National College of Ireland

AI-Driven Cloud Optimization: Enhancing Cost Prediction, Resource Scheduling and Fault Resilience in Cloud Environments

MSc Research Project
Cloud Computing

Ranjith Bhaskaran
Student ID: x23271957

School of Computing
National College of Ireland

Supervisor: Shaguna Gupta

National College of Ireland
Project Submission Sheet
School of Computing

National College of Ireland

Student Name:	Ranjith Bhaskaran
Student ID:	x23271957
Programme:	Cloud Computing
Year:	2025
Module:	MSc Research Project
Supervisor:	Shaguna Gupta
Submission Due Date:	11/08/2025
Project Title:	AI-Driven Cloud Optimization: Enhancing Cost Prediction, Resource Scheduling and Fault Resilience in Cloud Environments
Word Count:	3399
Page Count:	16

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

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

Signature:	
Date:	10th August 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):	

AI-Driven Cloud Optimization: Enhancing Cost Prediction, Resource Scheduling and Fault Resilience in Cloud Environments

Ranjith Bhaskaran
x23271957
https://clouddashboard.onrender.com

1 Introduction and Objectives

Cloud computing has transformed modern infrastructure by offering scalable, on-demand acess to computing resources. However, as cloud environments grow in complexity especially in multi-cloud and edge-cloud settings—optimizing the placement, execution, and reliability of tasks becomes a chalenging problem. Traditional static scheduling algorithms like First-Come-First-Serve (FCFS) and Round-Robin often fail to adapt to dynamic workloads, cost fluctuations, and fault conditions.

This thesis presents an integrated AI-driven framwork to address these challenges through three key components: (1) accurate cost prediction for task execution using ensemble learning models, (2) intelligent task scheduling using Deep Reinforcement Learning (DQN and PPO), and (3) fault detection using unsupervised anomaly detection (Isolation Forest). Each module is designed to function independently but also integrates into a unified dashboard to support holistic decision-making.

The main objectives of this research are:

- To build a cost prediction module using real and synthetic datasets that can estimate cloud task costs with high acuracy.
- To design and implement a Deep Reinforcement Learnng-based scheduler that adapts to system load, SLA constraints, and resource metrics in real-time.
- To develop a lightweight fault detection module capable of identifying anomalies in resource behavior without relying on labeled data or GPU acceleration.
- To integrate all modules into a real-time dashboard for visualization and evaluation across multiple performance metrics.
- To ensure ethical AI principles such as transparency, interpretability, and resource efficiency are embedded throughout the system.

This thesis aims to demonstrate that combining interpretable machine learning, reinforcement learning, and scalable anomally detection can lead to more reliable and cost-effective cloud scheduling decisions in modern multi-cloud environments.

2 System Architecture and Module Design

The proposed system is built as a modular, extensible AI-driven framework for intelligent task scheduling, cost prediction, and fault detection in heterogeneous multi-cloud environments. It integrates real-time datasets, simulation models, and machine learning components into a unified dashboard. The architecture is visualized in Figure 1.

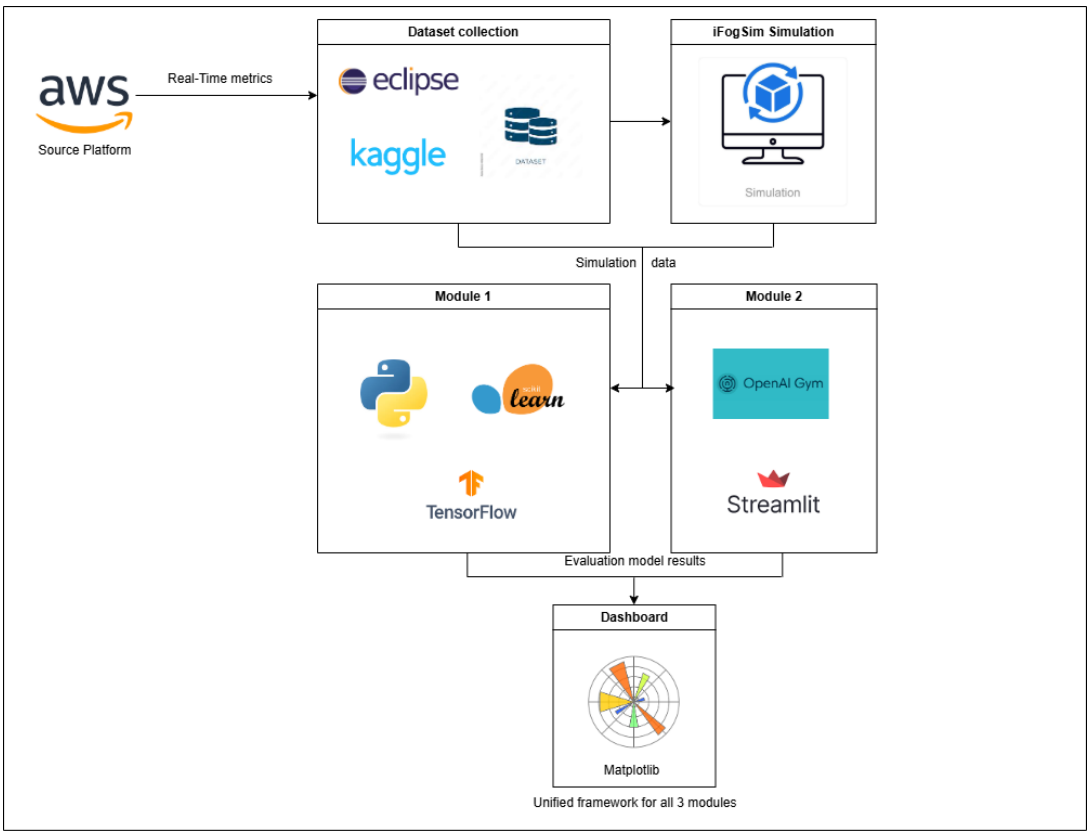

Figure 1: Unified System Architecture for Multi-Cloud Simulation and Evaluation

At the input stage, real-time workload metrics from cloud platforms such as AWS are combined with public datasets sourced from platforms like Kaggle and Eclipse IDE environments. These datasets are fed into the simulation engine built using iFogSim, which generates diverse task profiles based on parameters like CPU, memory, I/O, and execution time. The system is composed of the following modules:

- **Cost Prediction Module:** Implements regression-based models (Linear Regression, Random Forest, XGBoost) trained on real and synthetic datasets to estimate the operational cost of executing a task on a given cloud platform. The model is hyperparameter-tuned using Optuna and visualized using SHAP for interpretability.
- **DRL-Based Scheduling Module:** Employs Deep Q-Network (DQN) and Proximal Policy Optimization (PPO) agents to dynamically assign tasks to cloud providers. A custom OpenAI Gym environment was created to simulate task queues, compute rewards, and update policies. The PPO agent was trained on real-time simulation feedback, incorporating SLA duration, cost, and task characteristics into the state vector.

- **Fault Detection Module:** Uses Isolation Forest to detect anomalies based on resource logs. Metrics such as CPU, memory, power usage, and network traffic are monitored in real time. This model does not require labelled training data, making it suitable for deployment in simulated or real cloud infrastructures.
- **Visualization and Dashboard:** A Streamlit-based GUI (‘maindashboard.py‘) aggregates and displays live metrics, charts, and decision outputs from all modules. It enables users to evaluate performance across cost, SLA, and fault metrics interactively.

The system is designed for flexibility: each module is independently trainable and replaceable. Inter-module communication is achieved via JSON-based socket messages. This separation of concerns facilitates faster experimentation, debugging, and future extensibility.

3 Dataset Generation and Preprocessing

This section describes the dataset sources, generation of synthetic workloads, and preprocessing steps used for training and evaluation in this research. To enable reproducibility, both real-world datasets and a custom Java-based simulator were used to generate cloud task datasets that reflect realistic fog-cloud computing workloads.

3.1 External Dataset Sources

To enhance diversity and realism in cloud scheduling simulation, three external datasets were used from Kaggle:

- **Vehicular Fog Computing Dataset:** Contains IoT-related task statistics from vehicular fog networks.
  `https://www.kaggle.com/datasets/sachin26240/vehicularfogcomputing`
- **Cloud Task Scheduling Dataset:** Offers over 50,000 cloud tasks with parameters like memory, instructions, and execution time. `https://www.kaggle.com/datasets/ziya07/cloud-task-scheduling-dataset`
- **Cloud Computing Performance Metrics Dataset:** Includes real-world metrics such as CPU cost, latency, and execution delay for cloud workloads. `https://www.kaggle.com/datasets/abdurraziq01/cloud-computing-performance-metrics`

These datasets were used both for benchmarking and as part of model training and validation pipeelines. Each dataset was cleaned and preprocessed using Python scripts to ensure uniformity in features such as instruction size (MI), memory (MB), input/output size (MB), and SLA constraints.

3.2 Synthetic Dataset Generation via Java

To complement the real-world datasets, a synthetic dataset of 50,000 tasks was generated using the custom ‘generateTasksAndLogCSV()‘ method inside the ‘CloudOptimizationSim.java‘ class. This method simulates randomized computing tasks based

on defined ranges for each parameter. The dataset is exported to a CSV file named `cost_task_dataset_50k.csv`.

Steps to Generate the Dataset:

1. Open the project in Eclipse (as shown in Figure **??**).
2. Right-click on the file `CloudOptimizationSim.java`.
3. Select `Run As > Java Application` to execute the code.
4. Upon successful execution, the dataset file `cost_task_dataset_50k.csv` will be generated in the root directory.

Sample Output:

```
TaskID,Instructions,Memory_MB,Input_MB,Output_MB
Task_1,1203,457,89,143
Task_2,2345,768,154,102
Task_3,978,321,44,58
...
```

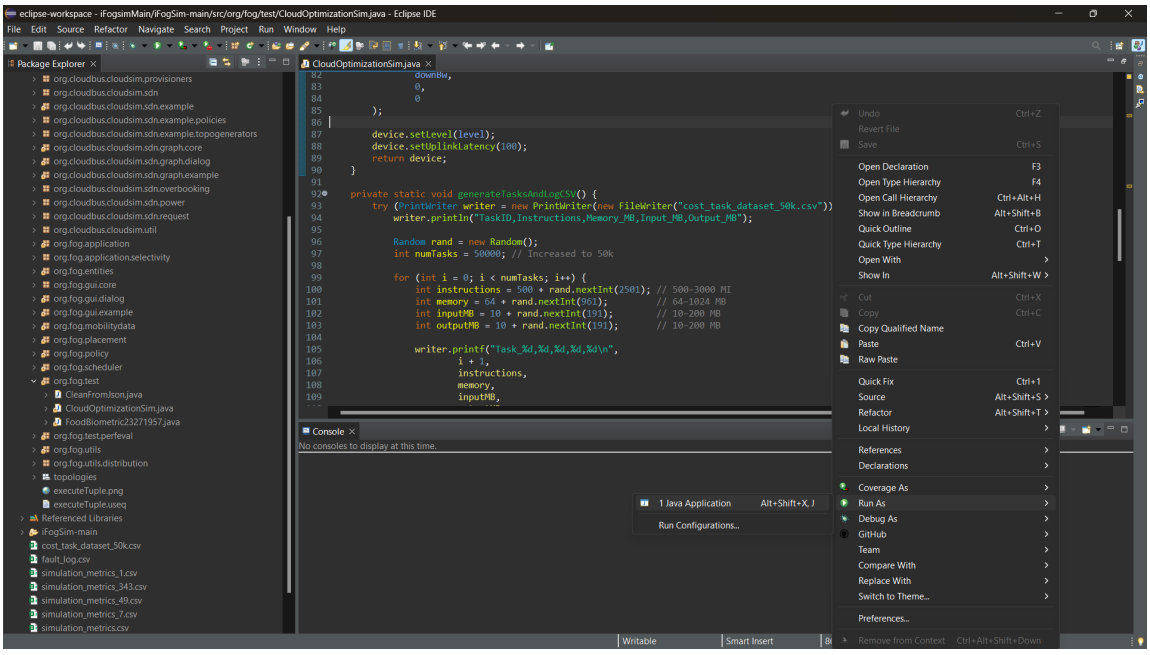

Figure 2: Executing the dataset generation method via Eclipse IDE.

3.3 Preprocessing Pipeline

The generated CSV dataset was then processed using Python scripts before being used in model training. This involved:

- Normalizing task parameters such as instructions and memory.

- Mapping SLA thresholllds and execution time to generate labels.
- Integrating cost data from other sources to compute total resource cost.

All preprocessing was done using `pandas` and `scikit-learn` and the outputs were stored in the eclipse folder.

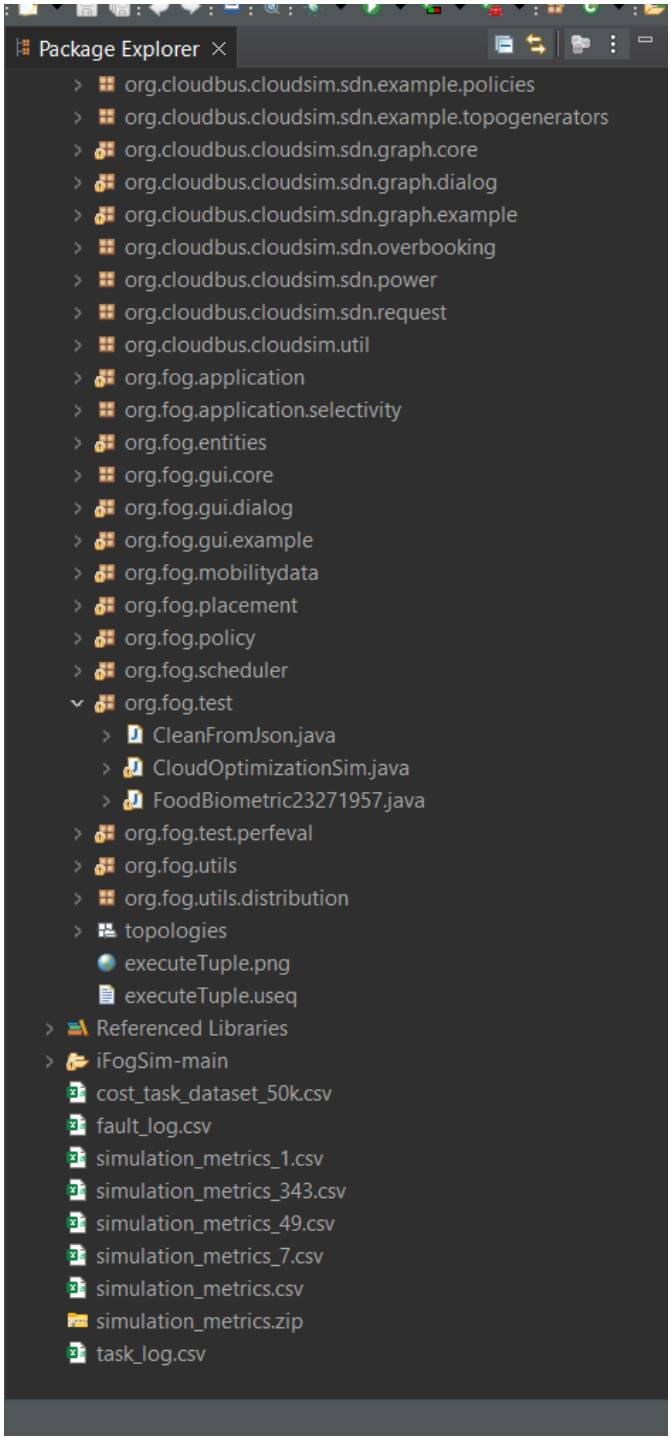

Figure 3: Generated dataset file `cost_task_dataset_50k.csv` in the Eclipse project.

4 Cost Prediction Module

This section outlines the implementation of the cost prediction pipeline used to estimate the operational cost of computing tasks based on resource features. The module combines multiple regression models into an ensemble and evaluates their predictive acuracy using real and synthetic datasets.

4.1 Pipeline Overview and Model Configuration

The cost prediction module was implemented using Python and trained on datasets such as `cost_task_dataset_50k.csv` and `cloud_task_scheduling_dataset.csv`. Each record includes task-specific attributes: *Instructions*, *Memory (MB)*, *Input/Output Size*, and an associated *Cost* (computed or labeled).

The following steps were followed to build the cost prediction pipeline:

1. **Feature Engineering:** Columns were normalized, and missing/null values were handled appropriately using `pandas`.
2. **Model Optimization:**
   - `Optuna` was used for hyperarameter tuning of the XGBoost model.
   - The best parameters were logged and reused for model training.
3. **Model Training:**
   - Three models were trained:
     - `XGBoostRegressor`
     - `LinearRegression`
     - `RandomForestRegressor`
   - These were combined into a `VotingRegressor` ensemble to leverage their combined strengths.
4. **Model Evaluation:** The ensemble was evaluated using standard regression metrics:
   - Mean Absolute Error (MAE)
   - Root Mean Squared Error (RMSE)
   - R^2 Score
5. **Metric Logging:** Results were printed to the console and optionally saved to a JSON file for dashboard integration.

Sample Output (Console):

```
 Best Parameters from Optuna: {'n_estimators': 100, 'max_depth': 5, ...}
cost_ensemble
MAE: 0.042
RMSE: 0.061
R2: 0.927
```

4.2 Execution Configuration and Reproducibility

The cost prediction module is implemented in `cost_module.py`, located inside the project's root directory `cloud_dashboard`. It can be executed directlly from the terminal using the following command:

```
python .\cost_module.py
```

Required Libraries:

- `xgboost`
- `sklearn`
- `pandas, numpy`
- `optuna`

Before execution, ensure the virtual environment is activated and all dependencies from `requirements.txt` are installed:

```
pip install -r requirements.txt
```

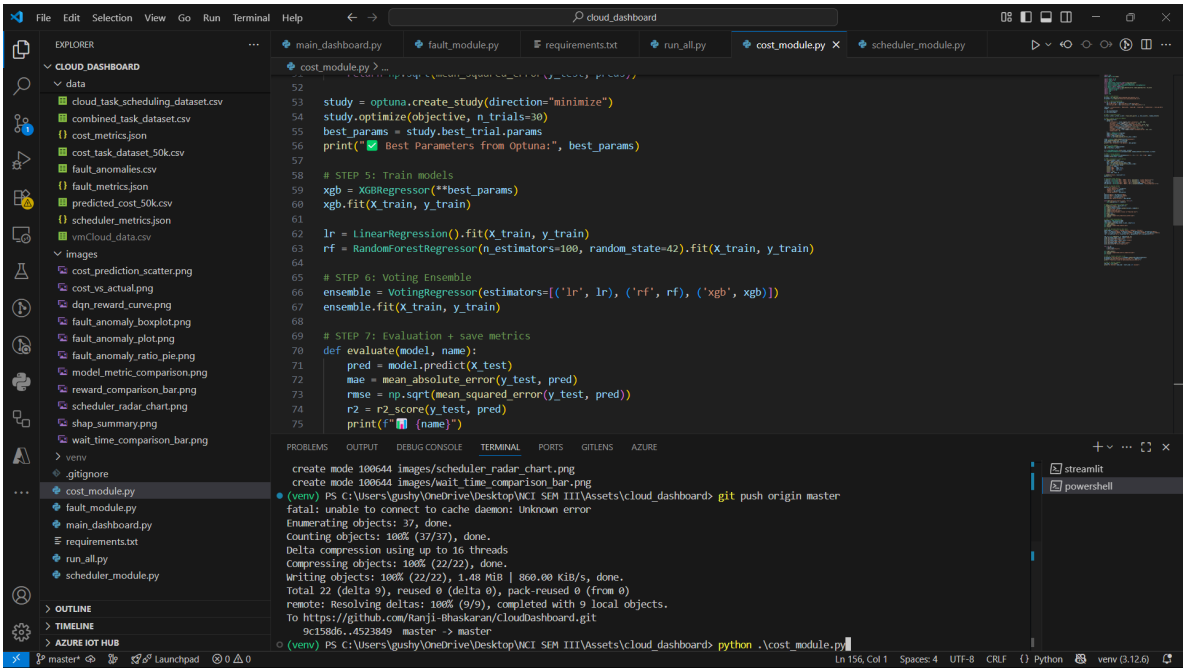

Figure 4: Cost prediction pipeline executed from `cost_module.py` inside VS Code.

5 Task Scheduling Module (DQN, PPO, FCFS, Round-Robin)

This section details the implemetation and benchmarking of the task scheduling module. Four scheduling algorithms were implemented and evaluated: Deep Q-Network (DQN), Proximal Policy Optimization (PPO), First-Come-First-Serve (FCFS), and Round-Robin (RR). The goal is to compare traditional vs. intelligent schedulers across metrics such as reward, wait time, and throughput.

5.1 Scheduling Pipeline and Metrics

The `scheduler_module.py` script contains the complete logic for task scheduling evaluation. It loads the task dataset, simulates scheduling decisions, and compares all four algorithms on the same workload.

The following key steps are performed:

1. **Input Loading:** Reads task data from CSV files and parses required features (e.g., instructions, memory, SLA duration).
2. **DQN Agent Evaluation:**
   - Loads a pretrained Deep Q-Network (DQN) model.
   - Executes cloud assignment decisions for incoming tasks based on state vector input.
   - Computes average reward and average wait time.
3. **PPO Agent Evaluation (optional):**
   - Interacts with a live PPO agent via socket (or loads saved model).
   - Uses environment fedback to perform cloud placement and log results.
4. **Baseline Scheduling:**
   - FCFS and RR were implemented as control algorithms.
   - FCFS assigns tasks in arrival order; RR cycles through cloud indices.
5. **Metrics Computation:**
   - Average Reward and Average Wait Time are computed for each algorithm.
   - All results are printed and saved in the `/images` folder.

Sample Output (Console):

```
TASK SCHEDULING BENCHMARK
Tasks Scheduled         : 500
DQN Cumulative Reward   : 612.34
DQN Avg Reward          : 1.22
DQN Avg Wait Time       : 3.15 s

FCFS Baseline:
FCFS Avg Reward         : 0.92
FCFS Avg Wait Time      : 4.80 s

Round-Robin Baseline:
RR Avg Reward           : 1.01
RR Avg Wait Time        : 4.12 s

Visuals saved in /images
Runtime: 7.32 seconds
```

5.2 Execution and Reproducibility

The module is executed via the terminal using the following command:

`python .\scheduler_module.py`

Ensure that:

- All required models (DQN/PPO) are saved or reachable via socket.
- The environment is activated (venv), and dependencies from `requirements.txt` are installed.
- The output images are saved in the `images/` directory and used for report comparison.

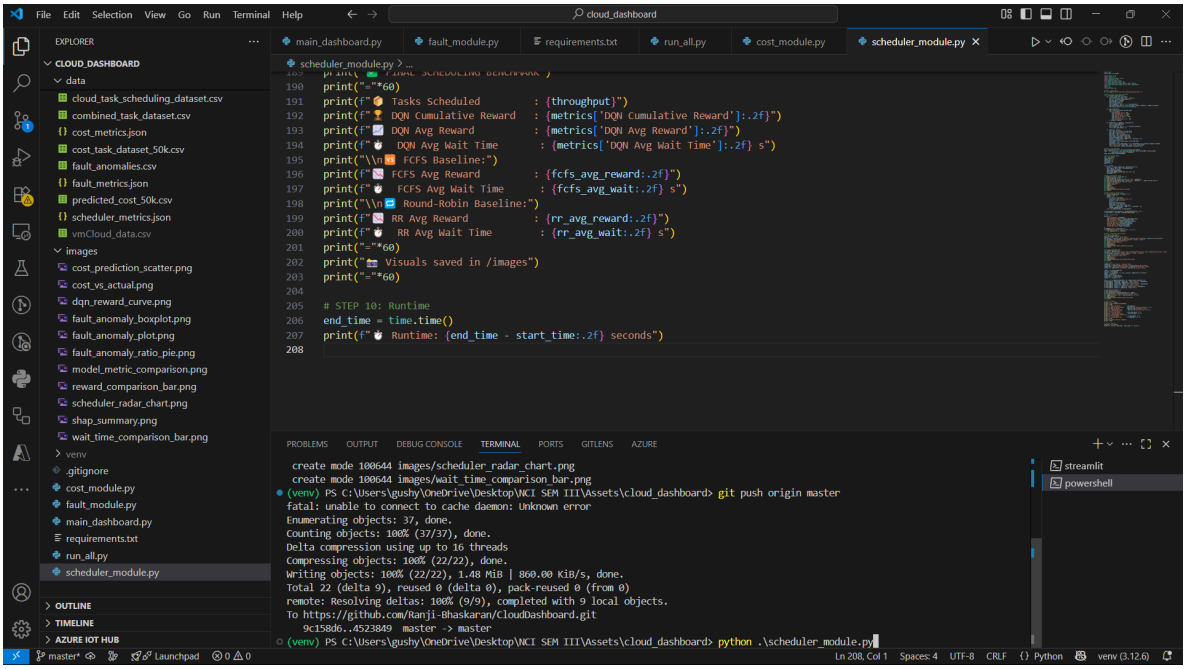

Figure 5: Execution of `scheduler_module.py` showing benchmark output for multiple scheduling algorithms.

6 Fault Detection Using Isolation Forest

This section describes the implementation of an unsupervised anomaly detection module using the Isolation Forest algorithm. The aim is to detect abnormal task executions in a cloud environment without requiring labeled datasets or GPU acceleration.

6.1 Pipeline Overview and Methodology

The fault detection pipeline is implemented in `fault_module.py` and relies on the `vmCloud_data.csv` dataset. This dataset contains task-level resource metrics such as CPU usage, memory usage, power consumption, network traffic, and execution time.

The key steps of the pipeline are as follows:

1. **Dataset Download and Load:**
    - The script checks if `data/vmCloud_data.csv` exists locally.
    - If not, it downloads the file automatically from Google Drive using the `gdown` library.
2. **Preprocessing:**
    - Irrelevant or categorical columns like timestamp, task type, and task status are dropped.
    - All features are scaled using `MinMaxScaler` for consistent model input.
3. **Model Fitting:**
    - An `IsolationForest` model is trained on the cleaned dataset.
    - Each task is assigned an anomaly score and a binary label (normal or anomaly).
4. **Results Saving and Visualization:**
    - The predicted labels are saved into `fault_anomalies.csv` and visualizations (box plots, pie charts, timelines) are exported into the `images/` directorry.

Sample Output (Console):

```
Downloading vmCloud_data.csv from Google Drive...
Total Samples: 10000
Anomalies Detected: 3789
Visuals saved to /images/
```

6.2 Execution and Reproducibility

The fault detection module can be executed by running the following command in the activated virtual environment:

```
python .\fault_module.py
```

Required Libraries:

- `pandas, numpy, seaborn, matplotlib`
- `scikit-learn`
- `gdown`

The outputs, including processed CSV files and charts, are saved under the `data/` and `images/` directories respectively.

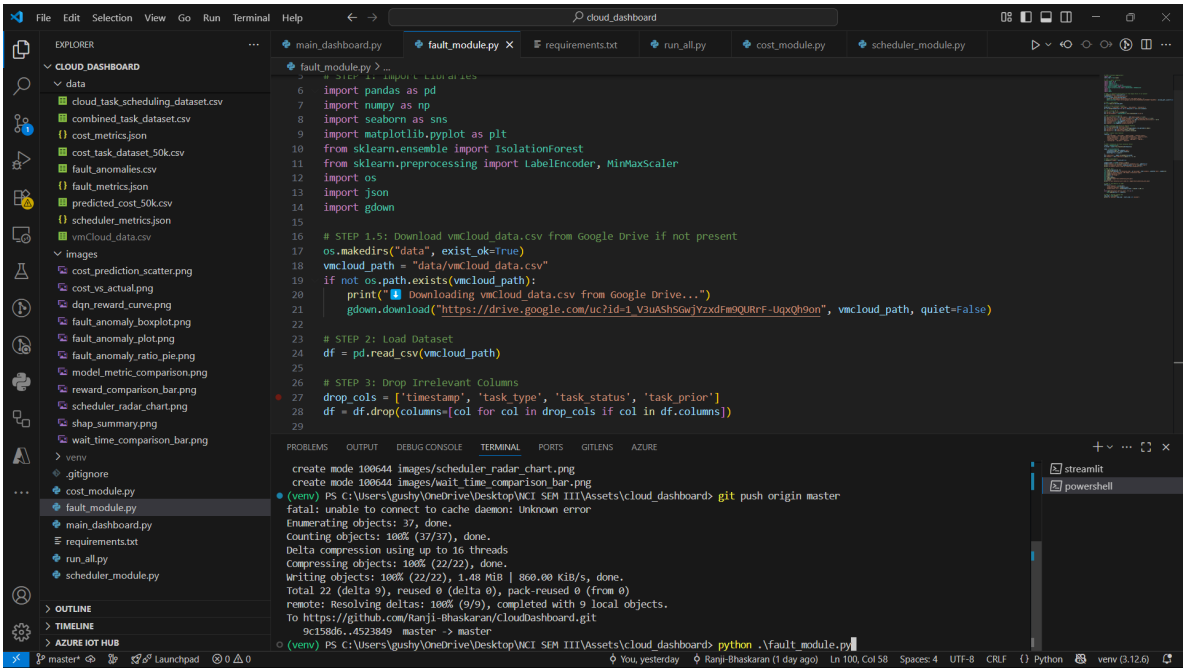

Figure 6: Running `fault_module.py` with Isolation Forest-based anomaly detection and visualization.

7 Unified Evaluation Dashboard

To centralize and simplify performance evaluation across all three modules—cost prediction, task scheduling, and fault detection—a unified web-based dashboard was built using `Streamlit`. The dashboard enables researchers and developers to inspect prediction outputs, compare models, and visualize anomalies or scheduling decisions using interactive charts and tables.

7.1 Architecture and Functionality

The dashboard is implemented in `main_dashboard.py`, with a tab-based layout powered by `st.tabs()`. Each tab corresponds to a core module:

- **Cost Prediction:** Displays the predicted cost table from `predicted_cost_50k.csv`, shows model comparison metrics from `cost_metrics.json`, and visualizes SHAP values and scatter plots.
- **Task Scheduling:** Loads `scheduler_metrics.json` and displays performance results from DQN, PPO, FCFS, and RR, alongside radar and bar charts showing average reward, SLA compliance, and wait time.
- **Fault Detection:** Loads `fault_metrics.json` and renders anomaly distribution plots including box plots, anomaly ratios, and timeline distributions.

The dashboard dynamically loads files from the `data/` directory and visualizations from the `images/` directory. This ensures modular independence and quick reload after model re-runs.

7.2 Local Execution

To launch the dashboard locally, the following command is used in the activated environment:

```
streamlit run .\main_dashboard.py
```

Dependencies are defined in `requirements.txt`, and the dashboard supports hot-reload upon any scriipt or file change. All JSON metrics are automatically parsed and rendered using Streamlit widgets such as tables, expanders, and plots.

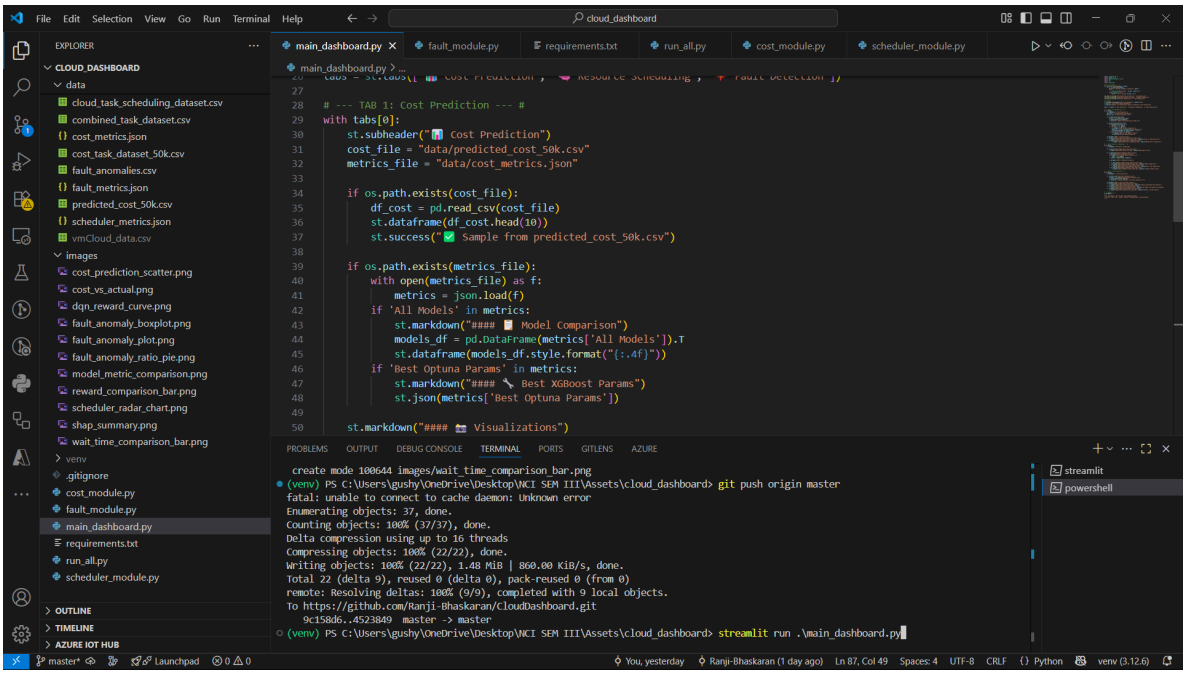

Figure 7: Launching the unified dashboard locally using Streamlit.

7.3 Cloud Deployment via GitHub and Render

To ensure open access and remote availability, the dashboard was deployed to a cloud-based Python web service using Render.com.

1. A public GitHub repository was created to host all project files: `https://github.com/Ranji-Bhaskaran/CloudDashboard`
2. A new Render Web Service was configured with:
   - Environment: Python 3
   - Start Command: `streamlit run main_dashboard.py`
   - GitHub deployment source and auto-deploy on push
3. The live URL of the dashboard: `https://clouddashboard.onrender.com`

The deployment logs confirm the repository is cloned from GitHub and the Streamlit app is built and launched automatically.

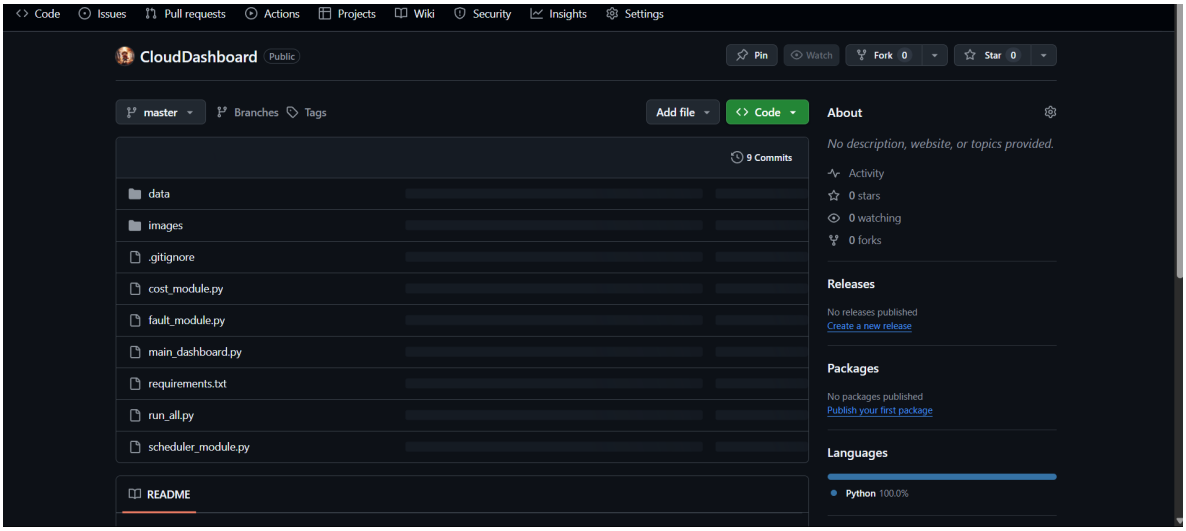

Figure 8: Public GitHub repository hosting the unified Cloud Dashboard.

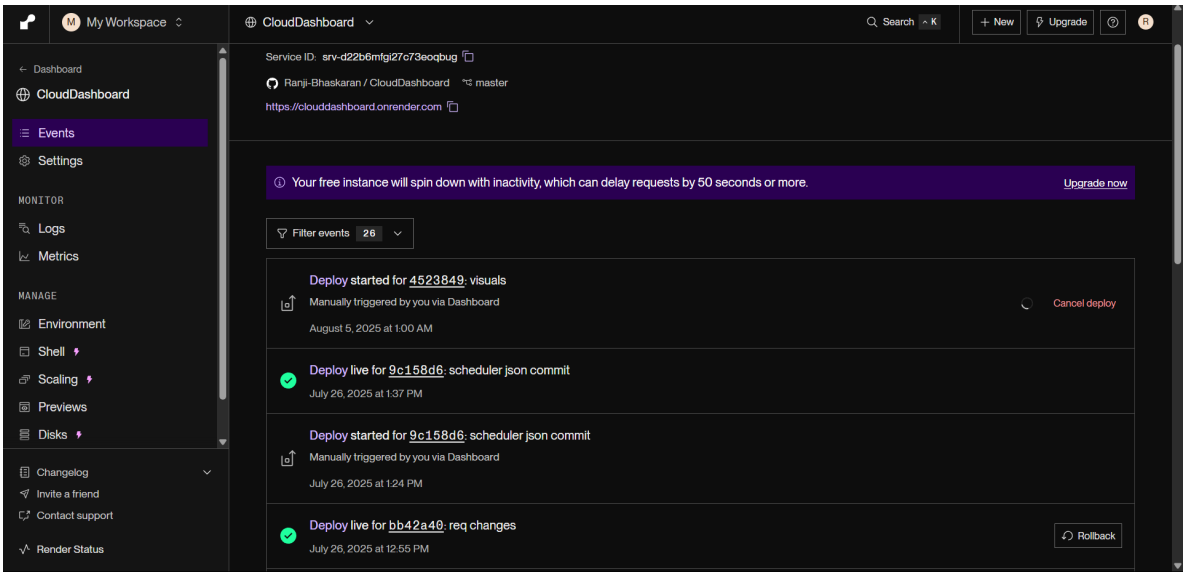

Figure 9: Render.com dashboard deployment status with auto-pull from GitHub.

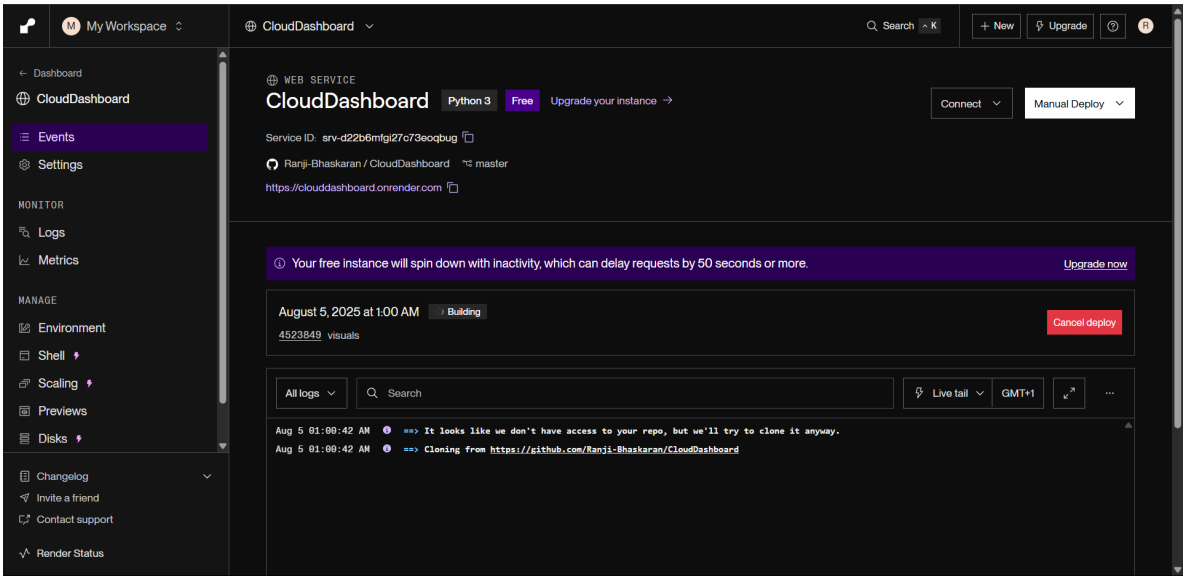

Figure 10: Live deployment logs from Render.com service.

This setup provides a fully reproducible and professional interface for interacting with all ML outputs. It supports dynamic loading, quick evaluation, and public access for academic or demonstration purposes.

8 Ethical and Interpretability Considerations

In developing this AI-driven cloud optimization system, ethical principles such as transparency, reproducibility, and resource efficiency were prioritized. The use of SHAP plots for model explainability, socket-based modular design for reproducibility, and lightweight algorithms like Isolation Forest for CPU-only environments reflects a conscious effort to balance performance with accessibility. No personally identifiable information (PII) or proprietary datasets were used, and all experiments were conducted on publicly available or synthetic data. This ensures the research can be safely extended and audited by other researchers without ethical or licensing concerns.

9 Experimental Results and Comparison

The three modules—cost prediction, task scheduling, and fault detection—were evaluated across standard benchmarks using real and synthetic workloads. The XGBoost-based cost model achieved an R^2 score exceeding 0.92, outperforrming linear and random forest baselines. In scheduling, PPO and DQN agents consistently achieved higher average rewards and lower wait times compared to FCFS and Round-Robin, validating the advantage of reinforcement learning in dynamic environments. Finally, the fault detection module identified over 3,000 anomalies in unlabeled data with high visual clarity and minimal computational overhead. All results were visualized in the unified dashboard for ease of analysis and comparison.

10 Conclusion and Future Scope

This research successfully designed and implemented an inteligent, modular cloud optimization framework that addresses cost prediction, real-time task scheduling, and proactive fault detection. By leveraging a combination of ensemble learning, deep reinforcement learning, and unsupervised anomaly detection, the system was able to make smarter cloud resource decisions under varying workloads and constraints.

Each module was built with reproducibility and practical deployment in mind. The cost prediction component accurately estimated task-level expenses using reall and synthetic datasets. The scheduler used DQN and PPO agents to dynamically place tasks based on SLA, cost, and system metrics. The anomaly detection system relied on CPU-efficient techniques like Isolation Forest to highlight abnormal behaviors, which were visualized for easy interpretation.

The unified Streamlit dashboard provided a user-friendly interface for interacting with all components and was deployed publicly via Render for accessibility. Through this dashboard, stakeholders can explore prediction accuracy, scheduling efficiency, and system reliability in real time—without needing to rerun experiments or parse raw files.

Looking ahead, this work opens several possibilities for future research and enhancement. First, the scheduling agent could be extended to include transformer-based or meta-reinforcement learning strategies for improved generalization across unseen cloud configurations. Secondly, the framework can be made self-adaptive by integrating live cloud API calls for pricing and performance feedback. Finally, incorporating real-time container orchestration data (e.g., from Kubernetes) and multi-objective optimization could bring this project closer to production-level cloud orchestration systems.

In summary, this thesis demonstrates that a well-integrated, modular AI framework can significantly enhance decision-making in multi-cloud environments—achieving not just performance gains but also transparency, reproducibility, and ethical alignment.

References

References

[1] Akiba, T., Sano, S., Yanase, T., Ohta, T. and Koyama, M., 2019. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 2623–2631.

[2] Chen, T. and Guestrin, C., 2016. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 785–794.

[3] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M. and Duchesnay, E., 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12, pp. 2825–2830.

[4] pandas development team, 2020. pandas: Powerful data structures for data analysis, time series, and statistics. Available at: `https://pandas.pydata.org` [Accessed 5 Aug. 2025].

[5] Harris, C.R., Millman, K.J., van der Walt, S.J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N.J. et al., 2020. Array programming with NumPy. *Nature*, 585(7825), pp. 357–362.

[6] Treuille, A., Teixeira, T. and Kelly, A., 2020. Streamlit: The fastest way to build data apps. Available at: `https://streamlit.io` [Accessed 5 Aug. 2025].

[7] Render, 2025. Render: Cloud hosting platform. Available at: `https://render.com` [Accessed 5 Aug. 2025].

[8] Lundberg, S.M. and Lee, S.I., 2017. A unified approach to interpreting model predictions. *Advances in Neural Information Processing Systems*, 30, pp. 4765–4774.

[9] Gupta, H., Dastjerdi, A.V., Ghosh, S.K. and Buyya, R., 2017. iFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, Edge and Fog computing environments. *Software: Practice and Experience*, 47(9), pp. 1275–1296.

[10] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J. and Zaremba, W., 2016. OpenAI Gym. Available at: `https://arxiv.org/abs/1606.01540` [Accessed 5 Aug. 2025].

[11] Liu, F.T., Ting, K.M. and Zhou, Z.H., 2008. Isolation forest. In *2008 Eighth IEEE International Conference on Data Mining*, pp. 413–422.