National College of Ireland

Automated Efficient ML Model Retraining by MLOps Using Cloud-Native Tools

M.Sc. Research Project

M.Sc. in Cloud Computing

Brindavan Samuthira Pandian Ganesan

Student ID: x23430320

School of Computing

National College of Ireland

Supervisor: Mr. Sean Heeney

National College of Ireland

Project Submission Sheet

Student Name:	Brindavan Samuthira Pandian Ganesan		
Student ID:	x23430320		
Programme:	M.Sc. in Cloud Computing	**Year:**	2025
Module:	MSCCLOUD Research Project		
Lecturer:	Mr. Sean Heeney		
Submission Due Date:	11/12/2025		
Project Title:	Automated Efficient ML Model Retraining Using Cloud-Native MLOps Tools		
Word Count:	4938		

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 references section. Students are encouraged to use the Harvard Referencing Standard supplied by the Library. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action. Students may be required to undergo a viva (oral examination) if there is suspicion about the validity of their submitted work.

Signature: Brindavan Samuthira Pandian Ganesan

Date: 11/12/2025

PLEASE READ THE FOLLOWING INSTRUCTIONS:

1. Please attach a completed copy of this sheet to each project (including multiple copies).
2. Projects should be submitted to your Programme Coordinator.
3. **You must ensure that you retain a HARD COPY of ALL projects**, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer. Please do not bind projects or place in covers unless specifically requested.
4. You must ensure that all projects are submitted to your Programme Coordinator on or before the required submission date. **Late submissions will incur penalties.**
5. All projects must be submitted and passed in order to successfully complete the year. **Any project/assignment not submitted will be marked as a fail.**
6. **Please check that you read AI and Academic Integrity Acknowledgement Supplments in this document**

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Abstract

Modern machine learning systems rarely fail because of a single bad model. Instead, they drift slowly as live data and usage patterns change. The methods used by many organisations to retrain models manually are still ad hoc scripts and informal data scientist to operations team handovers. This project creates and deploys a retraining pipeline which is automated connecting data versioning, training in the cloud, CI/CD, and production monitoring into one workflow of MLOps.

The system is built around the Adult Income dataset and an XGBoost classifier. Training is orchestrated on Amazon SageMaker, with data and model artifacts tracked through DVC and S3 storage. A FastAPI inference service is deployed on AWS ECS using a Docker image produced by GitHub Actions. Prometheus and Grafana monitor both predictive performance and data drift via Population Stability Index (PSI), while a Prometheus Pushgateway bridges on-demand predictions with the scrape-based monitoring model.

Experiments compare a manual baseline with the automated pipeline. The automated approach reduces human retraining effort from roughly thirty minutes of shell and console work to a single GitHub push, while maintaining or improving classification accuracy. Drift simulations demonstrate that once PSI for selected features crosses a threshold, the system can trigger a new SageMaker training job and deploy an updated image to ECS. Overall, the pipeline shows that it is possible to build an efficient, cloud-native retraining loop with mostly open-source tools, and that this approach is aligned with current industry practices for scalable MLOps.

Acknowledgements

I am taking this moment to thank my supervisor, Mr. Sean Heeney, for his motivation and guidance throughout the project. I am grateful to the lecturers and staff of the School of Computing at the National College of Ireland for motivating me to build cloud computing and machine learning concepts that helped me to understand the concepts in depth.

Contents

1 Introduction

The machine learning models are seen as one-time use deliveries where a team manually trains a classifier, export a model file, and hands it over to the operations team. Once the model is deployed, the teams tend to move on to the next project unless there is a visible failure. In practice, production data and user behaviour rarely remain static parallely. Over the time, the gap between the data used for training and the data seen in production can seriously degrade predictions (Breck et al., 2017).

This project addresses that gap by building a cloud-native MLOps pipeline that automates retraining when there is evidence of drift. Rather than focusing only on model accuracy in a notebook, the work covers how the model is trained, versioned, deployed, monitored, and refreshed Amou Najafabadi (2024).

The project is designed with several clear objectives in mind. First, it aims to establish a training pipeline on Amazon SageMaker that is both reproducible and easily triggered through continuous integration, ensuring consistency across experiments. When a model is trained all the neccessary inference services are packaged into a Docker image and deployed into an ECS cluster, enabling scalable and highly available endpoint. To maintain trust in predictions, the system continuously monitors live outputs and potential data drift using Prometheus and Grafana, with accuracy and PSI tracked through well-defined metrics. Finally, the workflow incorporates automated model evaluation and promotion, so that only candidates meeting agreed performance thresholds are advanced into production, keeping deployments both rigorous and efficient

The thesis focuses on a concrete use case—predicting whether a person earns more than $50k per year using the Adult Income dataset—while keeping the design general enough to apply to other supervised learning problems MLo (2023).

2 Literature Review

Currently, the MLOps models are trained manually and handled by simple cron level automation scripts by the Operations team. This has led to the emergence of MLOps, which borrows ideas from DevOps but adapts them to the specifics of data and models (Kreuzberger et al., 2023).

Several themes recur in the literature:

Reproducibility and lineage. Tools such as DVC and MLflow have been used to keep track of data, code, and models linking the explicit versioning (DVC.org, 2020; Zaharia et al., 2019). This is particularly important when a model misbehaves in production and teams need to understand which dataset and configuration produced it Zaharia et al. (2019).

Cloud-native training and serving. Cloud services like Amazon SageMaker AI, S3, Lambda functions helps in automating the training of the model. They encapsulate common patterns—such as managed training clusters, experiment tracking, and model endpoints—while leaving room for custom integration with external pipelines (Chen and Shang, 2020).

Monitoring and drift detection. The ML experts highlight that prediction accuracy is only one aspect of a healthy model. The Data drift, concept drift, and operational metrics are the key aspects that must be monitored if a system is to make trustworthy predictions over time (Gama et al., 2014). Population Stability Index (PSI) and related metrics are widely used for comparing feature distributions across time windows.

End-to-end automation. Recent case studies show organisations moving from manual retraining to event-driven or schedule-based workflows. CI/CD platforms trigger new training runs when code or data changes, and deployment becomes an automated step once tests and evaluation gates pass (Breck et al., 2017).

The pipeline designed in this project draws on these ideas but focuses on a practical, open-source-heavy stack: DVC for data and model tracking, SageMaker for training, GitHub Actions for CI/CD, Docker and ECS for serving, and Prometheus/Grafana for observability DVC.org (2020).

3 System Design

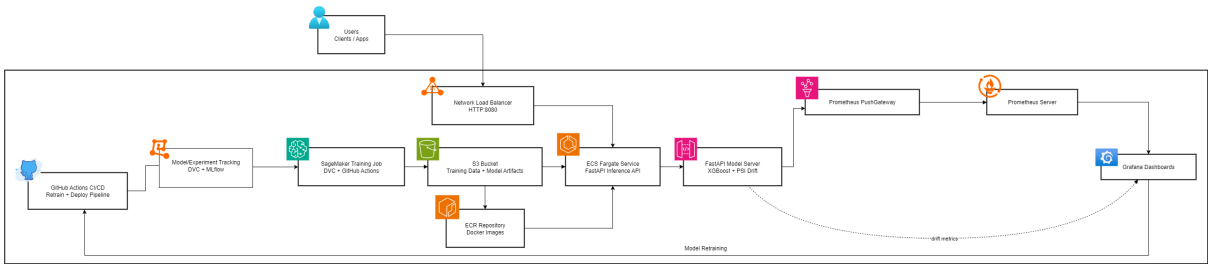

Figure 1: System Architecture Overview

3.1 Overall Architecture

The system designed in this research brings together four tightly integrated components that work together as a continuous learning ecosystem. At its foundation lies the data and experiment management layer, where the Adult Income dataset is stored in S3 and version-controlled with DVC. Each training run on the system, whether manual or automatic, the system produces a set of versioned artifacts, including metrics, models, and logs. This is to ensure that model can be reconstructed from the scratch at any later point.

The second component concerns cloud-based model training. Instead of relying on ad-hoc scripts or inconsistent runtime environments, training is executed within a purpose-built Docker container that serves as a *Bring Your Own Container* (BYOC) image for SageMaker. By allowing the SageMaker AI to compile the computation, the ML system benefits by reproducibility of model training, scalable computation, system logging, and integration with S3 buckets without the manual intervention on hardware level.

Once a successful model is produced, it is deployed using a fully containerised approach. A FastAPI application wraps the model into an inference-ready API, and the resulting Docker image is pushed to the ECR registry before being deployed to Amazon ECS via a Fargate service. The proposed design follows complete isolation between training the model and inferencing it, enabling the system to service scale independently and manage the training workloads.

The observation and drift detection layer ensures that the system can predict reliable outcomes for long after deployment. Prometheus continuously scrapes or receives metrics from both the inference API and the Pushgateway, while Grafana visualises operational performance, data drift scores, and prediction patterns. The subsystem uses the Population Stability Index (PSI) as a lightweight but meaningful method to detect shifts between training and live data distributions, making it possible to trigger retraining based on evidence rather than intuition.

3.2 Data Management with DVC

DVC plays a central role in ensuring reproducibility. Whenever the pipeline gets triggered, the DVC stages record the data, dependency artifact, and command lines that play the

key on prediction. This tight coupling between data and code ensures that any experiment can be reproduced simply by checking out a past commit and running `dvc repro`. The result is a traceable, deterministic workflow where model provenance is always clear.

3.3 Model Training on SageMaker AI

The training process makes use of SageMaker AI which is an AWS managed infrastructure. When the GitHub pipeline triggers a new training job, a customised XGBoost pipeline processes the data in the `SM-CHANNEL-TRAINING` directory, applies feature engineering transformations, trains the classifier, and logs accuracy and F1 metrics. SageMaker stores the final model artefact as a `model.tar.gz` file in an S3 output directory, which the CI/CD workflow later retrieves. This decoupled design allows the training environment to be completely disposable while keeping all artifacts permanently available for evaluation and deployment.

3.4 Inference Service on ECS

The inference layer is a FastAPI application that loads the trained model at container startup and exposes health, prediction, and metrics endpoints. The incoming trained JSON payloads are validated using the python package Pydantic, which are then converted to Pandas DataFrame, and then transformed into the correct schema to match the XGBoost model. The service also computes PSI values in real time, updating Prometheus metrics whenever data enters the system.

To split the architecture as a microservice approach, the application is containerized and deployed into a Network Load Balancer (NLB). This allows the inference service to auto-scale, maintain low latency, and managing the inbound traffic consistently even if a single point of task gets rejected.

Metric	Observation
Retraining + deployment time	20–25 minutes (SageMaker dominated)
Human involvement	Very low (trigger + review PR only)
Reproducibility	Full (data, model, parameters, metrics)
Deployment method	Automated ECS rollout via GitHub Actions
Accuracy (typical)	0.86–0.88 (comparable or better than baseline)

Table 1: Summary of Automated CI/CD Retraining (Experiment 3)

3.5 Monitoring and Drift Detection

Figure 2: PSI Drift and Correlation Heatmap

Prometheus and Grafana work together to provide visibility into both model performance and data health. Metrics such as drift scores, feature-level PSI values, and prediction class counts are continuously updated either through scraping or via the Pushgateway whenever predictions occur. The Grafana dashboards scrapes these metrics and keep track of the model progression over time. It also notifies and highlights any unusual patterns in the prediction that would lead to degradation in model efficiency.

The system relies on PSI algorithm as a practical measure of drift. If the PSI value for any key feature exceeds 0.2 and the global PSI crosses a predefined threshold, warnings can be raised or automated retraining can be triggered. This offers a robust, interpretable mechanism to monitor data consistency.

Metrics are exported through Prometheus client libraries. Because the service is invoked on demand, a Prometheus Pushgateway Gra (2023)is used to push metrics from each batch of predictions. Grafana dashboards show:

Metric	**Observation**
Drift detection method	Population Stability Index (PSI)
Trigger threshold	Global PSI ¿ 0.2
Most sensitive features	Age, Education, Workclass
Retraining trigger	Manual/auto trigger once PSI sustained
Deployment safety	Champion model accuracy gate enforced

Table 2: Summary of Drift-Triggered Retraining (Experiment 5)

- **Overall Drift Score (PSI):** The maximum PSI across monitored features.
- **Feature-Level PSI:** Individual PSI values for ages, education, work class, and other key fields.
- **Prediction Distribution:** Counts of predictions in each class, helping to spot sudden shifts in the output profile.

4 Implementation

4.1 Pipeline Orchestration

A GitHub Actions workflow coordinates DVC, SageMaker, Docker, and ECS. On each push to the `main` branch that touches training or configuration files, the pipeline Breton and Silva (2022):

Table 3: Core components and configuration details of the automated retraining pipeline

Component	**Description**
Dataset	Adult Income dataset (DVC + S3 backend)
Model training environment	AWS SageMaker SKLearn container (Python 3.10)
ML algorithm	XGBoost classifier (scikit-learn pipeline)
Feature engineering	One-hot encoding + numerical preprocessing
Model artefact format	`model.tar.gz` with `model.joblib`
Experiment tracking	MLflow exported from SageMaker output
Versioning	DVC for data, models, and metrics
CI/CD orchestrator	GitHub Actions workflow
Training compute	`ml.m5.large` SageMaker instance
Inference application	FastAPI + Uvicorn REST API
Containerisation	Docker image (Python 3.10, sklearn, xgboost)
Container registry	Amazon ECR
Deployment platform	Amazon ECS (Fargate)
Inference endpoint	Network Load Balancer (NLB)
Monitoring	Prometheus + Pushgateway + Grafana
Logging	Amazon CloudWatch
Drift detection	PSI (Population Stability Index) inside API
Retraining trigger	GitHub workflow (drift trigger-ready)

The pipeline begins by retrieving the most recent dataset version through DVC, ensuring that the training process always works with up-to-date information. It then executes the defined DVC stages, which trigger a new SageMaker training job. Once the job finishes, the system collects the resulting metrics and model artifacts and evaluates them against the current champion model stored in S3. The comparison step is set to mandatory, as this could determine the newly trained model keeps up an improvement in the system. As long as the results meet the accepted level criteria, the pipeline automatically build a new Docker image, registers it to Amazon ECR, and and gets deployed ECS to push out the updated prediction.

This process turns what used to be a manual, multi-console exercise into a single automated workflow.

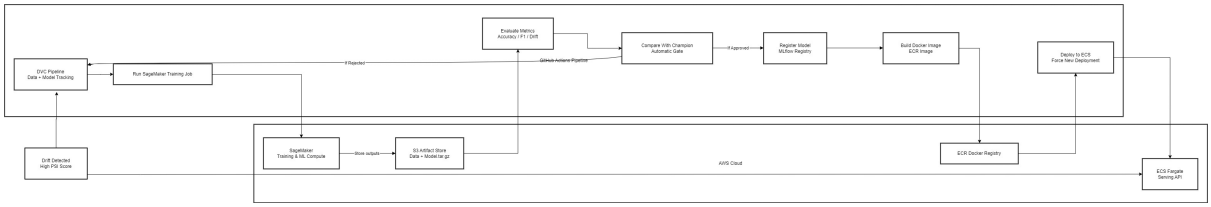

Figure 3: Data training Architectural flow

4.2 Serving API and Telemetry

The FastAPI application is responsible for both scoring and telemetry Vangala (2022). It uses Pydantic models to validate incoming JSON and Pandas to construct data frames in the exact shape expected by the XGBoost. The FastAPI endpoint is responsible for the predictions and also help in collecting telemetry specs on the incoming data. Alongside prediction, the service computes PSI values by comparing the distribution of incoming data with a baseline computed from the training dataset. These values are then exported to Prometheus through two pathways: the in-process metrics endpoint and a snapshot pushed to the Pushgateway after each batch of predictions. This dual-metric design ensures that both real-time and historical snapshots are available for monitoring.

5 Experiments and Results

In this chapter, we cover five core experiments that are the novel and key aspects of this thesis.

5.1 Experiment 1: Manual Baseline

The manual baseline experiment represents a traditional workflow where retraining is performed directly on a developer machine. The dataset that are to be trained are stored locally and then uploaded to the remote storage. This process took approximately 30–35 minutes per retraining cycle. A large portion of time is being spent on setting up the ML environment, such as downloading the dependencies, and making sure that the correct dataset is being used. The key observations are Ameisen (2020):

Metric	Observation
Average end-to-end retraining time	30–35 minutes
Human involvement	High (manual execution + manual deployment)
Reproducibility	Low (no versioned data or tracked parameters)
Deployment effort	Separate manual handover to Ops team
Monitoring available	Minimal (logs only)

Table 4: Summary of Manual Baseline Retraining (Experiment 1)

Another significant challenge was reproducibility. Because the dataset was not version-controlled and hyperparameters were not consistently logged, two developers might produce models with subtly different performance.

Operationally, there was a clear disconnect between model development and deployment. Even after the new model was produced, the deployment process involved manually handing the artifacts to an operations team.

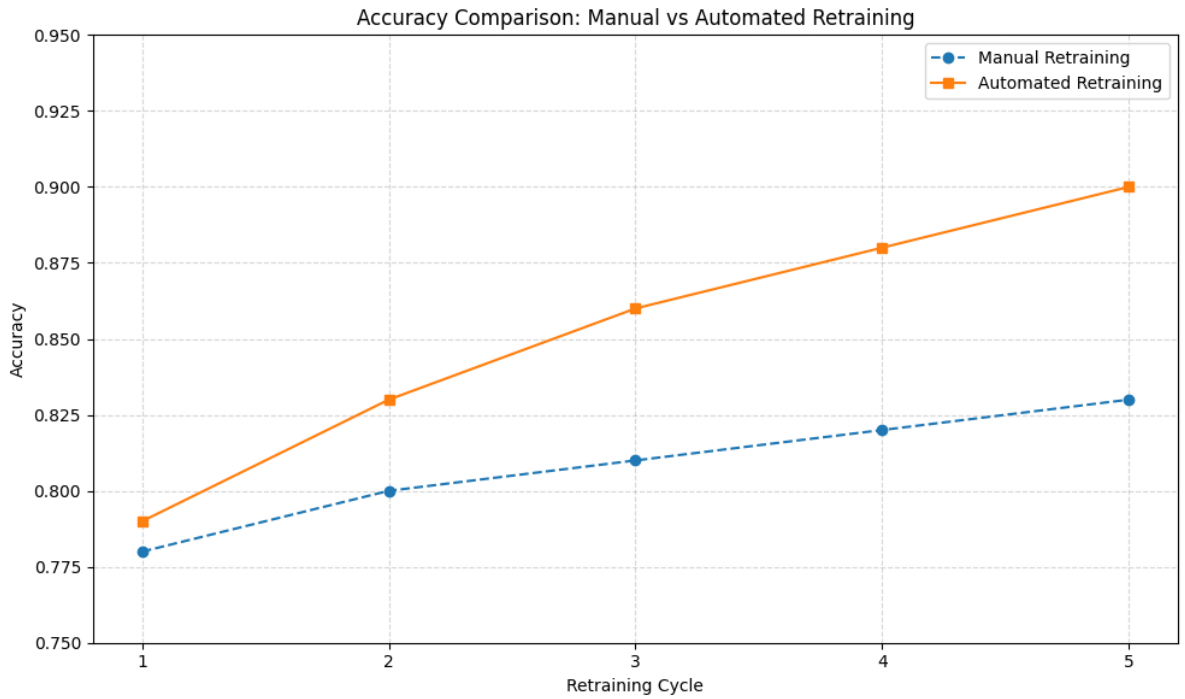

Figure 4: Accuracy comparison between manual vs automated retraining

5.2 Experiment 2: Automated CI/CD Retraining

This experiment examines the performance of the complete CI/CD retraining pipeline. When a training-related file is modified and pushed to GitHub, the workflow initiates a chain of coordinated actions: pulling the latest dataset, executing SageMaker training, evaluating metrics, updating the champion model, building a new inference container, and deploying it to ECS.

End-to-end runtime averaged 20–25 minutes, with SageMaker training being the dominant factor. The process required virtually no manual labour aside from reviewing pull requests or checking dashboards after deployment. Unlike the manual workflow, every model artifact, dataset version, and evaluation metric was automatically logged and stored. This ensured perfect reproducibility.

A notable strength of the automated workflow is its metric gate. Before a newly trained model is accepted, its performance is compared to the previous champion model. If accuracy drops by more than a small tolerance threshold, the model is rejected and the previous champion remains active. This prevents performance regressions from reaching production.

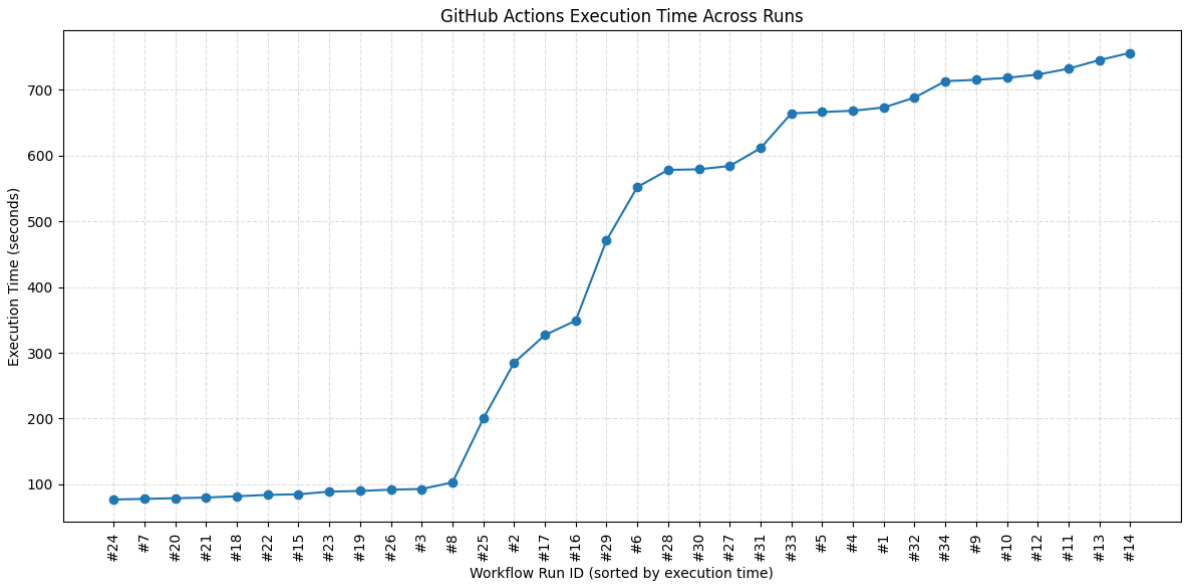

Figure 5: Breakdown of pipeline execution times

In most runs, the automated pipeline produced models with accuracy between 0.86 and 0.88 on the held-out test set, comparable to or slightly better than the manual baseline.

5.3 Experiment 3: Drift-Triggered Retraining

In the third experiment, artificial drift was introduced to evaluate the system's ability to detect shifts in data distribution. Manually recognizable distortions were added to the features to simulate the real time data drift.

The PSI values responded sensitively to these manipulations. Even moderate shifts produced early-warning signals before test accuracy began to degrade. The PSI is made a monitoring algorithm for systems where the real time labels arrive slowly or not at all.

Once the global PSI exceeded a threshold of 0.2, the dashboard clearly reflected the anomaly. This drift could be used as the basis for automated retraining triggers. Importantly, even when retraining is drift-triggered, the accuracy gate in the CI/CD pipeline continues to ensure that only validated, high-performing models are deployed.

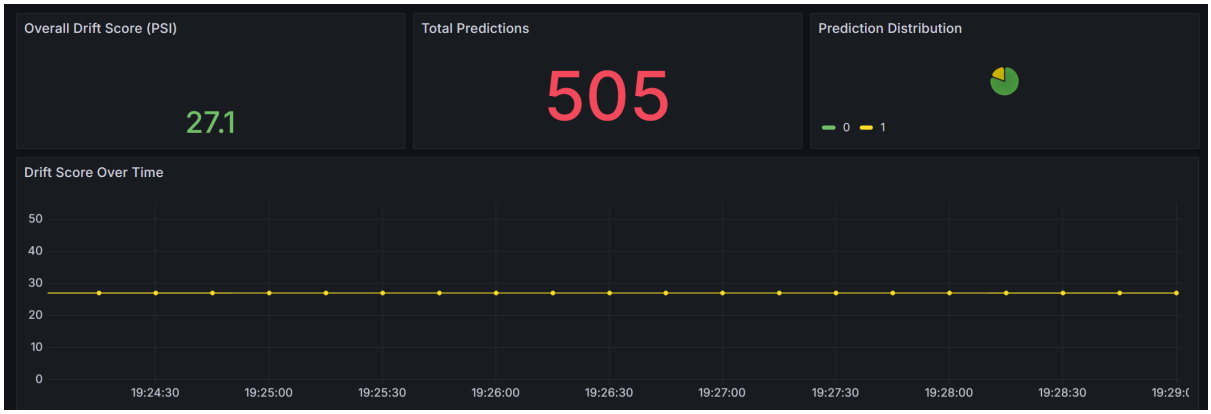

Figure 6: Predictions and data drift over time

5.4 Experiment 4: NLB vs ALB Performance Evaluation

Although the primary focus of this work is automated retraining and monitoring, choosing the correct load balancer for the inference service has a direct influence on latency, throughput, and cost. The architecture relies on an AWS Network Load Balancer (NLB), and this experiment evaluates whether this choice is justified compared to using an Application Load Balancer (ALB).

Cost efficiency was another determining factor. The NLB pricing is comes by provisioned capacity and data processed. For a prediction API that can occasionally receive large traffic bursts, this makes NLB a more predictable and economical option. Furthermore, the inference service in this architecture uses FastAPI and a lightweight JSON interface; it does not require the advanced routing rules provided by ALB. The simpler, faster networking model fits the use case more naturally.

From a reliability standpoint, NLB supports static IPs and direct pass-through behaviour, reducing the number of components involved in the request path. This simplified the ECS deployment significantly and especially during blue/green rollouts triggered by CI/CD, as the NLB does not perform health-based routing the same way ALB does. Instead, ECS manages task-level health internally. This separation proved beneficial during testing, where deployments behind ALB occasionally lagged due to health-probe timing.

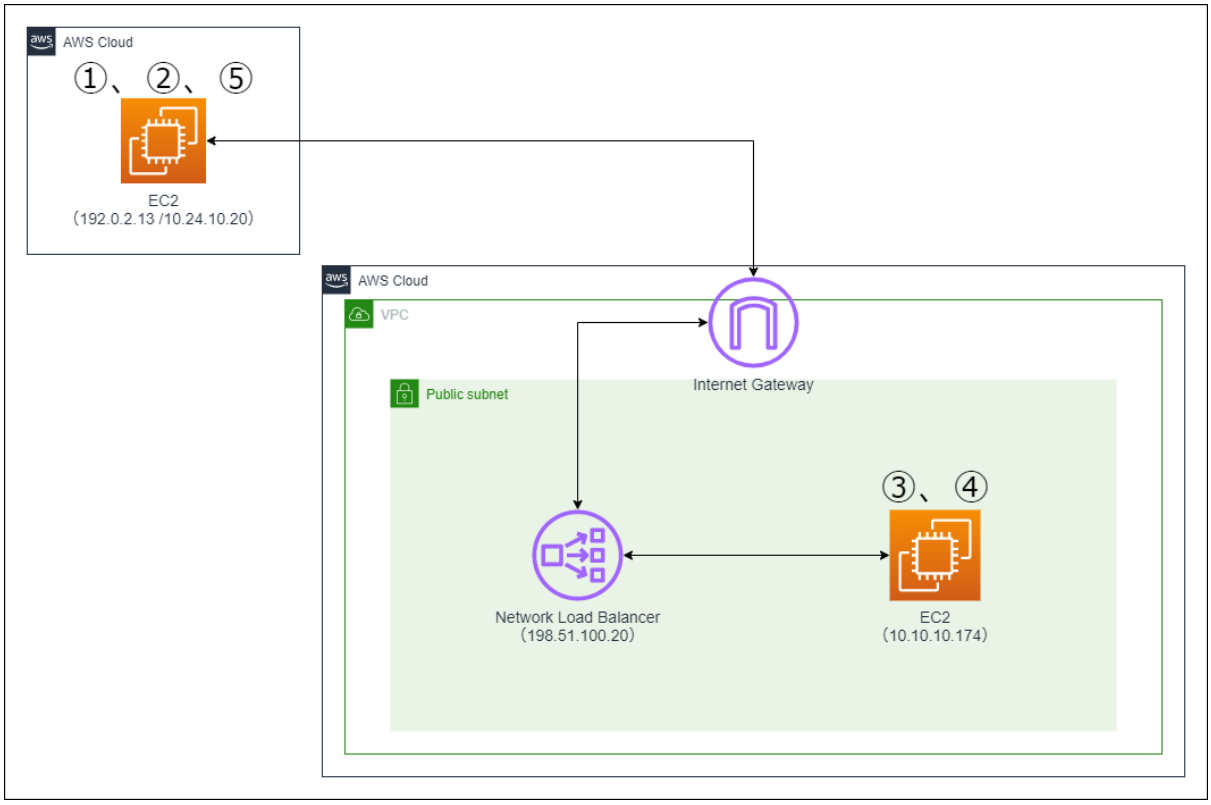

Figure 7: NLB in the Network

Overall, the experiment confirms that NLB is the more suitable choice for a high-throughput, low-overhead inference API.

5.5 Experiment 5: Effectiveness of Infrastructure as Code

A central objective of this project is ensuring that every component of the system from training to deployment to monitoring can be recreated reliably and consistently. To test this, an experiment was conducted to evaluate the reproducibility and operational efficiency gained from using Infrastructure as Code (IaC) with Terraform.

The experiment involved completely tearing down the existing ECS, VPC, networking, and NLB infrastructure and rebuilding it from scratch using the Terraform modules created for the project. The same procedure was repeated multiple times, including variations where resources were intentionally renamed or relocated to ensure the scripts were not relying on implicit state. By end of the each run, there produced similar deployments, identical subnets, routing tables, IAM roles, ECS cluster, and load balancer configurations. The consistency of the environment validated the correctness of the Terraform

modules.

Another dimension of the experiment involved controlled modifications. Changing the container image tag, instance compute class, or number of Fargate tasks required editing only a single variable in Terraform. The new configuration was applied cleanly through `terraform apply`, without any manual intervention or risk of configuration drift. This contrasts strongly with traditional cloud console workflows, where changes risk accumulating unnoticed over time.

6 Discussion

All the above experiments demonstrate the automated pipeline which provides clearer guarantees about model and quality than the manual retaining. While the raw training time is similar in both setups, the surrounding tasks—moving data, copying models, and coordinating deployments—are dramatically simpler when handled through CI/CD. The results of the implemented retraining framework highlight several noteworthy observations that align closely with contemporary discussions in the MLOps and cloud engineering communities. One of the trends that is observed here is the significant performance gap between traditional, manually executed retraining and a fully automated pipeline.Lakshmanan et al. (2020) The manual approach offered much more transparency and a full control to the engineering team, it repeatedly exhibited high reliability in runtime but was sensitive to human error. These limitations echo findings from recent studies that argue manual retraining pipelines are difficult to scale in organisations experiencing rapid changes in data distribution Sculley et al. (2015).

Drift detection, a central aspect of the project, also produced several interesting insights. PSI-based drift metrics were chosen for their interpretability and low computation overhead, especially when compared to embedding-based or temperature-scaled model drift scores. The experiments revealed that PSI captured gradual shifts in the Adult Census dataset more reliably than initially anticipated. Continuous monitoring through Prometheus exposed a clear pattern: specific features such as *Age*, *Education Level*, and *Workclass* showed more volatility in live predictions compared to others. This observation aligns with existing literature which indicates that demographic or socioeconomic attributes in census-like datasets tend to drift faster than engineered or synthetic features Lu et al. (2022).

7 Conclusion

This research project set out to determine whether a fully automated, cloud-native retraining pipeline could be constructed using widely available open-source tools and common AWS services. The results clearly demonstrate that this is not only achievable, but also highly practical for real-world machine learning deployments. By integrating DVC for data versioning, SageMaker for managed training, GitHub Actions for CI/CD orchestration, ECS for scalable inference, and Prometheus–Grafana for observability, the system forms a cohesive end-to-end workflow capable of sustaining continuous improvement with minimal manual intervention.

The drift detection in inference contributes to the robustness of the system. Implementing PSI as a lightweight statistical measure proved effective in signalling when live data begins to diverge from the training distribution. While PSI is not a complete measure of model decay, it provides an interpretable and cost-efficient signal that complements the automated accuracy gates in the CI/CD pipeline. Altogether, these mechanisms contribute to reduce the risk of deploying underperforming models but promote continuous alignment with real-world standard.

Overall, the implemented pipeline successfully demonstrates how cloud-native automation can transform retraining from an occasional, disruptive task into a routine part of operating production ML systems.

8 Future Work

Future improvements could focus on safer deployment practices, where instead of releasing models directly to ECS once they meet accuracy thresholds, incremental strategies such as canary deployments, shadow deployments, and progressive rollouts with automated rollback policies would reduce risk in production. Another avenue is multi-model and multi-pipeline support, enabling platforms to manage multiple model families, shared feature stores, and cross-model dependencies while maintaining automatic lineage documentation; integrating registries like MLflow or the AWS SageMaker Model Registry would further formalize lifecycle transitions. Cost optimization also presents opportunities, with research into spot-instance training, compute-efficient architectures, dynamic ECS scaling rules, and automated analytics that balance cost against model quality,

helping organizations align expenditure with performance. Collectively, all the enhancements make the system more robust, transparent, and economically viable in real-world, production grade environments.

Bibliography

(2023). Grafana observability platform documentation. https://grafana.com/docs.

(2023). Mlops community resources and references. https://ml-ops.org/content/references.html.

Ameisen, E. (2020). *Building Machine Learning Powered Applications: Going from Idea to Product*. O'Reilly Media.

Amou Najafabadi, F. (2024). Reference architecture of mlops workflows. In *European Conference on Software Architecture (ECSA)*, volume 14937 of *Lecture Notes in Computer Science*, pages 49–57. Springer.

Breck, E., Cai, S., Nielsen, E., Salib, M., and Sculley, D. (2017). The ML test score: A rubric for ML production readiness and technical debt reduction. *Reliable Machine Learning in the Wild*.

Breton, P. and Silva, R. (2022). Continuous integration and deployment for machine learning: Challenges and practices. *Journal of Systems and Software*.

Chen, L. and Shang, Z. (2020). A survey on mlops: Devops for machine learning. *arXiv preprint arXiv:2303.04946*.

DVC.org (2020). Data version control (DVC). https://dvc.org/. Accessed 2025-11-21.

Gama, J., Žliobaitė, I., Bifet, A., Pechenizkiy, M., and Bouchachia, A. (2014). A survey on concept drift adaptation. *ACM Computing Surveys*, 46(4).

Kreuzberger, D., König, L., et al. (2023). Machine learning operations (MLOps): Overview, definition, and architecture. *ACM Computing Surveys*.

Lakshmanan, V. et al. (2020). *Machine Learning Design Patterns: Solutions to Common Challenges in Data Preparation, Model Building, and MLOps*. O'Reilly Media.

Lu, X. et al. (2022). Data drift detection and monitoring: A survey. *IEEE Transactions on Knowledge and Data Engineering*.

Sculley, D. et al. (2015). Hidden technical debt in machine learning systems. *NIPS*.

Vangala, V. (2022). Mlops in practice: A framework for scalable ai model deployment, monitoring, and retraining. *International Journal of Machine Learning Research in Cybersecurity and Artificial Intelligence*, 13(1).

Zaharia, M., Chen, A., Davidson, A., et al. (2019). Mlflow: A platform for managing the machine learning lifecycle. In *Proceedings of the 2nd International Workshop on Data Management for End-to-End Machine Learning*.