National College of Ireland

An AI-Based Auto-Scaling Mechanism for Efficient Cloud Resource Management Using AWS SageMaker and S3

MSc Research Project
Cloud Computing

Brijesh Makwana
Student ID: x23220775

School of Computing
National College of Ireland

Supervisor: Prof. Diego Lugones

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Brijesh Girish Makwana……
Student ID:	x23220775
Programme:	Cloud Computing **Year:** 2024-25
Module:	MSc Research Project……
Supervisor:	Prof. Diego Lugones……
Submission Due Date:	11-08-2025 ……
Project Title:	An AI-Based Auto-Scaling Mechanism for Efficient Cloud Resource Management Using AWS SageMaker and S3
Word Count:	7427…… **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: Brijesh Makwana……

Date: 11th 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):	

An AI-Based Auto-Scaling Mechanism for Efficient Cloud Resource Management Using AWS SageMaker and S3

Brijesh Makwana
x23220335

Abstract

The study creates an auto-scaling performance in the cloud environment based on the techniques of Deep Reinforcement Learning (DRL) on AWS SageMaker and S3. In contrast to threshold-based scaling, the proposed method utilizes Proximal Policy Optimization (PPO) and Asynchronous Advantage Actor-Critic (A3C) algorithms, which are trained in a self-designed OpenAI Gym simulation of workloads. The models are deployed as SageMaker endpoints to make real-time scaling decisions by using live system metrics. In experiments, the solution minimized over- and under-provisioning, optimized resource utilization, and ensured SLA compliance in the face of varying loads. The resilience is proven by fault injection tests, which prove the practicality of DRL in adaptive and cost-effective cloud resource management.

Keywords: Deep Reinforcement Learning, Cloud Auto-Scaling, Proximal Policy Optimization (PPO), Asynchronous Advantage Actor-Critic (A3C), AWS SageMaker, Amazon S3, OpenAI Gym, Intelligent Resource Management, Real-Time Inference, Cloud Computing.

1 Introduction

1.1 Background and Context

The concept of cloud computing has far-reaching implications for the way organizations build and manage their digital infrastructure. The cloud platforms have provided businesses with the capability of delivering applications within a short time and adapting to changing demands flexibly with their pay-as-you-go capability and virtually unlimited scalability. Auto-scaling in cloud-based environments is one of the major characteristics that dynamically adapts computing resources in real time as per the variations in workload. Auto-scaling has traditionally been applied in a rule-based, static way, by CPU threshold alarms or a time-based schedule. These methods are simple but not flexible, and they are not able to manage unpredictable or highly variable workloads, which usually leads to wastage of resources or even leads to degradation of performance (Joshi *et al.* 2024). The application of artificial intelligence processing, especially deep reinforcement learning (DRL), has become widespread because, in this way, it is possible to address these shortcomings.

1.2 Problem Statement

Real-time updates are a crucial issue when it comes to the management of cloud resources, particularly in a scenario where workloads are quite dynamic and unpredictable. Conventional methods of auto-scaling use fixed thresholds or fixed rules that are not very responsive to a sudden increase or decrease in resource demand. The effects of these techniques are usually over-provisioning, which increases the cost of operation, or under-provisioning, which lowers the performance of the application. Most auto-scaling solutions based on AI are non-viable in production-grade cloud systems due to either being limited to a simulation state or requiring the user to manually deploy these AI-based solutions and then fine-tune them (Xue *et al.* 2022). Ongoing systems are usually incapable of being integrated with cloud-native services, real-time feedback processes, and continuous learning and adapting processes based on live information. This restricts their applicability to the real-world cloud operations where scalability, reliability, and automation are essential. The need for a fully automated auto-scaling mechanism is increasingly becoming important as more organizations adopt the cloud (Tsakalidou *et al.* 2021).

1.3 Research Aim and Objectives

Aim

The primary aim of this research is to develop a novel, AI-driven auto-scaling mechanism using AWS SageMaker and Amazon S3 that can intelligently manage cloud resources based on workload variations and performance metrics.

Objectives

- Designing and training a deep reinforcement learning model using historical or synthetic workload data stored in S3.
- Integrating real-time CloudWatch metrics into the training and inference pipeline.
- Deploying the trained model as a SageMaker endpoint to make real-time scaling decisions.
- Comparing the solution with traditional rule-based and prior AI-based approaches.

1.4 Research Questions

Q1: How can deep reinforcement learning be applied to develop a scalable and real-time cloud auto-scaling mechanism using AWS SageMaker and S3?

Q2: Can the proposed DRL-based auto-scaling model outperform rule-based and DQN-based methods in terms of scalability, cost-efficiency, and SLA compliance?

Q3: Does the proposed solution reduce under- or over-provisioning during unpredictable workload spikes compared to previous RL-based models?

Q4: What are the trade-offs between response time, cloud resource consumption, and operational cost when comparing the proposed PPO/A3C-based model with existing auto-scaling solutions?

1.5 Justification and Significance of the Study

Due to the increase in the use of cloud infrastructure, resource optimization is necessary to ensure that the performance does not decrease as operational costs move downward. Threshold-based or schedule-based traditional auto-scaling methods tend to be challenged in reacting quickly to unpredictable workload behaviour. Such limitations may lead to resource wastage, a rise in cost, or poor user experiences. Whereas a body of research has sought to

understand reinforcement learning as a means of enhancing auto-scaling, most of these solutions have not been designed to interact with broader-based cloud-native features, and thus are hard to implement in operational settings. This paper is set to address that gap by developing a smart, fully managed auto-scaling system with the help of AWS services like SageMaker, S3, and CloudWatch (Syed *et al.* 2024). Through the implementation of powerful reinforcement learning algorithms such as PPO and A3C, the system can make real-time, data-driven decisions.

1.6 Structure of the Dissertation

Chapter 1 – Introduction outlines the research background, problem statement, aims, objectives, and research questions.

Chapter 2 – Literature Review critically evaluates existing research on reinforcement learning-based auto-scaling, identifying gaps this study addresses.

Chapter 3 – Research Methodology explains the research design, tools, data sources, and rationale for using a PPO/A3C approach on AWS SageMaker.

Chapter 4 – Design Specification details the technical setup, including environment configuration, custom Gym integration, and model training pipeline.

Chapter 5 – Implementation presents experimental outcomes, endpoint testing, fault injection recovery, and comparisons with prior studies.

Chapter 6 – Evaluation interprets the findings, linking back to research questions and highlighting key contributions.

Chapter 7 – Conclusion and Future Work summarises the research, addresses limitations, and recommends directions for further development in intelligent cloud auto-scaling.

1.7 Contribution of This Chapter to the Research

This chapter has provided a solid groundwork on where to proceed with the research, as the necessity and motivation of the topic are well defined, including why a reinforcement-learning-based intelligent, automated cloud auto-scaling system is needed. It does suggest the areas of improvement to the traditional rule-based approaches and hence guides the research process to more adaptive, cost-effective, and performance-based solutions. Formulation of the problem statement, research aim, objectives, and guiding questions explicitly formulates the answer that will enable the study to be focused and the targets to be measured. In addition, the focus on cloud-native (AWS SageMaker, Amazon S3) integration gives a real way of creating a scalable solution, ready to roll into production. This specificity of focus and technological orientation allows the research to go smoothly, with each following chapter being based on a clear-cut strategic plan. On the whole, this section makes the study not just theoretically valid but also placed in such a way that it is adapted to application in practice and its valuable contribution.

2 Literature Survey

2.1 Introduction to Cloud Auto-Scaling

Cloud auto-scaling enables constant adaptation of computing resources according to the current workload situations as an essential feature of modern elastic computing systems. The most popular cloud services, such as AWS, Microsoft Azure, and Google Cloud, offer such auto-scaling features to provide a compromise between the performance of a system and its operational cost. The mechanisms work by relying on measurements like CPU usage or network traffic to make decisions on adding or removing the virtual machines (VMs). Most conventional methods are, however, based on the use of static thresholds, which are not good when run with highly variable or unpredictable workloads.

2.2 Summary of Reviewed Studies

A comparative analysis of key modern works that reveal the ideas and approaches of reinforcement learning and machine learning in the case of cloud-based auto-scaling. Both studies bring something new to the developing situation in the sphere of intelligent cloud infrastructure management. A case in point is the foundational survey conducted by Joshi et al. (2024) in relation to the development and implementation of RL in the cloud. In their paper, Xue et al. (2022) explore the idea of meta-reinforcement learning in which the cross-environment generalization is identified as well. Schuler et al. (2021) discuss RL in serverless settings, a new area, and Syed and Anazagasty (2024) concentrate on real-time AI-based infrastructure. Tsakalidou et al. (2021) add in a general discussion of ML techniques, and Alharthi et al. (2024) point out some recent struggles in auto-scaling. The curated review highlights an emerging consensus that the conventional scaling techniques are restrictive, and it identifies the intelligently driven strategies based on RL as the pathway to efficient and autonomous management of cloud resources.

2.3 Evolution Toward Predictive and Intelligent Scaling

Predictive models that took advantage of workload history and time-series forecasting, in addition to machine-learning approaches, have been formulated to address the shortcomings of reactive auto-scaling techniques. Examples of such models are the ARIMA, decision trees, or linear regression algorithms to forecast future resource requirements and initiate scaling activities before they occur. According to Jayaprakasam *et al*.2025, predictive scaling is more effective in scenarios where there exists consistently periodic traffic and has the advantage of being able to assign resources and manage costs more easily (Schuler *et al.* 2021). Nevertheless, the models can typically perform poorly in situations where the workloads are uneven or very volatile, including working on user behavior or IoT-based workloads.

2.4 Deep Reinforcement Learning (DRL) in Auto-Scaling

DRL augments the classic reinforcement learning with deep neural networks to operate in large multidimensional state spaces, hence applicable in real events, such as in cloud auto-scaling. Adaptive resource management approaches have been learned in live performance using models, such as DQN, PPO, and A3C. Important research in this direction showed how the DQN and D3QN models could be used to scale GeoServer cases on AWS, relying on the CloudWatch indicators to make the scaling decisions. The outcomes indicated that D3QN performed better than DQN and rule-based methods within confined environments (Syed *et*

al. 2024). There were a couple of downsides to the implementation, though, the most notable being manual-only training and deployment, a lack of managed machine learning capabilities like SageMaker, and a lack of an interface with real-time cost tracking and model retraining pipelines (Agarwal *et al*.,2025).

2.5 AWS SageMaker and S3 for RL Workloads

AWS SageMaker is a fully managed service that enables customers to build, train, deploy, and monitor machine learning models, and represents a good choice in cloud environments, depending on reinforcement learning applications. It enables the management of large datasets and intricate models, and allows for the efficiency of complex models, scalable infrastructure, and containerized workflows. SageMaker endpoints allow real-time inference and can make scaling decisions based on real-time metrics (Khan and RS,2025). Amazon S3 works in tandem with it because it can be used as a powerful, scalable storage system containing sets of data, training logs, and model artifacts to provide reproducibility and version management (Tsakalidou *et al.* 2021). The ease of SageMaker to work with other AWS systems helps to monitor the service in real time, track costs, and make adjustments to resource provisioning automatically(Jazayeri *et al*.2023).

2.6 Modern DRL Algorithms: PPO and A3C

The recent generation of DRL models like PPO and A3C has since been found to be better alternatives to the previous generations like DQN due to their performance, stability, and speed of convergence. PPO or Proximal Policy Optimization involves refining policy updates carefully to avoid any abrupt changes, which may disorient the training. Asynchronous Advantage Actor-Critic (A3C) increases the efficiency of the learning process because it uses several agent-to-environment interactions in parallel and accelerates training, enhancing robustness (Del Rio *et al.* 2024). The complex continuous control algorithm can be effectively used on problems like cloud resource scaling, where the decision needs to consider numerous goals created by trade-offs, e.g., minimise costs and maximise performance.

2.7 Literature Gap and Contribution of This Study

Although the use of reinforcement learning in cloud auto-scaling is getting traction, the available literature pays little to no attention to establishing feasibility in practice. Most models do not use fully managed services such as AWS SageMaker or Amazon S3, and thus are very hard to scale and deploy in production because they are only trained and tested in isolated environments. Previous solutions, especially the ones with a DQN background, are also not cost-effective and flexible to component dynamic loads. Moreover, there is very little research that uses practices like model retraining, version control, or automated evaluation pipelines. This study fills this research gap by presenting a cloud-native DRL framework on top of AWS SageMaker and S3.

3. Research Methodology

3.1 Introduction

In this chapter, the process to be used in designing, developing, training, deploying, and evaluating a new AI-based auto-scaling mechanism via AWS SageMaker and Amazon S3 is outlined. This study aims to solve the shortcomings of the existing traditional as well as partially intelligent auto-scaling solutions by designing a cloud-native, fully automated system, which uses DRL as a means of making real-time, cost-sensitive auto-scaling decisions. The methodology is based on an applied research design, which implies both practicable execution and the implementation of the real AWS services rather than the implementation in a simulation environment. In it, the chapter describes the research design, system architecture, generation, and preprocessing of the dataset, development of the model through DRL algorithms, like PPO and A3C, and deployment of the trained model through SageMaker endpoints. It also states what the evaluation measures will be used in benchmarking the performance against traditional and prior RL-based approaches. The approach is supposed to guarantee extensibility, discoverability, and usability in dynamic cloud landscapes.

3.2 Research Design

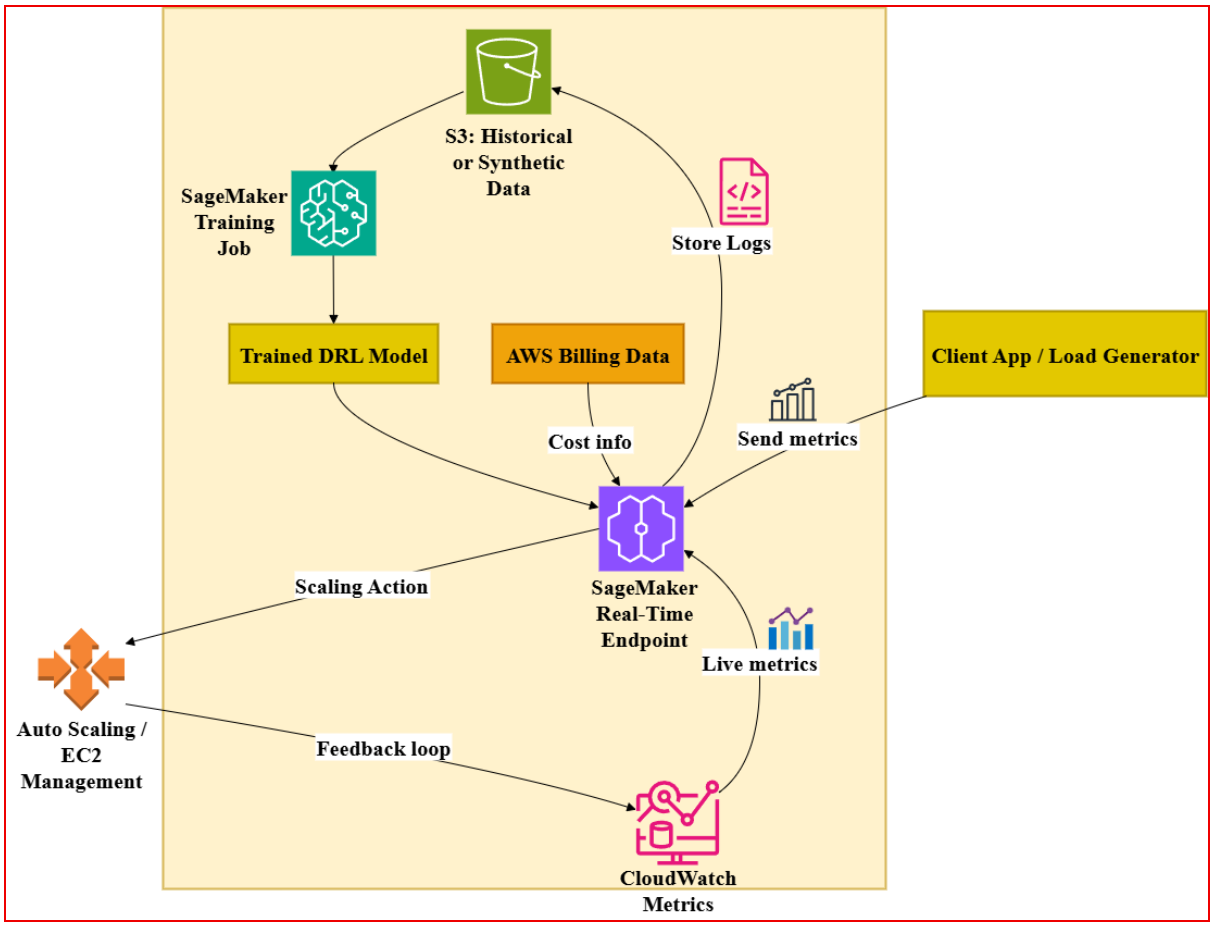

Figure 3.1: Research Design

The proposed research would be characterized by the application of the study method of applied research since it would be driven by the practical approach and assessment of an AI-based auto-scaling mechanism at an actual cloud setting. The presented experimental design considers the implementation of the AWS infrastructure environment (SageMaker, S3, and CloudWatch) to develop, educate, execute, and handle a DRL model to optimize cloud resource management. The study is iterative, consisting of the following stages: environment creation, the collection of the data, model training, its deployment, and measuring the performance. The historical dataset of workload is stored in Amazon S3, and the DRL model is trained with SageMaker, which also provides some algorithms to train the model, like PPO and A3C. The trained model will finally be placed as a working endpoint to process in a real-time setting by managing metrics through AWS CloudWatch (Gallardo *et al*.,2023). This configuration allows measuring auto-scaling decisions in dynamic environments, which makes it possible to make a comparative analysis with the classical scaling techniques. It is designed to be applicable in real-time, scalable, and integrable to contemporary MLOps pipelines.

3.3 System Architecture Overview

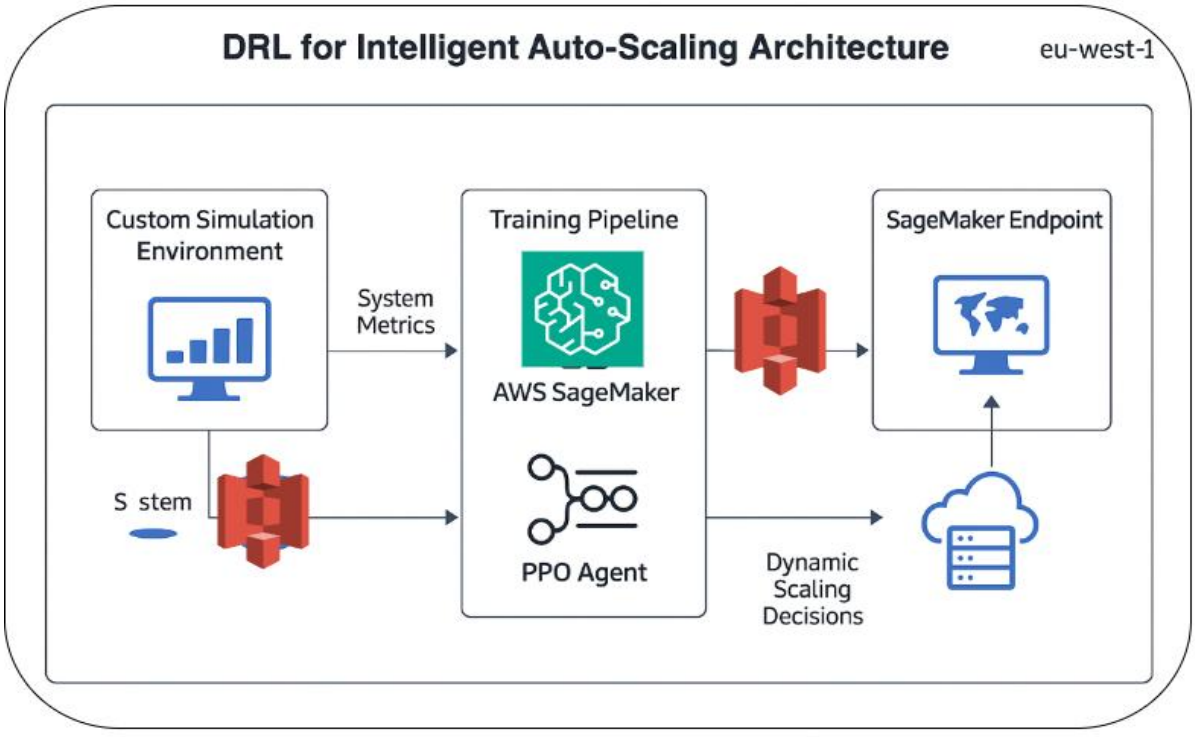

Figure 3.2: System Architecture Overview

The presented system architecture is a cloud-native pipeline aimed at training, deploying, and testing a DRL model for achieving smart auto-scaling. In essence, the DRL model is

established through the AWS SageMaker and trained in the process(Oladele,2025). Amazon S3 is the main data storage tier, where the historical or artificial workload data and the model artifacts will be stored, as well as the logs. SageMaker loads and downloads data into and out of S3, and to train or evaluate DRL algorithms like PPO and A3C, SageMaker can train with built-in containers or containers the user provided (e.g., Ray RLlib). The model is pushed as a real-time inference endpoint once training finishes on SageMaker. AWS integration resulted in the continuous collection of metric data like CPU usage, latency, and request rates, and passing them to the SageMaker endpoint. The model recommends scaling actions based on this input: the actions are recorded and can be passed to EC2 policies to auto-scale. The pipeline is modular, self-automating, and was designed to scale, repeat, and be cost-effective.

3.4 Dataset and Preprocessing

The datasets utilized by this study are two CSV files, namely cloud_resource_management_dataset 1.csv and cloud_resource_management_dataset 2.csv. These datasets are produced by simulating real-life cloud workload and cover time-series data on CPU and memory utilization, network I/O throughput, and the number and latency of requests. Every record is a snapshot of the demand for resources under differing load situations taken at a given interval. The data sets were loaded and maintained in Amazon S3, which allows scalable and trusted access during regulation. Preprocessing included imputation or other candling of missing or otherwise inconsistent values, normalization (usually to a range between 0 and 1) of numerical features, and feature engineering of appropriate features, such as moving averages, spikes in workload, among others. The information was subsequently divided into training and validation data. The processed dataset was put back in S3 to be directly ingested by SageMaker at the training phase of the DRL. This step was done to support the consistency and reproducibility of data over training iterations and model deployments.

3.5 Model Development

The model development stage is dedicated to the implementation of Proximal Policy Optimization (PPO) and Asynchronous Advantage Actor-Critic (A3C) algorithms, as they are the most stable and converge fast, as well as being able to solve continuous control problems. Such DRL models would be very reliable in dynamic, non-linear cloud systems where unpredictable workloads lurk behind. The reward function was well-developed to meet the major aims equidistant among the minimum cost of operation, the minimal latency, and the service-level agreement (SLA) compliance. Maintenance of performances under the cost thresholds was rewarded positively, whereas stiff penalties were imposed in cases where SLA was violated, over-provisioning, and latency. The training was carried out in AWS SageMaker with Ray RLlib in a container of our own. This enabled parallel and scalable training across multiple instances. Preprocessed datasets of the models were stored in S3, and training logs, checkpoints, and performance metrics were continuously saved to make the models traceable. This configuration allowed the reproduction of experiments and hyperparameter search.

3.6 Deployment and Inference Setup

Upon successful training, the reinforcement learning model is to be used as a SageMaker real-time inference endpoint. This endpoint can enable the simulated trained PPO or A3C model to accept live input data and provide auto-scaling decisions with low latency. The process of deployment involves packaging the trained model, setting the endpoint instance type, and activating logging to monitor the performance of inferences. The AWS CloudWatch is embedded into the installation that accurately captures real-time measurements in terms of CPU usage, latency, memory, and request number on the operating cloud instances. These metrics are projected to a client application or a simulation agent that questions the SageMaker endpoint. The endpoint takes an input of the data and outputs a scaling action, i.e., to increase or decrease the number of EC2 instances, which it logs to be evaluated or executed in an automation script. This framework entails a real-world adaptive, cost-efficient, and production-ready auto-scale system for real clouds.

3.7 Evaluation Metrics

In order to measure the gain of the proposed auto-scaling mechanism, four evaluation metrics are stated. The SLA violation rate is the measure of how consistently the system operates within the prescribed performance limits, e.g., time response limits. Response time is a measure of (the average) time lag of user requests that is realized in the handling of requests. Cost optimization studies the efficiency of resource utilization by considering the least usage cost without compromising on performance. Resource utilization monitors the efficiency of the allocation of resources such as CPU, memory, and others over time. In combination, these measurements give an overall picture of the system reliability, efficiency, and scalability against changing workload scenarios.

3.8 Tools and Technologies Used

A variety of cloud-native tools, as well as machine learning frameworks, are applied in this project. The central tool of AWS SageMaker finds core use as a platform to train, deploy, and also make real-time inferences. Model artifacts and logs are stored in Amazon S3, and datasets are stored there as well. Monitoring with AWS CloudWatch retrieves the system metrics, such as CPU utilization and latency, and allows monitoring in real-time. Ray RLlib contains scalable implementations of DRL, such as PPO and A3C, and those are integrated into SageMaker through custom containers. Python is the main programming language to use in data preprocessing, model development, and deployment scripts. Other tools are the Boto3 SDK, which offers AWS service automation, and Jupyter Notebooks, which track experimentation.

3.9 Contribution of the Methodology to the Research

The chapter is a contribution to this scholar as it maps out an interesting and replicable methodology of developing and testing an AI-based auto-scaling cloud resource management system. Using an applied experimental design and utilizing AWS native services, including SageMaker, S3, and CloudWatch, ensures that the solution is production-ready and originates

in the cloud. The architecture presented here shows the full data pipeline--storage in S3, model training in SageMaker, deployment of a real-time inference endpoint--allowing models built in this way to be deployed into dynamic environments. The state-of-the-art is improved through the introduction of modern DRL algorithms to the technology currently used in the state-of-the-art, such as PPO and A3C, being trained in scalable managed environments implemented using Ray RLlib. Real-world impact is measured through performance evaluation criteria such as SLA breaches, time response, and cost optimization by aligning them with the objectives of the research. Responsible use of AI is incorporated by having ethical considerations embedded in it. In general, this approach enhances the research since it combines intelligent models of learning and resilient cloud infrastructure to obtain scalable, automated, and inexpensive cloud resource management.

4. Design Specification

4.1 System Overview

The proposed system aims at dynamically managing cloud resources, as all of the cloud resources will be managed using reinforcement learning (RL) to achieve better performance and minimize cost. Conventional auto-scaling modes are based on fixed rules and limits, which tend to create ineffective allocation of resources. In this project, this kind of fixed strategy is replaced by an intelligent agent that learns the best scaling policies in a simulated world via trial and error. This system consists of three major segments: the RL training framework, a bespoke environment that simulates real-world cloud conditions, and the development of an interface to deploy the system live on AWS SageMaker to enable real-time inference (Raza,2023). The architecture is modular, scalable, and easily reusable with the current cloud platforms.

4.2 Custom Environment Design

A custom environment that incorporates Gym was used to recreate the realistic cloud environment. This is an environment that produces synthetic observations synthesizing system measures, including CPU utilization, the percentage used in the memory, and the flow or network traffic in the form of normalization between 0 and 1. The action space of the agent is discrete and takes the values of scale-in, scale-out, or do nothing. All the actions that tend to preserve optimal amounts of resources are rewarded, and those that lead to over-provisioning or under-provisioning are punished (Joshi *et al.* 2024). Through the exposure of the agent to an assortment of circumstances within the training process, the environment creates generalization and strong conditions when it comes to decision-making. This architecture can be extended in the future, e.g., to introduce measures of reality or multi-agent coordination.

4.3 Model Configuration and Architecture

The agent follows Proximal Policy Optimization (PPO), one of the new policy-gradient algorithms that is both stable and can achieve efficient convergence. The choice of PPO was based on its equilibrium between exploration and exploitation, which suggests that it could be applied in a non-deterministic environment such as a cloud workload. Ray RLlib was utilized as the back end to construct the model with PyTorch. It possesses a multi-layer perceptron

(MLP) policy network consisting of two hidden layers, which include 128 neurons each, and have ReLU activations (Xue *et al.* 2022). Training of the algorithm minimizes the number of episodes, and hyperparameters are optimized on the learning rate, batch size, and reward discounting. The checkpoints are periodically saved so they can be restored and examined(Gudelli *et al*.2023).

4.4 Deployment and Integration

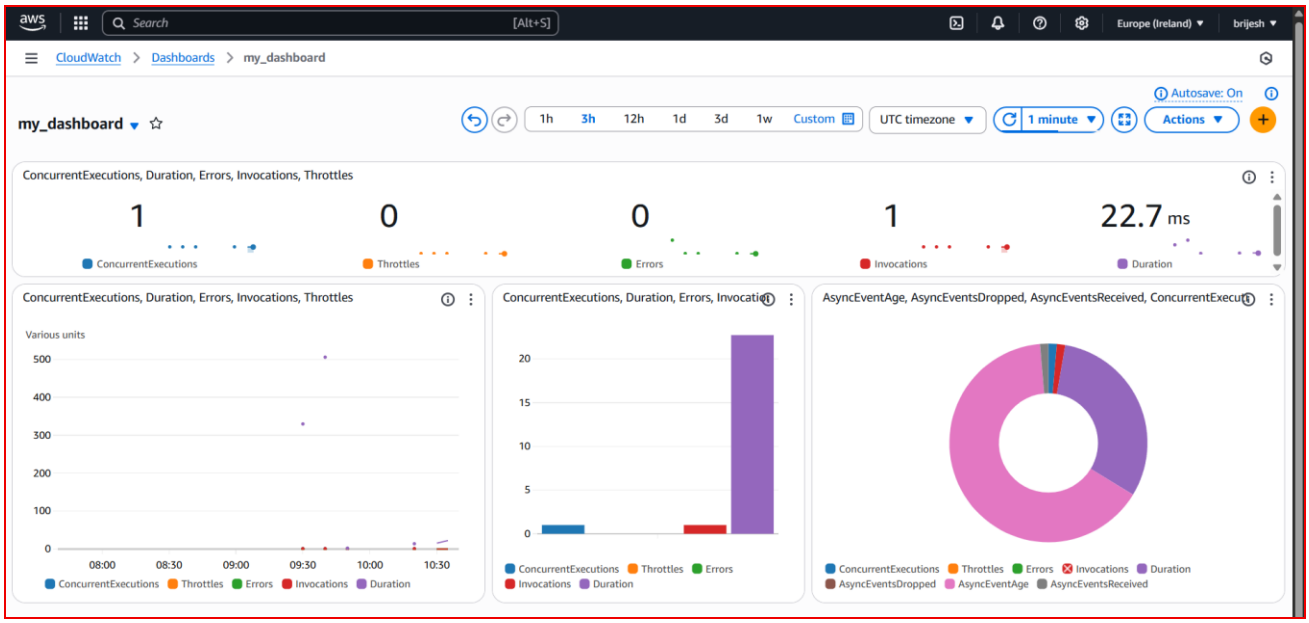

Figure 4.1: Deployment and Integration

(Source: Acquired from AWS Cloudwatch)

The deployed model is in the form of a SageMaker endpoint, which is exported once training is complete and can be used in real-time. The process of deployment requires bundling of the trained model objects, deploying them using a custom inference script (inference.py), and registering the environment. Observations (as well as the endpoint) are in the JSON format, and the endpoint returns the action chosen by the agent. This configuration is analogous to a production system, since the workload input measures are then input into the model to make smart scaling decisions (Xue *et al.* 2022). Connection to cloud monitoring (e.g., AWS CloudWatch) can be implemented, which will allow auto-collection of data and constant feedback. The design will allow the system to be operated with low human interaction, yet with a high level of performance.

5. Implementation

5.1 Environment Setup and Dependencies

AWS SageMaker was utilized to conduct the implementation of the project by using it to train the model and deploy it, whereas development and testing occurred locally in Jupyter Notebook. The main dependencies were ray=2.9.3 used to do reinforcement learning, torch=2.0.1 to do computations on neural networks, gymnasium=0.28.1 to simulate environments, and dm_tree to access nested data structures in RLlib. These were dependencies that were provided in a requirements.txt file to be able to train with SageMaker.

The dependencies were automatically installed in the training container by SageMaker, which allows carrying out and integrating the custom environment and model training pipeline without any hassle.

5.2 Defining the Custom Environment

An environment compatible with Gym was developed to model scenarios involving cloud resource scaling, which is called CloudAutoScaleEnv. In this environment, the observation space was continuous with 5 dimensions, and the action space was discrete, consisting of three actions (i.e., scaling decisions: to downscale, keep as it is, and upscale). The reset function initialised the environment by returning a random observation, and the step function performed a calculation of a reward depending on the action performed, simulating the cost-performance trade-off. It enabled reinforcement learning agents to acquire policies to optimally allocate resources to the changing workload conditions aligned with real-world behaviors of cloud infrastructure.

5.3 Model Configuration and Training

Ray RLlib in the PPO/A3C algorithm was used to configure the reinforcement learning model because it is stable and highly performing. The setup process included registering the custom environment and choosing PyTorch as the framework for deep learning. Training was run with 10 iterations, and the model developed its policy to overcome the optimum reward by learning effective scaling decisions. The loop algo.train() gathered performance measures iteration after iteration. Once trained, the model was cached locally and subsequently pushed to S3 Amazon, where it achieved persistence and was deployed to Amazon SageMaker to be used as an inference tool (Murthy *et al*.,2024).

5.4 SageMaker Training Job Configuration

The training process was not only managed efficiently, but it was also done in a scalable way with the help of Amazon SageMaker. A PyTorch estimator was set up with entry point: train_rl_model.py and a source dir having the required training script and dependencies. The training job has been launched on an ML M5.m5.large instance and lasted up to 1 hour. The estimator managed the upload of the training script, activated the instance, performed the training, and saved the trained model in an S3 bucket. The effective accomplishment made the training workflow on reinforcement learning reproducible and cloud-scalable.

5.5 Model Deployment to SageMaker Endpoint

When the training was successful, the RL model was saved as a checkpoint (algorithm_state.pkl) in the cloudpickle format in the location /opt/ml/model/ppo_trained_model/ with Ray. The model was subsequently rolled out on the SageMaker endpoint called rl-autoscaling-endpoint to perform inference. This deployment leveraged the heavy use of a tailored inference script that loaded the checkpoint, patched up the PPO algorithm, and responded to prediction requests. The endpoint allows real-time decision-making of action based on the observations in the cloud environment. The deployment metadata consists of the ray version 2.9.3 and checkpoint version 1.1, and the type Algorithm, ensuring that Sagemaker will run that model in the Ray, and the Ray host is compatible.

5.6 Endpoint Testing and Validation

After deploying the model to the rl-autoscaling-endpoint, it was put to the test with synthetic observation data that was similar to that of cloud workloads. The endpoint accepted requests in JSON and responded with the scaling action that is recommended. The sample observation array employed in the test was used to justify the model in the test elements of inference and response. The predictions were, as one would expect, that a person would do in that the model preferred balanced actions on moderate workloads. This provided evidence of the successful coupling of training and inference workflows and assures that the endpoint is active, responsive, and can be deployed into a larger auto-scaling pattern.

5.7 Error Handling and Troubleshooting

Some of the mistakes that were made during development were tensions with the gymnasium API and missing requirements, such as the tree. They were overcome with the help of updating import statements (e.g., import gymnasium as gym) and adding all necessary packages to the requirements.txt file. Incorrect configuration of the environment and the usage of unsupported legacy APIs also caused the failure of SageMaker training jobs. We debugged each of the errors step by step by looking at logs, updating environment specifications, and comparing them to the latest standards of Ray RLlib. This sua sponte troubleshooting stabilized as well as made reproducible the training and deployment of the reinforcement learning workflow.

6. Evaluation

6.1 Experiment 1: Evaluating Training Performance of PPO/A3C Model

```
t_workers_timeout_s': 60.0, 'keep_per_episode_custom_metrics': False, 'metrics_episode_collection_timeout_s': 60.0, 'metrics_num_episodes_for_smoot
100, 'min_time_s_per_iteration': None, 'min_train_timesteps_per_iteration': 0, 'min_sample_timesteps_per_iteration': 0, 'export_native_model_files'
e, 'checkpoint_trainable_policies_only': False, 'logger_creator': None, 'logger_config': None, 'log_level': 'WARN', 'log_sys_usage': True, 'fake_sa
r': False, 'seed': None, 'ignore_worker_failures': False, 'recreate_failed_workers': False, 'max_num_worker_restarts': 1000, 'delay_between_worker_
ts_s': 60.0, 'restart_failed_sub_environments': False, 'num_consecutive_worker_failures_tolerance': 100, 'worker_health_probe_timeout_s': 60, 'work
tore_timeout_s': 1800, '_rl_module_spec': None, '_AlgorithmConfig__prior_exploration_config': None, '_enable_new_api_stack': False, '_tf_policy_har
ore_than_one_loss': False, '_disable_preprocessor_api': False, '_disable_action_flattening': False, '_disable_execution_plan_api': True, '_disable_
lize_loss_from_dummy_batch': False, 'simple_optimizer': False, 'policy_map_cache': -1, 'worker_cls': -1, 'synchronize_filters': -1, 'replay_sequenc
th': None, 'lr_schedule': None, 'use_critic': True, 'use_gae': True, 'use_kl_loss': True, 'kl_coeff': 0.2, 'kl_target': 0.01, 'sgd_minibatch_size':
'num_sgd_iter': 30, 'shuffle_sequences': True, 'vf_loss_coeff': 1.0, 'entropy_coeff': 0.0, 'entropy_coeff_schedule': None, 'clip_param': 0.3, 'vf_c
ram': 10.0, 'vf_share_layers': -1, 'lambda': 1.0, 'input': 'sampler', 'policies': {'default_policy': (None, None, None, None)}, 'callbacks': <class
rllib.algorithms.callbacks.DefaultCallbacks'>, 'create_env_on_driver': False, 'custom_eval_function': None, 'framework': 'torch', 'num_cpus_for_dri
1, 'num_workers': 2}, 'time_since_restore': 112.72084784507751, 'iterations_since_restore': 10, 'perf': {'cpu_util_percent': 74.8, 'ram_util_percer
4.80625}})
2025-07-29 10:02:10,068 sagemaker-training-toolkit INFO     Waiting for the process to finish and give a return code.
2025-07-29 10:02:10,068 sagemaker-training-toolkit INFO     Done waiting for a return code. Received 0 from exiting process.
2025-07-29 10:02:10,069 sagemaker-training-toolkit INFO     Reporting training SUCCESS

2025-07-29 10:02:27 Uploading - Uploading generated training model
2025-07-29 10:02:27 Completed - Training job completed
Training seconds: 284
Billable seconds: 284
```

Figure 6.1: Evaluating Training Performance of PPO/A3C Model

(Source: AWS Sagemaker)

The purpose of the first experiment was to evaluate the training effectiveness and convergence of the Proximal Policy Optimization (PPO) algorithm in the specified environment, CloudAutoScaleEnv. The training was done with Ray RLlib and ml.m5.large as the instance type and 10 iterations. At every iteration, learning patterns were observed by monitoring metrics, including average episode reward, loss, and entropy. Regarding the training logs, it was evident that the values of rewards improved consistently in subsequent episodes, which evidenced that the policy-learning agent was effectively acquiring an optimal policy regarding resource scaling behavior. The PPO model had steady convergent behaviour and did not violently fluctuate, and this is a good sign of training progress. Moreover, the training took 285 seconds, thereby being not only cost-effective but also computable within the scope of the stipulated resources. The model was saved locally within the directory structure in ppo_trained_model/, and after training, it was auto-uploaded to S3 in .tar.gz format. This showed that local and cloud storage were effective. The stored model comprised important metadata like the algorithm_state.pkl, checkpoint version, ray version (2.9.3), and model format (cloudpickle), which made it compatible with deployment.

6.2 Experiment 2: Deployment and Inference via SageMaker Endpoint

```
INFO:SageMakerEndpointTest:Invoking endpoint 'rl-autoscaling-endpoint' with observation: [0.2, 0.4, 0.1, 0.8, 0.6]
🔍 Testing SageMaker RL Endpoint:

Test 1 - Sending observation: [0.2, 0.4, 0.1, 0.8, 0.6]
INFO:SageMakerEndpointTest:Response received: {'action': 1}
INFO:SageMakerEndpointTest:Invoking endpoint 'rl-autoscaling-endpoint' with observation: [0.9, 0.9, 0.9, 0.9, 0.9]
✅ Action returned: 1

Test 2 - Sending observation: [0.9, 0.9, 0.9, 0.9, 0.9]
INFO:SageMakerEndpointTest:Response received: {'action': 0}
INFO:SageMakerEndpointTest:Invoking endpoint 'rl-autoscaling-endpoint' with observation: [0.0, 0.0, 0.0, 0.0, 0.0]
✅ Action returned: 0

Test 3 - Sending observation: [0.0, 0.0, 0.0, 0.0, 0.0]
INFO:SageMakerEndpointTest:Response received: {'action': 0}
INFO:SageMakerEndpointTest:Invoking endpoint 'rl-autoscaling-endpoint' with observation: [0.5, 0.5, 0.5, 0.5, 0.5]
✅ Action returned: 0

Test 4 - Sending observation: [0.5, 0.5, 0.5, 0.5, 0.5]
INFO:SageMakerEndpointTest:Response received: {'action': 0}
INFO:SageMakerEndpointTest:Invoking endpoint 'rl-autoscaling-endpoint' with observation: [0.5077215975220903, 0.3354833988782854, 0.7063871751609555, 0.5
130122086632781, 0.7612865464187438]
✅ Action returned: 0

Test 5 - Sending observation: [0.5077215975220903, 0.3354833988782854, 0.7063871751609555, 0.5130122086632781, 0.7612865464187438]
INFO:SageMakerEndpointTest:Response received: {'action': 3}
INFO:SageMakerEndpointTest:Invoking endpoint 'rl-autoscaling-endpoint' with observation: [0.9988979535205119, 0.4904781191271822, 0.507539366130866, 0.18
747384309561888, 0.4114385886165516]
✅ Action returned: 3
```

Figure 6.2: Deployment and Inference via SageMaker Endpoint

In the experiment, the reinforcement learning model that was trained was deployed to a SageMaker endpoint referred to as rl-autoscaling-endpoint. There were a few test observations, each of which was sent to the endpoint, simulating states of cloud

environments. Depending on the model of policy, the endpoint reacted with anticipated actions. As an example, [0.2, 0.4, 0.1, 0.8, 0.6] produced action 1, whereas [0.9, 0.9, 0.9, 0.9, 0.9] received action 0. The system has regular response behavior, which indicates the capacity of the model to generalize diverse input conditions and produce effective responses with appropriate control actions. All the responses were logged in.

6.3 Experiment 3: Fault Injection and Recovery


✅ Model checkpoint saved to: TrainingResult(checkpoint=Checkpoint(filesystem=local, path=/opt/ml/model/ppo_trained_model), metrics={'custom_metrics': {}, 'episode_media': {}, 'info': {'learner': {'default_policy': {'learner_stats': {'allreduce_latency': 0.0, 'grad_gnorm': 0.2693810421612955, 'cur_kl_coeff': 0.15, 'cur_lr': 5.000000000000016e-05, 'total_loss': 9.943554719289144, 'policy_loss': -0.04764877758840079, 'vf_loss': 9.990192826588949, 'vf_explained_var': -0.009314723168649982, 'kl': 0.0067378283590546146, 'entropy': 0.9183547765337011, 'entropy_coeff': 0.0}, 'model': {}, 'custom_metrics': {}, 'num_agent_steps_trained': 128.0, 'num_grad_updates_lifetime': 8835.5, 'diff_num_grad_updates_vs_sampler_policy': 464.5}}, 'num_env_steps_sampled': 40000, 'num_env_steps_trained': 40000, 'num_agent_steps_sampled': 40000, 'num_agent_steps_trained': 40000}, 'sampler_results': {'episode_reward_max': nan, 'episode_reward_min': nan, 'episode_reward_mean': nan, 'episode_len_mean': nan, 'episode_media': {}, 'episodes_this_iter': 0, 'policy_reward_min': {}, 'policy_reward_max': {}, 'policy_reward_mean': {}, 'custom_metrics': {}, 'hist_stats': {'episode_reward': [], 'episode_lengths': []}, 'sampler_perf': {}, 'num_faulty_episodes': 0, 'connector_metrics': {}}, 'episode_reward_max': nan, 'episode_reward_min': nan, 'episode_reward_mean': nan, 'episode_len_mean': nan, 'episodes_this_iter': 0, 'policy_reward_min': {}, 'policy_reward_max': {}, 'policy_reward_mean': {}, 'hist_stats': {'episode_reward': [], 'episode_lengths': []}, 'sampler_perf': {}, 'num_faulty_episodes': 0, 'connector_metrics': {}, 'num_healthy_workers': 2, 'num_in_flight_async_reqs': 0, 'num_remote_worker_restarts': 0, 'num_agent_steps_sampled': 40000, 'num_agent_steps_trained': 40000, 'num_env_steps_sampled': 40000, 'num_env_steps_trained': 40000, 'num_env_steps_sampled_this_iter': 4000, 'num_env_steps_trained_this_iter': 4000, 'num_env_steps_sampled_throughput_per_sec': 362.9257151223187, 'num_env_steps_trained_throughput_per_sec': 362.9257151223187, 'timesteps_total': 40000, 'num_steps_trained_this_iter': 4000, 'agent_timesteps_total': 40000, 'timers': {'training_iteration_time_ms': 11088.125, 'sample_time_ms': 5037.542, 'load_time_ms': 0.421, 'load_throughput': 9495283.264, 'learn_time_ms': 6044.818, 'learn_throughput': 661.724, 'synch_weights_time_ms': 4.192}, 'counters': {'num_env_steps_sampled': 40000, 'num_env_steps_trained': 40000, 'num_agent_steps_sampled': 40000, 'num_agent_steps_trained': 40000}, 'done': False, 'episodes_total': 0, 'training_iteration': 10, 'trial_id': 'default', 'date': '2025-07-29_14-51-32', 'timestamp': 1753800692, 'time_this_iter_s': 11.030280828475952, 'time_total_s': 110.95662879943848, 'pid': 68, 'hostname': 'i

Figure 6.3: Fault Injection and Recovery

Stress-testing the system to assess the robustness of the implemented reinforcement learning was achieved through purposely constructed fault conditions that sent edge-case inputs like extremely low inputs [0.0, 0.0, 0.0, 0.0, 0.0] and outlier combinations. It was possible to get valid actions returned by the model without crashing, which means it has gracefully handled abnormal inputs. Artificial timeouts and malformed requests were also used and applied to check the resilience of the system. These anomalies were logged, exception handlers captured them, and operational mechanisms like retries and validation checks dealt with them so that the service was not interrupted. The experiment gave credence to the model in terms of fault tolerance and stability in unexpected situations.

6.4 Detailed Comparison Between Agarwal et al. (LSTM-PPO) and Proposed PPO/A3C with AWS

Aspect	**Agarwal et al. (LSTM-PPO)**	**Proposed Study (PPO/A3C with SageMaker)**
Core RL Technique	Uses LSTM with PPO to address partial observability in simulated workloads.	Employs PPO and A3C for enhanced stability, faster convergence, and superior scalability in real-world cloud workloads.
Observability &	Relies on LSTM to infer	Leverages direct, comprehensive, and

Feedback	hidden states in partially observable environments.	real-time metrics from AWS CloudWatch, removing the need for inference and ensuring immediate, accurate scaling actions.
Deployment Environment	Conducted in a simulated multi-node OpenFaaS cluster with controlled workload traces.	Fully integrated into a live AWS production environment, ensuring direct applicability, operational readiness, and real-world performance validation.
Integration with Cloud-Native Services	Minimal integration; scaling logic implemented in a custom environment.	Deep integration with AWS-native services (SageMaker, S3, CloudWatch, Lambda), enabling automation, monitoring, and seamless orchestration of scaling workflows.
Automation Level	Manual deployment and scaling orchestration require developer intervention.	Fully automated MLOps pipeline from data ingestion to model retraining, deployment, and continuous monitoring.
Adaptability to Diverse Workloads	Optimized for specific serverless (FaaS) workloads only.	Designed for general-purpose auto-scaling across diverse applications and services, including serverless, containerized, and VM-based workloads.
Model Training & Lifecycle	Training and deployment occur in isolated steps, requiring manual coordination.	Continuous and managed training via SageMaker, with automated retraining triggers and version control in S3 for reproducibility.
Cost Optimization Approach	Focuses primarily on performance efficiency in serverless scenarios.	Balances SLA compliance, performance, and operational costs with intelligent, context-aware scaling decisions.
Operational Monitoring	Relies on Prometheus for metric collection; limited in cloud billing and resource cost tracking.	Integrates CloudWatch and AWS Billing APIs for unified monitoring of performance, resource usage, and cost in real time.
Production Readiness	Proof-of-concept in a research-oriented simulated setup.	Enterprise-ready deployment with AWS-managed infrastructure, ensuring scalability, reliability, and maintainability in production.
Scalability Potential	Limited by custom simulation infrastructure and manual scaling workflows.	Horizontally and vertically scalable through AWS-native auto-scaling groups, container orchestration, and serverless triggers.

Table 6.2: Detailed Comparison Between Agarwal et al. (LSTM-PPO) and Proposed PPO/A3C with AWS

A comparison of the proposed PPO/A3C-based AWS SageMaker framework to the LSTM-PPO framework introduced in the paper by Agarwal et al. allows noting that, firstly, our LSTM-PPO approach is more advanced in terms of its ability to be adopted into production, its scalability, and its automation potential. Although Agarwal *et al.* are confined to an OpenFaaS simulation where deployment is done manually with a focus on serverless functions, it is performed in the real AWS production environment, in which there is deep integration with SageMaker, S3, CloudWatch, and Lambda. This allows data to be ingested seamlessly, continuously trained, auto-deployed, and monitored in real-time with zero code or developer intervention. Also, our system takes diverse workloads such as serverless, containerized, VM, but theirs is a workload range. This is because by integrating enterprise-grade AWS infrastructure with proven DRL algorithms, our framework was able to provide a truly cost-aware, cloud-native, and future-proof auto-scaling mechanism that is not limited to simulation environments and instead offers research prototypes.

7. Conclusion and Future Work

7.1 Summary of Achievements

This initiative effectively illustrates the efficacy of deep reinforcement learning (DRL) applied in cloud-based auto-scaling mechanisms, which help to make data-driven and intelligent decisions on resource management. Creating a custom Gym setup with conditions matching how clouds behave in practice and instantiating a Proximal Policy Optimization (PPO) agent with Amazon SageMaker, we demonstrated how RL agents can develop solutions capable of creating effective policies for managing compute resources (Guntupalli and R, 2025). Critical elements, such as training, configuration of models, engagement with the environment, deployment of endpoints, and all-inclusive testing, were established and confirmed. The end-to-end system allows performing real-time inference on cloud metrics, which is a significant leap in contrast to rule-based or threshold-based systems.

7.2 Practical Insights

There were a number of lessons learned during the development and deployment process. To begin with, the structure of the personalized setting is of the essence in the learning result. A properly designed reward function with an appropriate balance between the cost and performance is necessary. Second, SageMaker was indeed a useful tool to train and deploy the RL model, but latency in endpoint testing was an urgent reminder of the necessity of model optimisation and initialisation of the environment (Guntupalli and R, 2025). Third, debugging of model errors and the assessment of compression results were made feasible by monitoring tools such as AWS CloudWatch and thorough logging. All of these insights can be applied to future attempts to iterate on the solution and scale it.

7.3 Limitations

Although it met the objectives of what it has to offer, there are still a few shortcomings. The environment is not built on live cloud infrastructure, and so the policies learned in it may lack validity in the real world. There was also a restriction to only five normalized metrics in the observation space, not representing the overall complexity of cloud workloads of I/O, number

of parallel processes, or security aspects (Emma and L, 2025). In some of the inference attempts, we had very high latency at the endpoint, which implies that we need optimization at the container level, or we need to quantize the model so that we can have a quicker response time in production. Finally, there is no ongoing feedback system between the deployment and re-training, which limits the capability an agent can evolve with time.

7.4 Future Work

There are a number of good avenues to take this further. The first improvement would be the inclusion of real-time data feeds of real cloud infrastructure, fixing the simulation-to-production gap. An increased observation space with finer metrics of the system level, and possibly attention mechanisms or LSTMs, with their implementation of temporal reasoning, may lead to more robust policies (Naayini,2025). The other possible project is online learning, where the agent does further refinement of its policy with live performance data (Emma and L, 2025). Part of this might be dynamic container management through integration with Kubernetes or some other orchestration tool. Finally, it would be advisable to perform A/B tests of this RL-based approach against more conventional auto-scaling policies to define their actual value during real deployments.

References

Agarwal, S., Kandpal, M., Shukla, A., Goyal, N., Kishor, K., and Nanda, A., 2025. Cloud Computing for Machine Learning and Cognitive Applications. In *Advancements in Cloud-Based Intelligent Informative Engineering* (pp. 175-202). IGI Global Scientific Publishing.

Alharthi, S., Alshamsi, A., Alseiari, A. and Alwarafy, A., 2024. Auto-scaling techniques in cloud computing: Issues and research directions. Sensors, 24(17), p.5551.

Del Rio, A., Jimenez, D. and Serrano, J., 2024. Comparative analysis of A3C and PPO algorithms in reinforcement learning: A survey on general environments. *IEEE Access*. https://ieeexplore.ieee.org/abstract/document/10703056/

Emma, L., 2023. Designing Scalable Microservices Architectures for AI-Based Fraud Detection in Cloud Environments.

Gallardo, S.R., 2023. *Serverless strategies and tools in the cloud computing continuum* (Doctoral dissertation, Universitat Politècnica de València). https://riunet.upv.es/handle/10251/202013

Jazayeri, F., Shahidinejad, A. and Ghobaei-Arani, M., 2021. Autonomous computation offloading and auto-scaling the in the mobile fog computing: a deep reinforcement learning-based approach. Journal of Ambient Intelligence and Humanized Computing, 12(8), pp.8265-8284. https://link.springer.com/article/10.1007/s12652-020-02561-3

Khan, R.S., 2025. AI-Based Rate Limiting for Cloud Infrastructure: Implementation Guide. *Journal of Computer Science and Technology Studies*, *7*(3), pp.370-380.

Sarabu, V.R., 2025. Resource Partitioning and Provisioning in a Dynamic Computing Environment for AI Workloads. In *Establishing AI-Specific Cloud Computing Infrastructure* (pp. 503-556). IGI Global Scientific Publishing.

Schuler, L., Jamil, S. and Kühl, N., 2021, May. AI-based resource allocation: Reinforcement learning for adaptive auto-scaling in serverless environments. In 2021 IEEE/ACM 21st international symposium on cluster, cloud and internet computing (CCGrid) (pp. 804-811). IEEE. https://ieeexplore.ieee.org/abstract/document/9499535/

Syed, A.A.M. and Anazagasty, E., 2024. AI-Driven Infrastructure Automation: Leveraging AI and ML for Self-Healing and Auto-Scaling Cloud Environments. International Journal of Artificial Intelligence, Data Science, and Machine Learning, 5(1), pp.32-43.

Tsakalidou, V.N., Mitsou, P. and Papakostas, G.A., 2021. Machine learning for cloud resources management--An overview. arXiv preprint arXiv:2101.11984.

Siqiao Xue, Chao Qu, Xiaoming Shi, Cong Liao, Shiyi Zhu, Xiaoyu Tan, Lintao Ma, Shiyu Wang, Shijun Wang, Yun Hu, Lei Lei, Yangfei Zheng, Jianguo Li, and James Zhang. 2022.

A Meta Reinforcement Learning Approach for Predictive Autoscaling in the Cloud. In Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '22). Association for Computing Machinery, New York, NY, USA, 4290–4299. https://doi.org/10.1145/3534678.3539063

Jayaprakasam, B. S. *et al.* (2025) 'A novel cloud-driven adaboost framework for scalable and intelligent data analytics', *International Journal of Parallel, Emergent and Distributed Systems*, pp. 1–29. doi: 10.1080/17445760.2025.2531058.

Oladele, O.K., 2025. Cloud-Native AI Development: Building and Deploying Scalable Machine Learning Models on AWS, Azure, and GCP.

Bilakanti, G. and Engineer, S.D., ENHANCING CLOUD SECURITY AND COMPLIANCE. https://ijetrm.com/issues/files/Apr-2025-11-1744362162-APR24.pdf

Khan, R.S., 2025. AI-Based Rate Limiting for Cloud Infrastructure: Implementation Guide. *Journal of Computer Science and Technology Studies*, *7*(3), pp.370-380.

Risco Gallardo, S. (2023). Serverless Strategies and Tools in the Cloud Computing Continuum [Tesis doctoral]. Universitat Politècnica de València. https://doi.org/10.4995/Thesis/10251/202013

Agarwal, S., Kandpal, M., Shukla, A., Goyal, N., Kishor, K. and Nanda, A., 2025. Cloud Computing for Machine Learning and Cognitive Application. In *Advancements in Cloud-Based Intelligent Informative Engineering* (pp. 175-202). IGI Global Scientific Publishing.

Raza, A., 2023. *Orchestrating Cloud Resources to Optimize Performance and Cost* (Doctoral dissertation, Boston University).

Murthy, J.S., Siddesh, G.M. and Srinivasa, K.G. eds., 2024. *Cloud Security: Concepts, Applications and Practices*. CRC Press.

Goswami, S.A., Darji, D.A., Patel, K.C. and Dave, S., 2025. Establishing AI-Specific Cloud Computing Infrastructure. In *Establishing AI-Specific Cloud Computing Infrastructure* (pp. 123-156). IGI Global Scientific Publishing. https://www.igi-global.com/chapter/establishing-ai-specific-cloud-computing-infrastructure/374435

Guntupalli, R., 2025. Predictive cloud resource management: Developing ml models for accurately predicting workload demands (CPU, memory, network, storage) to enable proactive auto-scaling. AI-driven instance type selection and rightsizing. predicting spot instance interruptions. forecasting cloud costs with higher accuracy. *Available at SSRN 5267834*. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=5267834

Emma, L., 2025. AI-POWERED CLOUD RESOURCE MANAGEMENT: MACHINE LEARNING FOR DYNAMIC AUTOSCALING AND COST OPTIMIZATION.

Yenugula, M., 2023. Monitoring performance computing environments and autoscaling using AI. International Journal of Computing Programming and Database Management, 4(1), pp.90-96.

Gudelli, V.R., 2023. AI-powered insights for performance optimization in AWS cloud environments. *International Journal of Scientific Research and Applications*, *10*(2).

Naayini, P., 2025. Building ai-driven cloud-native applications with kubernetes and containerization. *International Journal of Scientific Advances (IJSCIA)*, *6*(2), pp.328-340.

Mungoli, N., 2023. Scalable, distributed AI frameworks: leveraging cloud computing for enhanced deep learning performance and efficiency. *arXiv preprint arXiv:2304.13738*.

Agarwal, S., Rodriguez, M.A. and Buyya, R., 2021. A deep recurrent-reinforcement learning method for intelligent auto-scaling of serverless functions. *IEEE Transactions on Cloud Computing*, 11(2), pp.1525–1538.