Pan et al.	Adds timing dimension
Liu et al., 2022	DRL for edge resource control	DRL in edge fog	Load-balanced scheduling	Load-aware DRL	Strong model; lacks cloud mix	Matches edge DRL scope	Good for fog tuning

2.4 Integrated Critical Reflection and Research Gap

The discussed readings reveal a helpful development of the employment of deep reinforcement learning (DRL) to make smart task scheduling in cloud, edge and fog systems. It is necessary to mention that PPO, DQN, and Actor-Critic have been identified as the key algorithms in terms of their flexibility and real-time decision-making (Zhou et al., 2024; Gu et al., 2025). Although such progress augers well in multi-cloud and fog-cloud infrastructures (e.g., Mangalampalli et al., 2024; Choppara & Mangalampalli, 2025), many of the studies are constrained to offline training (or they demand a lot of pretraining adaptation) and hence do

not allow 'real-time adaptation. Also, scalable decentralized DRL approaches (e.g., Sgambati et al., 2025), and multi-agent systems (Pei et al., 2024; Li et al., 2023) are often designed at the expense of additional overhead and communication overhead.

The current state of the art based on the DRL-based scheduling frameworks like DRLIS (in Wang et al., 2023) or RELMAS (in Blanco et al., 2024) performs well in the particular areas (IoT, DNN workloads), not being evaluated in a uniform way across a variety of cloud service providers and applications in the real-time SLA-oriented environment. Additionally, it is possible to note that majority of current research focuses too little on the use of explainability tools and does not consider complications related to reward sparsity or the compressed representation of the state that were mentioned by Gu et al. (2025).

This study fills these gaps by suggesting a PPO-based scheduler, operating in AWS, Azure, and GCP on the simulated fog-cloud environment (iFogSim2), but with a compact state vector, dynamic reward shaping, and modular performance logger. The system compares itself to the DQN, A2C, FCFS, and Round Robin thus creating a complete ecosystem and taking steps in powerful and explainable DRL scheduling that is SLA and cost-sensitive in contemporary multi-cloud.

3 Research Methodology

3.1 Research Approach

This study takes up an experimental, simulation-based approach of assessing intelligent task scheduling in multi-cloud settings with Proximal Policy Optimization (PPO). The essence is to create the simulation of the real-time task completion within a heterogenous collection of cloud suppliers (e.g., AWS, Azure, GCP), with the help of the iFogSim2 platform and have a PPO-based deep reinforcement learning (DRL) argument attempt to learn at the cloud supplier choice plan based on the nature of the tasks.

The method is modular and combines:

- Java-grounded simulation (iFogSim2) to instill a mockup of cloud/fog infrastructure.
- A Python implemented PPO model that was trained with Stable-Baselines3 library.
- A socket interface allowing real-time scheduling decision between the simulator and the learning agent.

This approach allows learning, flexibility and performance comparison of PPO to the classic (RR, FCFS) and cutting-edge (DQN, A2C) scheduling policies in the same workloads. The result is realistic and applicable since the simulated environment is transformed to the real-life latency, SLA restrictions, and cloud pricing structures.

The study uses a closed-loop simulation scheme that incorporates task generation, intelligent scheduling, execution and feedback. This is the cascade to follow:

- **Task Generation**: iFogSim2 Simulator: The simulator offers several types of tasks of different requirements in terms of SLA and resource consumption.

- **State Vector Formation**: The demand of the CPU, memory and SLA deadline are the major attributes obtained to initiate the creation of the task state vector.
- **PPO agent Policy Selection**: The policy selection of the cloud selection is done by passing the state vector to a trained PPO agent.
- **Task Execution and Monitoring**: Task is executed by the selected cloud and results regarding execution time and cost and SLA compliance are logged by iFogSim2.
- **Training**: The PPO agent is rewarded and modifies its policy when in training mode. In inference mode, no additional learning is done before a decision is arrived at.

3.2 Tools and Frameworks Used

The study incorporates a wide list of tools and frameworks in simulation, reinforcement learning, visualization, and data processing levels. Tool selection was dictated by the consideration of tools that should be modular, interoperable and capable of handling, real time intelligent scheduling of a multi-cloud model.

3.2.1 iFogSim2 (Java)

iFogSim2 is a version of the initial iFogSim simulator that has new attributes and focuses on edge-fog-cloud emulations with microservices, mobility as well as cluster support (Mahmud et al., 2022). It constitutes the main simulation engine in this project. To reflect heterogeneous support and latency, the custom classes were created that simulated the multi-cloud nodes that reflected the AWS, Azure, and GCP. It also captures important, task execution statistics such as SLA compliance, CPU utilization, and execution time. These logs are sent to the PPO agent trained and inferred.

3.2.2 PPO with Stable-Baselines3 (Python)

The alternative to the reinforcement learning agent is the PPO (Proximal Policy Optimization) that is conducted with the stable-baselines3 library in Python. This library has strong implementations of state-of-the-art DRL algorithms and supports full custom gym environments. PPO was chosen due to some stability in the learning and also continuous learning in a real-time scenario. Training targets state vectors of state features (e.g.,CPU demand, SLA duration, memory) and state values which are the reward signals created by SLA success and cost optimality.

3.2.3 Streamlit (Python)

Streamlit was used to create a real-time monitoring and write a visualization dashboard. It shows the results of scheduling choices, SLA values, cost values, and comparison of behaviors of various scheduling algorithms (RR, FCFS, DQN, A2C, PPO). Streamlit can perform interactive filtering, charting, and comparison of simulation logs to assist in studying model behavior and trends.

3.2.4 Socket Communication Interface (Java-python Bridge)

iFogSim2 (Java) is JavaScript based, whereas the PPO agent (Python) uses a custom TCP socket interface. After every simulation process, the iFogSim2 collects its task state vectors and presents it to the python server, which in response gives the chosen cloud provider according to the PPO model. The simulation progresses as such. With such an

interface, the simulation layer and the learning agent can also integrate with each other, and dynamic scheduling may take place without human intervention.

3.2.5 Supporting Libraries and Toolkits

- **Python**: The main language within which DRL agents are developed, as well as logs and dashboard fabrication.
- **Pandas / NumPy**: Pandas / NumPy is used to manipulate, preprocess, and aggregate logs.
- **Matplotlib / Plotly**: It helps in the incorporation of charts and plots on the dashboard.
- **Google Colab (GPU)**: It was used to train PPO model on GPU hardware with hardware acceleration.
- VS code / Eclipse: Python and Java developmental IDEs respectively.

The combination of these tools supported the development, training, simulation, and visualization parts of this study thus allowing the researcher to design, evaluate, and iterate an intelligent task scheduler on multi-cloud environments.

3.3 Dataset Description

This study will make use of a hybrid data test that can be related to both the synthetically created workloads of a task process that can be constructed through the iFogSim2 simulation as well as a cloud provider setup that is realistic representative of AWS, and the other clouds provided by Azure and GCP. The given dataset is considered the basis to train and test the Proximal Policy Optimization (PPO) agent.

3.3.1 Synthetic Task Workloads (iFogSim2 Generated)

iFogSim2 generates task tuples in a dynamic way depending on realistic IoT application behaviour. Every task (or tuple) consists of:

- **CPU Demand (in MIPS)**: The workload was randomized between low and high CPU workload.
- **RAM Need (MB)**: A simulation of memory hungry applications and light applications.
- **SLA Deadline (ms)**: This is one that is assigned to assess deadline-sensitive scheduling.
- **Start Time**: This defines the arrival of tasks distribution in the simulation.
- **Network Delay and Latency**: Simulates varying cost of communication among an edge, fog, and cloud.

A total of more than 4000 tasks were simulated both to train and test on, with the logs of their execution being stored as CSV files (i.e. ppo_training_dataset_cleaned_5f.csv) to study the resulting actions of the PPO in the long run.

3.3.2 Cloud Provider set ups

Three cloud nodes were set up so that they represented:

- **AWS**: More expensive and low latency node that provides better performance.
- **Azure**: The moderate-performance, moderate-cost, moderate-latency.

- **GCP**: Affordable, having a little more latency, modeling economic selecting resources.

Different MIPS capacities, bandwidth, RAM and price per CPU cycle were assumed in each node and each represents a diverse node in the real world. These set-ups are the same in different experiments so as to be comparable.

3.3.3 Offline Training Dataset (Pretraining PPO)

In the case of an offline training of the PPO, the logs collected during early simulations with random or heuristic schedulers (e.g. Round Robin or FCFS) and a dataset from kaggle (cloud_task_scheduling_dataset) were wrapped into a dataset constituting:

- **State Vectors**: Normalized CPU, SLA, RAM etc.
- **Actions**: The cloud provider was chosen.
- **Reward**: Depends on SLA and on cost penalty.

This was the training dataset (e.g, ppo_training_dataset_cleaned_5f.csv) which was utilised to pretrain PPO until it can be incorporated in live simulations.

3.4 DRL Environment Configuration

An artificial model of intelligent task scheduling in a multi-cloud set up has been established by developing a custom version of Proximal Policy Optimization (PPO) model powered by the stable-baselines3 library on Python to modulate agent-based schedules. It is on the basis of this that PPO agent was trained and ran in cooperation with the iFogSim2 simulation engine over worker via socket communication. The environment configuration is as listed below:

3.4.1 State Space

The basic features that dictate the scheduling actions are cast in a 5-dimensional state vector which represents each cloud task:

Table 3: Input features for DRL-based cloud scheduling

Feature	Description
CPU Demand (MIPS)	Task processing requirement
RAM Required (MB)	Memory footprint of the task
SLA Deadline (ms)	Expected task completion time
Current CPU Cost	Cost of executing task on each cloud (simulated)
Time Since Submission	Helps prioritize aged tasks

All features were normalized between 0 and 1 for stable training.

3.4.2 Action Space

The action space is finite and has the following 3 possible actions:

- 0 →AWS
- 1→ Azure
- 2 →GCP

The agent is trained that the cloud provider should be selected in such a manner that it minimizes cost and guarantees SLA adherence.

3.4.3 Reward Function

The reward function was constructed to have the balance between compliance and cost efficiency in SLA:

$$\text{Reward} = \begin{cases} +1 - 0.1 \times \text{NormalizedCost}, & \text{if SLA met} \\ -1 - 0.1 \times \text{NormalizedCost}, & \text{otherwise} \end{cases}$$

- **SLA met**: Task completed within the deadline.
- **Normalized cost**: Scaled between 0–1 to penalize expensive decisions.

This reward shaping encourages the agent to prioritize SLA satisfaction while minimizing CPU cost.

3.4.4 Offline Pretraining and Real-Time Inference

First, to decrease training time necessary to reach a certain level, the PPO agent was trained on historical simulation data (ppo_training_dataset_cleaned_5f.csv). After training, the model (ppo_v2.zip) was used in real time inference and would take as input task state vectors and would provide cloud decisions through a socket connection.

3.5 Simulation Workflow

Both parts are aimed at creating the overall system, a hybrid simulation-training loop, with iFogSim2 (Java) running a simulation of the tasks execution and PPO agent (Python) does an intelligent scheduling. The interconnection between the parts occurs through the TCP socket programming.

3.5.1 Step-by-Step Execution Flow

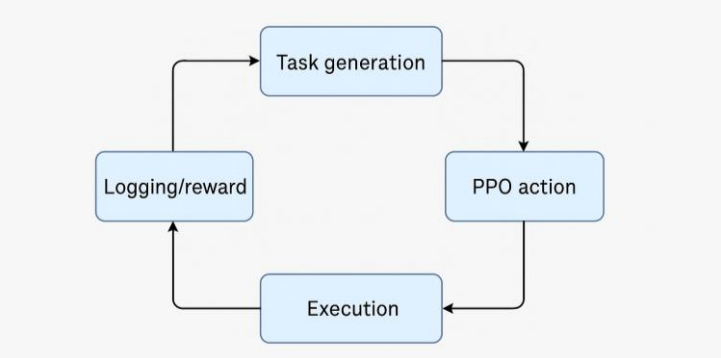

Figure 1: Research Workflow

The real-time simulation and decision-making loop follow these steps:

- **Task Generation**: iFogSim2 creates cloud jobs having different amounts of resource requirements (CPU, RAM) and SLA deadlines connected. Every task is translated into a made-up normal condition vector.

- **Cloud Selection Request to PPO:** iFogSim2 passes this task state vector to PPO socket server of Python.
- **Cloud Decision by PPO Agent**: PPO agent, pretrained on simulation logs analyses the state and returns an optimal cloud index (AWS = 0, Azure = 1, GCP = 2).
- **Task Execution and Metric Logging**: Using the PPO decision, the task is assigned to the picked cloud. The iFogSim2 simulates the task execution and keeps a track of:
    - Execution time
    - Cost
    - SLA compliance status
- **Reward Generation and PPO Update**: SLA and costs are the used metrics to calculate the reward. This reward, and next state are returned to the PPO agent in training mode to be used in real-time learning and policy updating. Under the inference mode, the model will simply record findings.
- **Result Logging and Dashboard Integration:** The metrics (SLA met, cost, cloud index, task ID) get recorded into a CSV file. These are monitored in the form of real-time performances through the Streamlit dashboard.

The overall system is designed as a hybrid simulation-training loop, where iFogSim2 (Java) simulates task execution and the PPO agent (Python) performs intelligent scheduling. Communication between the components is handled via TCP socket programming.

3.5.2 Hybrid Mode Execution

The system has a hybrid reinforcement learning:

- **Offline Training**: The PPO model undergoes training in advance based on simulation records that include task conditions, and actions along with the SLA results and amounts.
- **Online Inference w/ Logging**: When simulating, the trained PPO agent is fed state vectors and it gives cloud decisions with no live learning (.learn() is turned off). Nevertheless, rewards and outcomes are recorded to compare the performance of PPO in different loads.

This is a compromise between the efficiency of training and the practicability of deployment at real time without getting mired down by the updates of live models.

4 Design Specification

4.1 System Architecture Overview

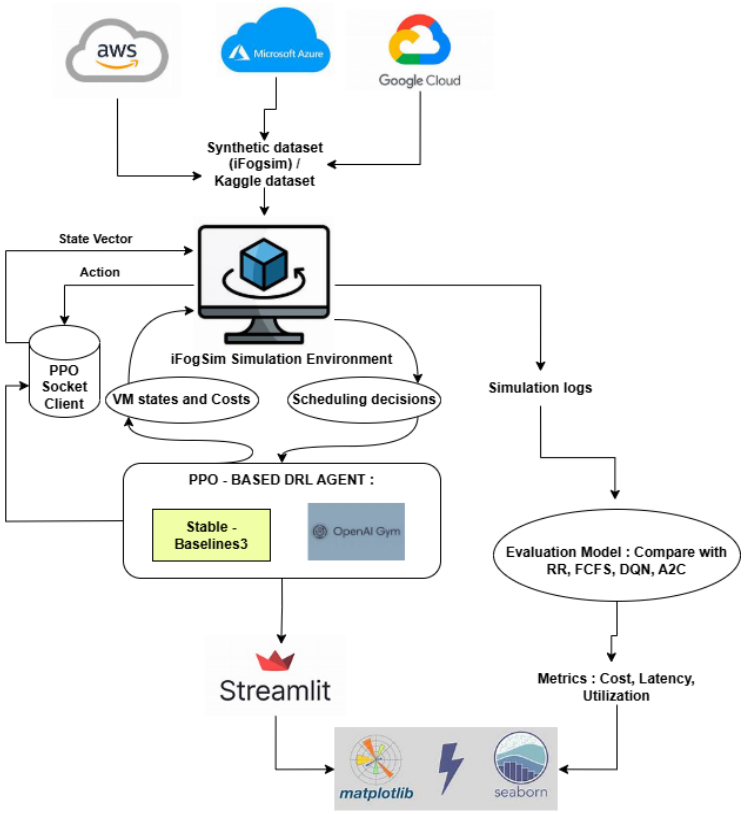

Figure 2: Architecture diagram

The system is built as a modular, real-time simulation and scheduling environment which rates intelligent task scheduling across a multi-cloud provider with a PPO-based DRL agent. The architecture is divided into the following major ingredients.

- **Task Workload Generator (iFogSim2):** This simulates various IoT based tasks whose CPU, memory, SLA requirements are randomly chosen.
- **Cloud Nodes (AWS, Azure, GCP):** Each cloud is modelled after with separate latency, cost and performance parameters.
- **State Vector Generator:** normalizes task features to a 5 dimensional state vector.
- **Socket-Based Scheduler Bridge:** Allows communicating the iFogSim2 (Java) and the PPO agent (Python).
- **PPO Scheduler (Python):** Agent trained with Stable-Baselines3 and Gym environment and is pre-trained to make scheduling decisions.
- **Logger and Dashboard (Streamlit):** Records task output, SLA, and cost to be visualized and analyzed in real time.

4.2 Reward System Design

Reward function is central to influence and drive towards behaviour of the PPO agent to make the important cloud selection decisions that maximise the compliance of SLA and the cost efficiency. The reward system of this project has been designed in such a way that two opposing goals are achieved:

- Maximizing SLA Success Rate
- Reducing Cost of Tasks being performed

The reward R of any task is a dependent of the following formula:

$$R = \text{SLA\_Success\_Reward} - \alpha \times \text{Normalized\_Cost}$$

Where:

SLA Success Reward = 1 assumes that the task is achieved in time of SLA, otherwise it is 0.
Normalized_Cost = actual CPU cost divided by (max-min) cloud-wise (min-max normalized).
α (alpha) = a factor of cost penalty factor (e.g. 0.1) with which SLA and cost priorities can be balanced.

Reward Design justification

- **SLA Priority First**: The main driver is the SLA deadlines; not beating them can make the task effectively useless in the real-world solutions (e.g. IoT alerts).
- **Cost Sensibility**: Of all solutions that meet the SLA, the agent will be billed at a higher cost when it opts to choose clouds that are more costly than it should be. This will also promote better economic choices.
- **Smooth Learning**: Normalizing of the value of the cost avoids reward spikes and stabilises the PPO learning.

Examples Scenarios

Table 4: Reward calculation based on SLA outcome and normalized execution cost for each scheduled task.

SLA Met	Cloud Used	Normalized Cost	Reward Calculation	Final Reward
Yes	Azure	0.4	1 - 0.1 × 0.4	0.96
No	AWS	0.2	0 - 0.1 × 0.2	-0.02
Yes	GCP	0.1	1 - 0.1 × 0.1	0.99

The reward structure increases SLA-first, cost conscious cloud choices.

4.3 Dashboard Design

In order to track the performance of the PPO based scheduler in real time, as well as compare it with other algorithms, an interaction dashboard was created with the help of Streamlit. This dashboard allows easy understanding of task scheduling measures like SLA compliance, cost and cloud usage.

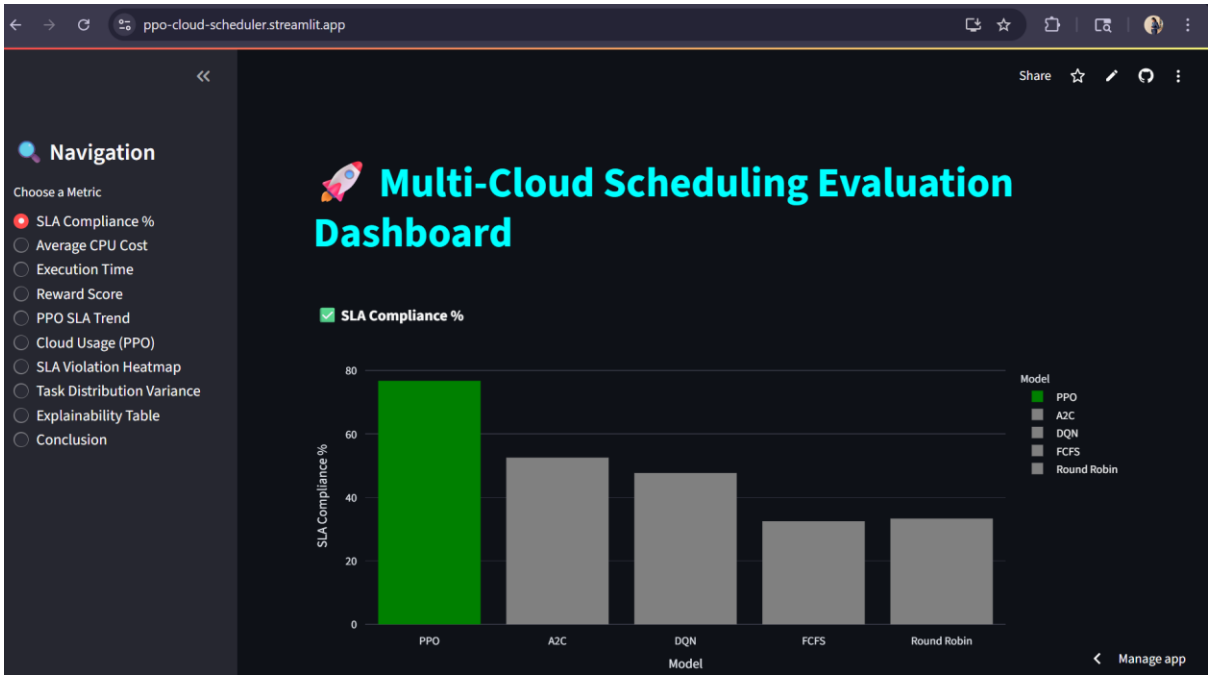

Figure 3: Streamlit Dashboard

- **SLA Compliance %**
  A bar chart shows the percentage of successful tasks that they reached their SLA before time under each algorithm (PPO, A2C, DQN, FCFS, Round Robin).
- **Average CPU Cost**
  The line plots or the bar charts trace average computational cost incurred per task across the schedulers.
- **Cloud Usage Distribution**
  Pie charts depict the percentage of activity abdicated to each cloud provider (AWS/azure/GCP) of each scheduler.
- **Execution Time Distribution**
  Aids to measure efficiency in task processing and latency in cloud resources.
- **SLA Violation Heatmap**
  Displays the cloud types or the types of tasks that tend to suffer although SLA violations are more likely to occur in the long term.
- **Reward Score Trend (PPO)**
  Line plot where reward scores are plotted against the simulation episodes which proves useful in training phase.

5 Implementation

The proposed system is an extension of the modular hybrid system of clouds simulation and reinforcement learning as well as real-time scheduling. The simulation was developed in iFogSim2 (Java), whereas reinforcement learning was based on Proximal Policy Optimization (PPO) which was developed in Python with the help of libraries such Stable-Baselines3. A custom TCP socket interface was created in order to facilitate the interaction between these two components so as to be able to send the state vectors created using the task features in iFogSim2 to the PPO agent and the decision of the cloud to be received dynamically.

The possible resources that were assigned to each simulated task in iFogSim2 were based on realistic requirements like CPU (MIPS), RAM and SLA Deadlines. These were transformed into normalized state vectors and subjected to the PPO model based on Python to make the schedules. These vectors were passed through a PPO agent pretrained on historical simulation logs and a cloud selection index was returned: AWS (0), Azure (1) or GCP (2) by the agent according to its policy. iFogSim2 then performed the task on the chosen cloud, recording the time of the execution, its cost and SLA.

In assisting the PPO training, logs were logged during simulation of actual state, action, reward and next-state transition. The model trained in hybrid mode by using data such as ppo_training_dataset_cleaned_5f.csv (offline) that is then used in inference when the model is deployed in simulation with a possibility of reward logging to track performance. This design reduced the amount of overhead incurred during run time and still allowed future extensions to online learning to be introduced when necessary.

Each of the performance metrics, including SLA compliance, distribution in the cloud, and the price of the CPU was saved in structured CSV logs. Based on these logs, a real-time monitoring interactive Streamlit dashboard was filled. The dashboard provided graphical summaries of metrics like cloud uptake charts, SLA percent, average task costs by the use of several scheduling strategies providing clear details into how effective the PPO agent is.

The whole system was powered by a local machine which had 16 GB RAM, multi-core CPU, and optional GPU (Google Colab) employing which offline model training was being done. The main development environments were Python (and their add-ons Pandas and NumPy, Matplotlib, Plotly, and Streamlit) and Java (with Eclipse). This application allowed end-to-end assessment functionality- full assessment loop, including the synthetic generation of the task on the one hand, and intelligent scheduling and visualization on the other, recreating a real-world DRL-based multi-cloud scheduling pipeline.

6 Evaluation

6.1 Case Study 1: PPO vs Baseline Schedulers and Modern DRL Models

PPO was evaluated against traditional (Round Robin, FCFS) and modern DRL-based (A2C, DQN) schedulers using a common workload of 4000+ tasks. All schedulers ran under identical cloud configurations. Performance was assessed using SLA compliance, average CPU cost, and execution time.

Traditional Schedulers vs PPO

Table 5: Performance comparison of heuristic scheduling algorithms based on SLA compliance, average CPU cost, and execution time

Scheduler	SLA (%)	Avg. CPU Cost	Avg. Execution Time (ms)
PPO	**76.75**	**128.2**	**0.70**
RR	33.34	127.0	1.25
FCFS	32.50	128.2	1.26

PPO achieved the highest SLA and lowest cost. FCFS was marginally faster due to its simplicity.

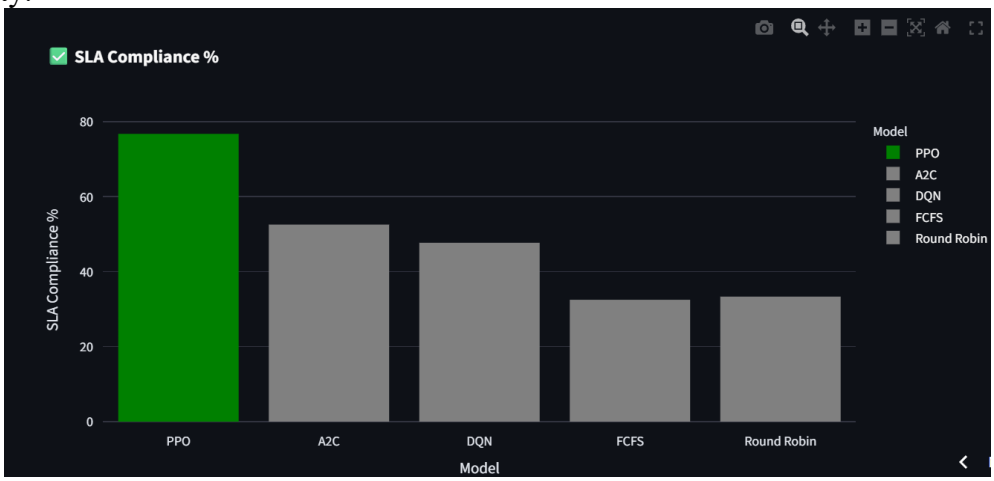

Figure 3: SLA compliance (%) across different schedulers

PPO vs DRL Models

Table 6: Performance comparison of DRL algorithms based on SLA compliance, average CPU cost, and execution time

Scheduler	SLA (%)	Avg. CPU Cost	Avg. Execution Time (ms)
PPO	**76.75**	**128.2**	**0.70**
A2C	52.55	125.7	0.81
DQN	47.70	158.6	0.86

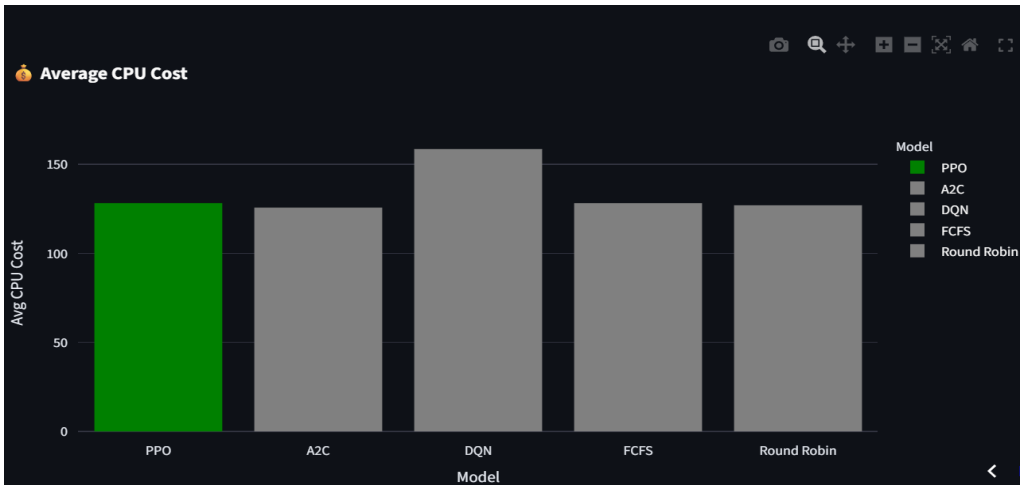

Figure 4: Average CPU cost across different schedulers

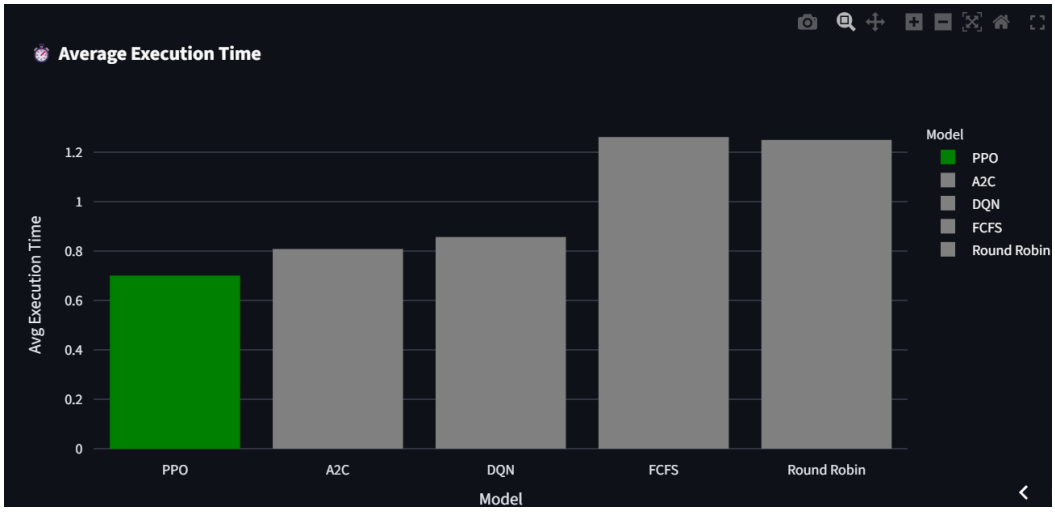

Figure 5: Average Execution time across different schedulers

PPO outperformed A2C and DQN across all metrics, demonstrating stronger generalization and better scheduling stability.

```
=== Scheduler Evaluation Summary ===

Scheduler  SLA (%)  CPU Cost  Time (ms)
      PPO    76.75     128.2       0.70
       RR    33.34     127.0       1.25
     FCFS    32.50     128.2       1.26
      A2C    52.55     125.7       0.81
      DQN    47.70     158.6       0.86

🏆 Best Model:
Model: PPO, SLA: 76.75%, CPU Cost: 128.2, Time: 0.7 ms
```

Figure 6: Evaluation output showing PPO as the best scheduler with highest SLA and lowest execution time

6.2 Case Study 3: PPO Performance Across Cloud Types (AWS, Azure, GCP)

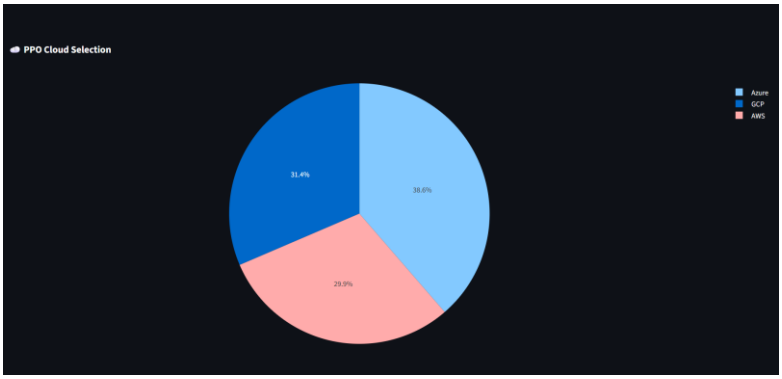

Figure 7: PPO cloud selection distribution across AWS, Azure, and GCP

Observation:

- AWS (high cost, high performance) was chosen for latency-sensitive tasks.
- GCP (low cost, high latency) was selected for cost-effective workloads.
- Azure had balanced usage.

PPO effectively balances SLA vs cost tradeoffs and dynamically adapts cloud selection based on task characteristics.

6.3 Discussion

These experiments demonstrate that PPO performs much better than the heuristic and modern DRL baselines, namely, in real-time cloud task scheduling tasks. These findings provide further support to the claims of PPO of learning cost-SLA tradeoffs and generalization across diverse types of tasks.

Limitations:

- No real-world API/service was integrated (e.g. AWS billing) and it was not modeled in costs: this was simulated.
- Final inference runs disabled live training (.learn()) because it is painfully slow in simulation.
- No SHAP/LIME explanations were provided, which put a strain on interpretability.

Suggested Improvements:

- Future iterations could enable real-time learning with adaptive reward tuning.
- To achieve more clarity of PPO decision-making, explainability modules (SHAP) may be inserted.
- A more realistic simulation of pricing variability could be achieved by means of Integrating real cloud APIs or simulating them.

7 Conclusion and Future Work

7.1 Conclusion

- Proved the possibility of the intelligent task scheduling (PPO-based) in heterogeneous multi-clouds.
- PPO agent was cost effective and would always comply with SLA better than Round Robin, FCFS, DQN and A2C.
- Realized a hybrid simulation- training architecture that combined iFogSim2 (Java) and PPO (Python) through socket messages.
- Formulated a standardised 5 dimensions state vector and specialised SLA-cost aware reward function.
- Real-time inference was used in conjunction with offline pretraining to make it more adaptable.
- The visualization, which is designed on streamlit, affirmed the fact that PPO was able to demonstrate better SLA performance and cheaper CPU with workloads.

7.2 Future Work

- Realistic simulation: Add live task streams and real-time cloud pricing APIs in order to increase the degree of realism.
- Model explainability: Include explainable DRL methods (e.g., SHAP, LIME) to incorporate higher visibility and credibility.
- Scalability: Scale test to large, federated, or distributed multi-cloud environments.
- Energy efficiency: Combine PPO scheduling with energy-aware policies for sustainable operations.
- Real-world use: Distribute as a plug-in intelligent schedule engine to support DevOps pipeline, hybrid cloud scheduling, and SLA based automatic provisioning.

References

1. Mahmud, R., Pallewatta, S., Goudarzi, M. and Buyya, R., 2022. IFogSim2: An extended iFogSim simulator for mobility, clustering, and microservice management in edge and fog computing environments. Journal of Systems and Software, 190, p.111351.
2. Das, P. and Rao, V., 2023. Energy and cost optimization in edge-cloud systems using PPO. Cluster Computing, 26(5), pp.2941–2956.
3. Zhou, G. et al., 2024. Deep reinforcement learning-based methods for resource scheduling in cloud computing. Artificial Intelligence Review, 57(5), p.124.
4. Mangalampalli, S., Karri, G.R. and Mohanty, S.N., 2024. Efficient deep reinforcement learning based task scheduler in multi cloud environment. Scientific Reports, 14(1), Article 21850.
5. Choppara, P. and Mangalampalli, S., 2025. An efficient deep reinforcement learning based task scheduler in cloud fog environment (DRLMOTS). Cluster Computing, 28(1), Article 67.
6. Sgambati, M., Vakanski, A. and Anderson, M., 2025. Decentralized Distributed PPO for HPC Scheduling. arXiv preprint.
7. Gu, Y., Liu, Z., Dai, S. et al., 2025. Deep Reinforcement Learning for Job Scheduling and Resource Management in Cloud Computing: An Algorithm-Level Review. arXiv preprint.
8. Pei, L., Zeng, X. and Zhang, Y., 2024. MADQN-based task scheduling for multi-agent cloud systems. Journal of Cloud Computing.
9. Pan, J., Wei, Y. and Meng, L., 2025. Dual-layer DQN scheduling framework for dynamic cloud workloads. Journal of Supercomputing.
10. Wang, Z., Goudarzi, M., Gong, M. and Buyya, R., 2023. DRLIS: Deep reinforcement learning for IoT/Edge task scheduling in fog-cloud environments.
11. Xu, Y., Gong, Y., Zhou, Y., Bao, Q. and Qian, W., 2024. Enhancing Kubernetes automated scheduling with deep learning and reinforcement techniques for large-scale cloud computing optimization. Computers & Industrial Engineering, 190, p.109066.
12. Ismail, A.A., Khalifa, N.E. and El Khoribi, R.A., 2025. Survey: DRL scheduling strategies in multi-access edge computing. Cluster Computing.
13. Li, Y., et al., 2023. GNN-based multi-agent RL scheduler for ML workloads in edge computing
14. Liu, Q., Xia, T., Cheng, L. et al., 2022. DRL for load-balancing-aware resource control in edge computing. IEEE Transactions on Parallel and Distributed Systems, 33(6). DOI: 10.1109/TPDS.2021.3116863
15. Blanco, F.G., et al., 2024. RELMAS: DRL-based online scheduling for multi tenant DNN systems.

16. Russo, E., et al., 2024. Real-time QoS scheduling via reinforcement learning in multi-accelerator systems.
17. Sun, C., Yang, T. and Lei, Y., 2022. DDDQN TS: Load- and deadline-aware task scheduling in cloud systems. International Journal of Intelligent Systems, 37(11), pp.9138–9172.