National College of Ireland

AI-Assisted Migration Assurance: A Coherent Framework of Cloud Deployment with Security and Reliability.

MSc Research Project
MSc Cloud Computing

Apurva Ghodekar
Student ID: X23304995

School of Computing
National College of Ireland

Supervisor: Sean Heeney

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Apurva Ghodekar
Student ID:	X23304995
Programme:	MSc Cloud Computing
Year:	2025
Module:	MSc Research Project
Supervisor:	Sean Heeney
Submission Due Date:	11/12/2025
Project Title:	AI-Assisted Migration Assurance: A Coherent Framework of Cloud Deployment with Security and Reliability.
Word Count:	6312
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:	Apurva Ghodekar
Date:	11th December 2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

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

AI-Assisted Migration Assurance: A Coherent Framework of Cloud Deployment with Security and Reliability.

Apurva Ghodekar
X23304995

Abstract

The automation of cloud migration processes is the new trend, but currently, the vast majority of migration pipelines do not have tools of checking the correctness of infrastructure and its running behaviour post the deployment. Such a loophole leaves organisations vulnerable to configuration fault, security threats, and failure to handle reliability failures, which standard CI/CD pipelines cannot identify. The project creates a single, five-layer cloud migration assurance system comprising of Infrastructure-as-Code provisioning, policy-as-Code validation with Open Policy Agent (OPA), AI-assisted telemetry analysis with the help of LSTM and Autoencoder models, and automated decision-making in a CI/CD pipeline. The system provisions AWS resources with the help of Terraform, implements serverless applications with the help of the Serverless Framework, performs verification of configuration policies prior to deployment, and does runtime anomaly detection with the assistance of real-time CloudWatch metrics. It can be evaluated that OPA policies are effective at identifying misconfigurations, including insecure subnets and weak IAM roles, and the machine-learning models can determine the presence of behavioural anomalies with little telemetry. As the dataset size increases, the model accuracy and stability is enhanced greatly so that the latency spikes and performance deviations can be detected reliably. The findings indicate that policy validation with AI-based runtime analysis is more assuring as compared to the current migration processes. It is a developer-friendly framework that provides a scalable, automated and reproducible framework that will increase the reliability of cloud migrations, correctness and safety, and is a groundbreaking and practical contribution to the research on DevOps and cloud engineering.

1 Introduction

1.1 Background

Cloud migration has emerged as a core element of contemporary digital transformation, facilitating organisations to move out of on-premise systems that are inflexible and non-cost-effective to scalable and automatic cloud systems. Regardless of these benefits, the migration process is not simple as the infrastructure elements, security policies, and application services are to be rebuilt within the target environment correctly. Misconfigurations instead of the software defects are a major cause of cloud failures has repeatedly

been observed in research in the industry, with many small configuration oversights, including cloud storage policies that leak sensitive data by default, being reported on numerous occasions (Sharma and Chen; 2021; Humayun and Niazi; 2020). These issues support the need to have the validation mechanisms that would identify the configuration errors that can otherwise lead to the loss of operational security or reliability.

Cloud deployments have become fragile and automated as Infrastructure-as-Code, serverless architecture and CI/CD pipelines continue to define the way modern development is created. IaC tools enable quick infrastructure construction, however, these tools do not necessarily impose compliance or correctness; and equally, CI/CD pipelines usually test application logic only, and do not consider how the deployed environment will behave. According to recent studies, a variety of cloud migration models focus more on orchestration and automation without offering multiple mechanisms to guarantee the post-deployment process (Faria et al.; 2024; Tao; 2024). It is due to this gap that there is a need to create intelligent validation frameworks that can detect misconfigurations and anomalies at a very early stage of the migration workflow.

1.2 Research Problem

In spite of the wide range of automation in provisioning and deploying applications in cloud platforms, these features fail to automatically assure that migrating environments are secure, compliant, and operationally reliable. Misprovisioned network components, weak IAM roles assignments, wrong database parameters, and unhealthy performance metrics can be ignored by CI/CD pipelines, as the traditional pipelines do not check infrastructure and runtime behaviour in an organized way. According to previous researchers, failures in migration that can happen are usually due to lack of validation following deployment; this exposes organisations to reliability, cost, and security vulnerabilities (Faria et al.; 2024; Tao; 2024). This study thus meets the necessity of having a comprehensive validation structure that evaluates policy adherence and telemetry of operations.

1.3 Research Objectives and Research Question

This study aims to design and implement a multi-layer cloud migration assurance framework that integrates policy-as-code verification, Infrastructure-as-Code provisioning, AI-assisted telemetry analysis, and CI/CD governance. The framework automates the provisioning of AWS services, evaluates infrastructure configurations through Open Policy Agent (OPA), analyses runtime signals through machine-learning models such as LSTMs and Autoencoders, and incorporates these validation outputs as decision gates within the CI/CD workflow. It is aimed at facilitating the migration processes initiated by the developers through automated validation, anomaly detection, rollback, and reproducible artifacts in accordance with the DevOps practices. This framing puts the emphasis on the migration pipelines facing the developers directly instead of platform-oriented migration pipelines and tooling so that the framework tackles the challenges that are faced directly in day-to-day development processes.

The research is guided by the following question:

To what extent can an AI-assisted validation framework improve the reliability of automated cloud migration through integrated policy and runtime behavioral analysis?

1.4 Conclusion of Introduction

This thesis is developed in a coherent flow to provide the research and its contributions. The following sections review the available literature on cloud migration, Infrastructure-as-Code, policy-as-code technologies, and AI-based anomaly detection. The section focused to the methodology then describes the conceptual design of a multi-layer assurance pipeline and how every element of the validation communicates in the deployment workflow. The implementation section outlines the way the system was built with the help of AWS, Terraform, Serverless, OPA, and machine-learning models. Performance, accuracy and reliability of the framework are evaluated in the evaluation section. Last but not least, the thesis ends with a conclusion of findings and prospects of future improvement.

In this chapter, the motivational factor behind the study was given, the research problem was defined, and the objectives and the guiding research question were presented. The rest of the thesis is based on this premise as the review of the previous work and the description of the methodology, implementation and evaluation of the proposed multi-layer cloud migration assurance framework.

2 Related Work

Cloud migration research spans multiple domains, including strategic decision-making, infrastructure automation, security assurance, and operational optimisation. Although each of these areas contributes valuable insights, they often operate in isolation, resulting in fragmented migration practices that fail to guarantee reliability after deployment. Existing studies highlight the importance of planning and orchestration, while others showcase the potential of automation, policy enforcement, and artificial intelligence in improving cloud operations. However, the literature reveals a clear absence of integrated frameworks that unify these components into a single migration assurance pipeline. This chapter reviews the most relevant strands of research to understand their strengths and limitations and to position the contribution of the present study. Notably, the reviewed studies do not suggest a single framework which incorporates IaC, policy-as-code, and AI-based validation into a CI/CD piping.

2.1 Cloud Migration Frameworks and Strategies

The idea of cloud migration has received extensive analysis in terms of its strategic, architectural, and operational aspects. Numerous other studies dwell on the conceptual and planning parts of migration instead of implementing the right methodology post-deployment. The framework of structured decision-making proposed by Patel et al. (2024) will help organisations to choose migration strategies, risk assessment, and align cloud adoption with business goals. Likewise, Garg et al. (2024). introduce the roadmap that underlines the process of the migration of on-premise ERP systems to cloud platforms and focuses on governance, communication with stakeholders, and systematic planning.

Although such contributions reveal that migration is to be carefully prepared and managed, they view deployment activities, in particular, validation, as being mainly

manual processes. They give top-level advice and do not touch upon the intricacies of checking infrastructure behaviour after the deployment.Complementary studies are done on how to automate migration processes. Faria et al. (2024). present a serverless-based automation framework that has AWS services that coordinate the steps in migration. Their methodology decreases the burden of operations and enhances consistency, but does not have a mechanism to judge the validity of the resulting environment. Kumar et al. (2024). pay attention to the portability of migration and minimize the vendor lock-in through service deployment model reconfiguration. Their work is more efficient at automation but it has not yet reached the level of validation layers detecting any misconfigurations or operational risks. So, the current frameworks reinforce planning and automation but rarely include automated policy validation, runtime verification or post-migration validation These models are primarily used to aid planning and automation as opposed to offering automated and repeatable post-deployment validation.

2.2 Reliability and Security in Cloud Migration

The key issue during cloud migration is security and reliability. Dhaman and Suman (2024) compare security measures applied to data migration to clouds by analysing cryptographic measures that apply to safeguard data in transit. Their publication points out the value of encryption and secure channel but is restricted to data level issues. The work of Kaur and Gargrish (2024)is devoted to the idea of secure virtual machine migration, with the authors suggesting the mechanisms of authentication and integrity assurance to ensure the safety of transitions between hosts. These works prove that there is a need to have technical protection, but they consider only several layers, but not the entire deployment life cycle. In addition to security, a number of researchers also estimate reliability and performance risks of migration. Zou (2024) puts forward a demand-sensitive migration model which is trained on deep reinforcement learning to enhance stability, resource distribution and responsiveness to the fluctuation of the load. (Salanke et al.; 2024). use the methods of predictive modelling to predict failures in migrations and initiate corrective measures. Nevertheless, these papers mostly use simulation platforms such as CloudSim, and not the actual development process.

2.3 AI and Machine Learning for Cloud Operations and Anomaly Detection

There is a fast emerging field of cloud studies exploring how AI and machine learning can be used to improve operational intelligence. Lokhande et al. (2024). create a hybrid anomaly detection methodology based on LSTM networks and graph neural networks to detect abnormal working patterns in multi-cloud environments. This evidences the usefulness of deep learning in real-time monitoring of complex systems.Abouelyazid (2024) proposes Deep-Hill, a machine learning-based technology to optimise SaaS instances configuration based on performance and responsiveness.

Sustainability and resource optimisation are also becoming the use of AI. The framework suggested by Agomuo et al. (2024). is based on LSTM forecasting, and reinforcement learning aimed at minimising energy usage without deteriorating the quality of the provided services. Beena et al. (2025). come up with a carbon conscious scheduling mechanism that can adjust resource allocation in response to carbon intensity signals. Bahreini et al. (2024) provide algorithms to schedule energy efficiently in a distributed

data centres and Kubernetes clusters, whereas Singhal et al. (2024). demonstrate how it can enhance the scaling of an intelligent load balancing during variable workloads.

Though they are good, these AI-based developments usually operate as independent of cloud migration processes. They can be run on run-time telemetry, scheduling, or infrastructure orchestration, but they are rarely integrated into develop pipeline systems, or run to verify that they are correct after deployment. Subsequently, AI has not been fully exploited in early-stage migration assurance.

2.4 Research Gap

The available literature also notes two major constraints of current practices in cloud migrations: first, most of the available cloud migration research focuses on strategic planning, orchestration, and automation, yet does not offer any post-deployment validation, and existing models tend to provide strategic guidance and partial automation without using any systematic mechanism to ensure that deployed infrastructure is secure, properly configured, or operating as intended. Validation is mostly manual, piecemeal or limited to a specific technical layer like encryption or VM integrity. Second, despite recent achievements in AI and machine-learning research, such as anomaly detection, performance prediction, energy optimisation, and scheduling, they are not usually integrated into operational processes or deployed to assist developers when they are actually performing cloud migration processes. Based on these trends, a distinct research gap will be identifiable: a single unified framework that integrates Infrastructure-as-Code provisioning, policy-as-code validation, machine-learning-based telemetry analysis, and CI/CD enforcement as one automated migration assurance pipeline has not been created yet.

None of the existing literature demonstrates how these elements can interact with each other: to automatically provision infrastructure, test it against organisational policies using OPA, model runtime behaviour using AI models and use CI/CD decision gates to deploy or rollback deployments. The current study aims at addressing this void by suggesting an approach to implement a multi-layer framework of migration assurance that integrates policy verification and AI-based telemetry analysis directly in the process of the cloud deployment model to facilitate automated verification, data-driven decision-making, and more trustworthy cloud migrations. It is the gap that this research study is aimed at filling by suggesting a developer-centric migration assurance model that incorporates IaC, OPA-based policy validation, AI-based telemetry analysis, and CI/CD decision enforcement.

3 Methodology

This section is aimed at describing the methodological approach that will be employed to design, implement and evaluate the proposed cloud migration assurance framework. The study takes the design science approach since the task is to design an artefact a multi-layer automation and validation pipeline which addresses a practical issue in cloud migration reliability. The method is complemented by experimental testing, which allows the system to be tested with known misconfigurations and runtime abnormalities. The combination of these makes the methodology not only generate a functional solution, but also has empirical evidence of its working.

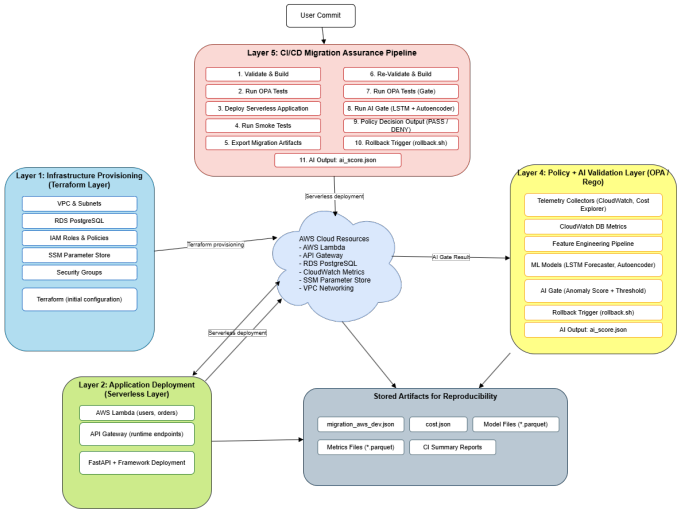

Figure 1: 5-layer CI/CD assurance pipeline diagram

3.1 Research Design Rationale

The problem of cloud migration is complex, and it encompasses both provisioning of infrastructures and policy compliance, as well as runtime behaviour. To solve these mutually supporting problems, the methodology is based on a layered system design and enables each step of a cloud deployment process to be isolated, tested, and proved. The research design is aimed at addressing the most important question of whether AI-aided validation can be used to improve the reliability of migration. It does so by building a pipeline that uses Infrastructure-as-Code provisioning, policy-as-code analysis, anomaly detection through AI and CI/CD enforcement to allow the behaviour of every layer to be analysed in different scenarios.

3.2 Architecture Design and Tools Justification

The framework combines Terraform, Serverless Framework, OPA/ Rego, AWS services, and machine-learn models because of their popularity, compatibility and automation features. The reason why Terraform is chosen is that it offers deterministic and repeatable infrastructure provisioning that minimizes the chance of mistakes within manual configuration. The reason why OPA is used is that it allows declarative and auditable policy enforcement, which ensures that all deployments meet the needs of organisations. The AWS services used, which include Lambda and CloudWatch, RDS, API Gateway, and SSM Parameter Store, are selected so as to represent the current serverless architecture in a real world cloud. The machine-learning models facilitate the anomaly detection layer since AI methodologies are more efficient than rule-based checks in detecting slight behavioural anomalies, which manifest after the deployment.

3.3 Telemetry Collection and Dataset Preparation

The validation that AI performs needs proper telemetry to distinguish between normal and abnormal operational conditions. Telemetry is gathered on AWS CloudWatch since it gives real-time metrics of latency, CPU usage, memory performance, database throughput and error rates of the applications. These metrics are the foundation of the set of data that is utilized to train and test the LSTM and Autoencoder models. An application of feature engineering is used to convert raw logs into meaningful inputs such as normalising values, creating sequences to use in time-series prediction and patterns that might be correlated with misconfigurations or system instability.

Table 1: Tools and Technologies Used in the Proposed Framework

Tool / Technology	**Why It Was Used**
Terraform	Provides deterministic, reproducible infrastructure provisioning and reduces human error in cloud resource configuration.
Serverless Framework	Automates packaging and deployment of AWS Lambda applications, enabling consistent and scalable application deployments.
Open Policy Agent (OPA) / Rego	Offers declarative policy-as-code validation to enforce security, compliance, and configuration rules before deployment.
AWS CloudWatch	Captures real-time telemetry (CPU, latency, errors, DB metrics) required for anomaly detection and model training.
LSTM Model	Suitable for time-series forecasting to predict expected resource behaviour and detect deviations over time.
Autoencoder Model	Effective for unsupervised anomaly detection by learning the normal operational patterns of cloud workloads.
GitHub Actions (CI/CD)	Provides automated pipeline execution, gating logic, rollback mechanisms, and artifact management for reproducible deployments.

3.4 AI Model Development, Policy Validation, and Runtime Enforcement

The machine-learning aspect of the methodology combines two types of models- LSTM forecasting and Autoencoders- since they are effective in identifying abnormalities in time varying workload of clouds. LSTM gives forecasts on the likely behaviour of the system, and the Autoencoder determines the anomalies to the normal patterns that have been learnt. These products constitute the "AI Gate" that decides on proceeding with a deployment or rolling over. Simultaneously, OPA policies assess the accuracy of infrastructure by verifying VPC architecture, IAM role definition, enforcement of HTTPS, RDS config, as well as subnet assignment. This policy validation with AI anomaly detection forms a two-level assurance system: OPA will certify the correctness of a policy

should be before deployment, whereas AI will certify that things are running correctly at runtime.

3.5 CI/CD Pipeline Integration

CI/CD pipeline is used as the control layer, which coordinates validation and decision-making throughout the system. It is implemented via a GitHub Actions or GitLab CI workflow since it allows testing to be automated, and gates can be provided, as well as rollback. The pipeline performs Terraform deployments and Serverless deployments, OPA checks, smoke tests, analysis of telemetry (AI models) and the rule of thumb is to promote or reject the deployment. Experiments are run by injecting operational misconfigurations and runtime anomalies to determine whether the framework identifies and blocks faulty migrations.

3.6 Methodology Summary and Justification

The research has some limitations on its research methodology that affect the extent and generalisability of research. The machine-learning models require a lot of quality and quantity of the telemetry data they receive, and as a result, their accuracy to predict might be limited by the comparatively small size of the dataset generated in the course of the experiment testing. Also, the framework is only applied to AWS and this can restrict its use to other cloud services that have varied service architectures and policy frameworks. These constraints establish the range of interpretation of these results and indicate the areas in which the model can be expanded to enhance its strength and usability in the future. In spite of these limitations, the methodology offers a discipline and orderly way of measuring the reliability of the cloud migration activities. The framework provides a two-fold guarantee scheme by incorporating the policy-as-code validation, as well as runtime behaviour assessment through the use of AI-based telemetry analysis. A combination of these validation layers into a CI/CD pipeline will allow making automatic decisions, rollback, and repeatable processes. This experimental approach and design science can be used to developing a strong, reproducible, and scalable framework of migration assurance that can enhance the reliability and openness of cloud deployment by developers. Combined, these methodological decisions directly answer the research question, since they allow the framework to be tested regarding its ability to avoid misconfigurations, detect runtime anomalies, and automatically make safe deployment decisions.

4 Design Specification

The architecture of the suggested model of migration assurance is aimed at designing a multi-layered structure that will incorporate infrastructure provisioning, application deployment, policy verification, telemetry analysis, and CI/CD control. This design is aimed to provide an opportunity to make all the stages of the cloud migration process legitimate, controlled, and repeatable. This chapter outlines the architectural choices, the interaction of components, validation workflow, and flow of data that all help in supporting achieved and automated cloud migrations.

4.1 High-Level Architecture Overview

It is designed based on a five-layer architecture that corresponds to the steps of the current cloud deployment pipelines, The five-layer model was selected to reflect the phases of a current DevOps-providing, deploying, validating, monitoring and governing pipelines and ensure that each issue was modular and could be tested individually, which include Infrastructure Provisioning, Application Deployment, Policy Validation, Telemetry and AI Validation, and CI/CD Migration Assurance. Such a layered organization makes every step of the process, including the creation of cloud resources and their end deployment, perform their own validation tasks and add to the overall assurance process. The layers communicate with each other using Terraform outputs, API Gateway settings, runtime telemetry, and CI/CD gating systems which constitute an integrated system where the infrastructure correctness and application behaviour are constantly assessed. Such an interconnection design helps the framework to identify misconfigurations at an early stage, react to anomalies in running, and maintain a steady level of reliability during the migration process.

4.2 Validation Flow and Interaction Design

The design has various validation check points that are spread all over the pipeline. The process of validation starts with infrastructure validation based on OPA rules, which analyze the output of Terraform. After the deployment of the application, anomaly detection with the help of AI is performed on the telemetry of CloudWatch. The CI/CD pipeline organises these phases with each test running one after another, policy and AI results are analysed and automatic rollbacks are performed in case an error or abnormality is identified. This design guarantees configuration correctness as well as runtime behaviour is checked out prior to finalising migration steps.

4.2.1 Data Flow and Model Integration

The system will facilitate policy evaluation as well as the detection of anomalies through the flow of data in a structured fashion. Terraform outputs are used as the input of OPA policies, which allows to analyze configuration files statically. In the meantime CloudWatch metrics are fed to the feature engineering script which prepares sequences to the LSTM and Autoencoder models. The scores of anomalies obtained by AI are sent back to the CI/CD pipeline where they impact deployment. These data flows can be designed explicitly, enabling the framework to guarantee the reproducibility of these data flows, and allow the effortless integration of AI predictions into migration governance.

4.3 Justification and Summary

The selected architecture is a compromise between modularity, scalability, and automation, so the individual components of the framework have the opportunity to evolve uncoordinated and nonetheless integrate successfully with the rest of the validation pipeline. Each layer has a specific task it does but the assurance workflow is interrelated and assists in troubleshooting and part reuse on a system-wide basis. Terraform and OPA provide deterministic infrastructure provisioning and policy enforcement, minimising human error and enabling repeatable deployments. By comparison, the LSTM and Autoencoder models cope with dynamic runtime variability that is not able to be considered by policy

checks, providing a behavioural view of the health of a system. The CI/CD pipeline is the system which incorporates all these components to a single control plane that automatizes the decision-making process, rollback behaviour and artifact traceability. This design will allow the system to be not only functional, but also to be consistent with the actual DevOps and cloud engineering practices, where automation, validation, and continuous improvement are the key factors.

In general, it is possible to note that the architecture is supposed to offer a holistic and dependable process of validating cloud migrations through the integration of the following elements: the static analysis, behavioural modelling, and automated governance. The framework provides preventive and corrective validation services by designing the system in a layering of functional works, applying AI-driven telemetry to policy enforcement. This combination enables the system to identify configuration errors before they take place, check on the anomalies of the runtime, and keep the migration reliability the same across deployments. The design produced is highly suitable in term of experimentation, evaluation and subsequent expansion in cloud automation research.

5 Implementation

The implementation phase converts the conceptual architecture to a fully operational platform that automates, validates and controls cloud migration processes. This chapter describes the manner in which all the components of the framework were constituted, the rationale behind the necessity, the exact functions it carries out, and the interaction with the other components. The implementation is focused on simplicity and reproducibility to ensure that a system can be deployed responsibly and expanded by the developers. All layers play a part in a structured process with cloud resources being automated and provisioned, policy checked, AI model analysed, and controlled by a CI/CD pipeline.

5.1 Infrastructure Provisioning Layer

The initial aspect of the implementation is on the Infrastructure Provisioning Layer which is there to minimize the high probability of human error in case of cloud migrations. Three typical causes of the failures of cloud deployments are manual configuration errors, which may include wrong location of subnets, insecure security group policies, or poorly defined IAM access. In order to mitigate this risk, all core AWS resources were written in Infrastructure-as-Code. The reason behind the selection of Terraform was that it guarantees that the process of provisioning is predictable and repeatable, which is critical when it comes to sustaining consistency of migration. Practically, Terraform scripts were developed to make up the VPC, subnets, RDS PostgreSQL instance, IAM roles, API Gateway base configuration, and SSM parameters. It was done with the help of variables in order to ensure that the infrastructure was adaptable to new settings. The systematic outputs like resource identifiers with endpoint URLs produced after deploying Terraform were then sent to pipeline stages. This methodology would make sure that the basis behind the system is never implemented in an unregulated and untraceable manner.

5.2 Application Deployment Layer

The second key element is the Application Deployment Layer whose duty is to package and deploy the application in a uniform environment-independent way. The layer is

due to the fact that application behaviour may not be the same in case dependencies or runtime configurations have not been handled properly. The structure employs a serverless design, which implies that the application will run on AWS Lambda and will use API Gateway as a communication method. The FastAPI based application is developed with the Mangum adapter enabling the ASGI based applications to run directly in Lambda. The Serverless Framework takes care of deployments since it automates packaging, IAM setup, injection of environment variables, and mapping of routes. In the implementation, the application was structured into readable modules, and `requirements.txt` file made sure that Lambda was provided with all the required dependencies. The CI/CD pipeline was used to deploy the system automatically and simple smoke tests were used to verify that endpoints were reachable and operating as expected.

5.3 Policy Validation Layer

The third layer in the system is the Policy Validation Layer whose aim is to intercept errors in the configuration that can lead to the system having issues in production. Even in the case of automatically generated infrastructure, there may still be problems with insecure public subnets or inappropriate IAM permissions. The layer is based upon Open Policy Agent (OPA) to analyse the output files of Terraform against a collection of predeveloped Rego policies. These policies have significant rules enforced in organizations, including forcing only HTTPS-only traffic, putting all databases into private subnets, ensuring that the IAM roles adhere to the least-privilege principles, and making sure that security-groups do not put the system on the open internet. Terraform was set up to create a JSON file named `migration_aws_dev.json` which has all definitions of resources. This file is read by OPA where each rule is checked. In case of failure of any of the rules, the deployment is halted on the spot by the CI/CD pipeline. This will guarantee that no non-complaint infrastructure is ever implemented.

5.4 Telemetry and AI Validation Layer

The fourth element is Telemetry and AI Validation Layer that supplements the policy layer by examining the real behaviour of deployed system. The runtime problems that cannot be identified in the course of the static checks include sudden latency spikes, CPU overload, or even minor performance anomalies. In order to fill this gap, CloudWatch telemetry data, including Lambda duration, API latency, RDS CPU utilization, and error rates, were gathered and trained two machine-learning models, including an LSTM forecaster model and an Autoencoder anomaly detector. The reason behind the selection of these models is the fact that they process time-dependent data and are able to learn normal behavioural patterns without having labelled datasets. A data pipeline (written in Python) periodically gathered metrics and made them ready to be analyzed. LSTM was used to predict the appearance of normal values, and the Autoencoder quantified reconstruction error. Their combination created an anomaly score. Assuming that this score surpassed a threshold, the system identified abnormal behaviour and indicated this to the CI/CD pipeline, so that an automated rollback could be performed as desired.

5.5 CI/CD Migration Assurance Layer

The fifth significant element is the CI/CD Migration Assurance Layer, which also coordinates all validation steps and finalizes the deployment decisions. The layer is explained by the fact that manual reviews are vulnerable to errors, and slow, whereas automated governance is consistent. This pipeline was implemented via GH actions, with every step being a workflow step. The pipeline initially confirms the code and develops the project, before executing the OPA against outputs of Terraform, deploying the serverless application, smoke tests, exporting migration artifacts, executing the AI validation models, and finally deciding on a decision to move on or roll back. To have develop rollback, the use of a simple script which restores the last known working version was adopted in order to make sure that there is safety and reversible deployments. This workflow will make sure that the infrastructure correctness, application behaviour and runtime performance all will be verified before a migration can be defined as a success.

5.6 CI/CD Migration Assurance

In this subsection, a consolidated view of the CI/CD stages of assurance will be given. Although the above section outlined the CI/CD Migration Assurance Layer implementation, Table 2 summarises each checkpoint and its role within the automated migration pipeline.

Table 2: Overview of CI/CD Migration Assurance Stages

Stage	**Summary**
Infrastructure Setup	Terraform provisions cloud resources required for the application.
Policy Validation (OPA)	Terraform outputs are checked against security and compliance rules.
Application Deployment	Lambda functions, API routes, and configurations are deployed.
Smoke Testing	Basic endpoint-level tests validate functionality post-deployment.
AI-Based Validation	CloudWatch telemetry is analysed by LSTM and Autoencoder models.
Deployment Decision	CI/CD approves or rolls back the deployment based on validation results.

The implementation is able to transform the proposed five-layer architecture into a running system that automatically provides cloud provisioning, policy validation, anomaly detection, and CI/CD governance. These layers are introduced to serve a particular purpose, but they collaborate to deliver a uniform and consistent migration assurance process. It has led to a useful, developer-friendly framework which minimizes the risk of misconfigurations, enhances reliability and shows the manner in which AI can be woven in to cloud migration pipelines in a meaningful way. In general, the given implementation achieves the five-layer architecture as a functioning system which can be ran and executed end-to-end by the developer commit. That is the basis on which the evaluation will be made in the next chapter where the concrete evaluation of each assurance mechanism is evaluated under controlled experiments.

6 Evaluation

The section will measure the performance of the proposed multi-layer cloud migration assurance framework. The evaluation does not simply aim to evaluate the correctness of each of the components separately, but rather to find out how these checks of policy as a whole, machine-learning models, and CI/CD governance provide a higher level of reliability than a typical deployment pipeline. The test is based on a set of controlled experiments, which target three aspects, namely, the prevention of misconfigurations, the ability to detect the presence of runtime anomalies at an early stage, and safer deployments resulting of the use of automated CI/CD decisions. This chapter contains eight figures of experiments in order to demonstrate the behaviour of the Autoencoder and LSTM models during the use of a small dataset and a full dataset.

6.1 Experiment / Case Study 1

The experiment and first one is the effectiveness of the Policy Validation Layer (OPA/Rego) in identifying typical misconfigurations during cloud migration, including insecure public subnets, excessively lenient IAM roles, lack of HTTPS enforcement, and weak security group configurations. These were deliberate bugs placed on the Terraform templates to determine whether the system would detect unsafe deployments and prevent it. Policy-as-code is essential since any minor oversights in infrastructure may cause a failure (crash), attack, or a non-conforming environment, particularly in automated pipelines, where human inspection remains restricted. The experiment illustrates the implementation of the organisational rules by OPA towards the beginning of the workflow so that the violations can be detected prior to the implementation of the application layer. In both instances of tests, OPA successfully reported the misconfigurations injected and immediately caused a CI/CD pipeline to fail as a result, proving that this layer is a dependable first line of defence. These findings also indicate that embedding the policy assessment into the automated processes can significantly decrease the operational risks and enhance the reliability of the cloud migration that was performed by the developers.

Algorithm 1 OPA Policy Validation Process

Require: terraform_output.json
1: **for** each policy_rule in OPA_policies **do**
2: result ← evaluate(policy_rule, terraform_output)
3: **if** result == "deny" **then**
4: mark_pipeline_failed()
5: stop_deployment()
6: **return**
7: **end if**
8: **end for**
9: **return** "all policies passed"

6.2 Machine-Learning Prototype Evaluation

The second experiment measures the Autoencoder and LSTM models performance under the condition of a specifically small telemetry dataset, which is typical of the situation in early cloud systems when there is not much historical data at their disposal. The

histogram of reconstruction-error of the Autoencoder indicates that the majority of values are concentrated around the point of zero, although there are also large spikes here and there, which indicates that the model is capturing general trends of normal behaviour, but at the expense of a lack of training samples. This is supported by the time-series plot which indicates that spikes of errors are way above the baseline thus indicating that even though the model is sensitive to abnormal runtime behaviour, its threshold changes over time.

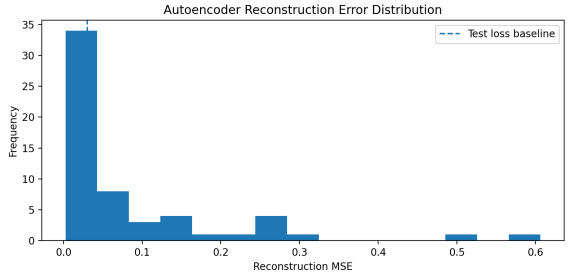

Figure 2: Autoencoder reconstruction error distribution (small dataset)

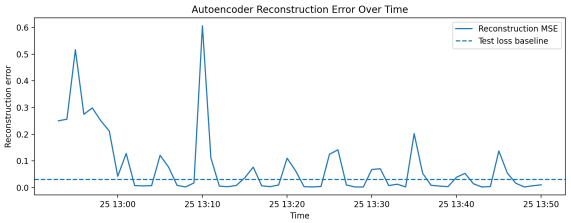

Figure 3: Autoencoder reconstruction error over time (small dataset)

On the same note, the LSTM model, in the comparison of predicted and actual database read time, would follow the overall trend, but would fail to be fine-grained based on the predictions, as it would have been with a time-series model trained on limited data. A comparative boxplot has shown that the Autoencoder has a greater error variance, and so is more sensitive to anomalies but also less stable, whereas LSTM has a smaller error variance and thus more stable but less sensitive to anomalies. In general, this experiment proves that machine learning applied in telemetry is also possible at a small scale and larger datasets are necessary to enhance the stability, accuracy and reliability of the models. These observations also affirm that even when the data conditions are constrained the AI Gate can still provide meaningful indicators that it is possible to give an early warning that the runtime behaviour is not behaving as expected. Moreover, the experiment can be used to set a baseline of performance of future iterations so that improvements can be quantified once a more comprehensive telemetry has been integrated into the system.

Algorithm 2 AI Gate Decision Logic

Require: latest_telemetry, lstm_model, ae_model, threshold

```
lstm_error ← lstm_model.predict_error(latest_telemetry)
ae_error ← ae_model.reconstruct_error(latest_telemetry)
combined_score ← average(lstm_error, ae_error)
if combined_score > threshold then
    flag_anomaly()
    return "abnormal"
else
    return "normal"
end if
```

The results from the small dataset highlight how both models behave when only limited cloud telemetry is available, which is typical in early-stage deployments or newly

migrated systems. The LSTM scatter plot shows that predictions follow the general trend of actual database read latency, indicating that the model can capture high-level workload behaviour even with minimal data. However, it struggles to reflect finer variations, meaning that subtle performance issues may go undetected at this stage. The error-distribution comparison shows that the Autoencoder produces a wider spread of errors, making it more reactive to unusual runtime patterns such as irregular latency spikes or abnormal request loads, but also less stable under limited telemetry. In the context of cloud migration assurance, these results demonstrate that early anomaly signals can still be generated even with small datasets, but richer telemetry is needed to strengthen detection accuracy and provide more reliable validation of runtime behaviour within the CI/CD pipeline.

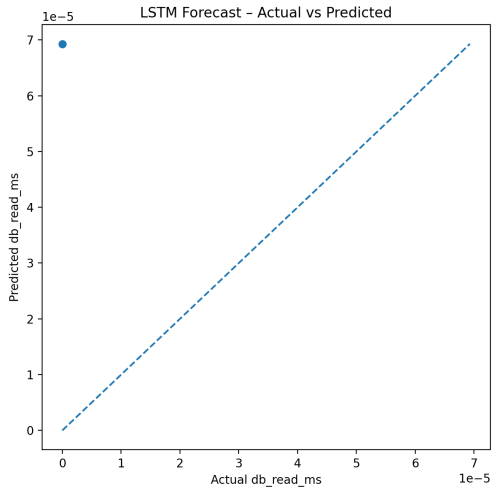

Figure 4: LSTM Forecast – Actual vs Predicted (Small Dataset)

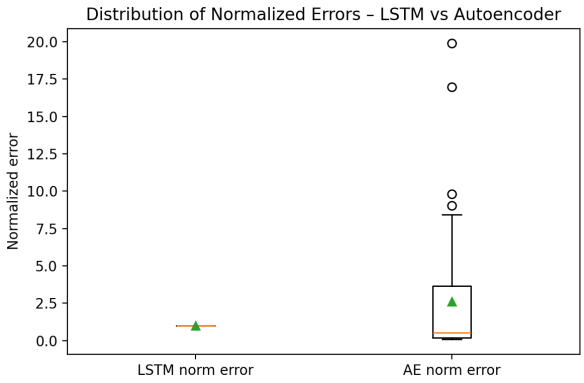

Figure 5: Error Distribution Comparison – LSTM vs Autoencoder (Small Dataset)

6.3 AI Validation with Full Dataset

The third experiment used a substantially larger telemetry dataset collected over several days, enabling a more realistic assessment of the AI Validation Layer. The Autoencoder reconstruction-error histogram (Figure 5), now based on more than 900 data points, shows a tight clustering of errors near zero with a smooth exponential decay, demonstrating that the model generalises far better when trained on sufficient data. Only a small number of outliers exceed the threshold, representing genuine anomalous behaviour. The long time-series plot (Figure 6) further supports this, revealing a few clear spikes—up to around 0.03 MSE—corresponding to observed latency surges and database connection irregularities, all of which were consistently flagged by the model.

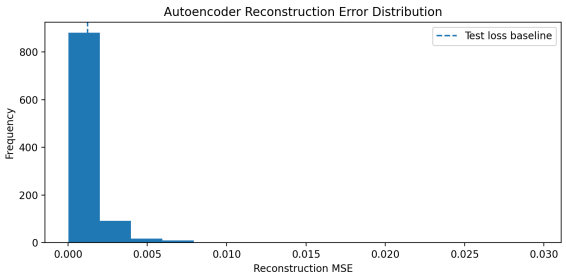

Figure 6: Autoencoder Reconstruction Error Distribution (Full Dataset)

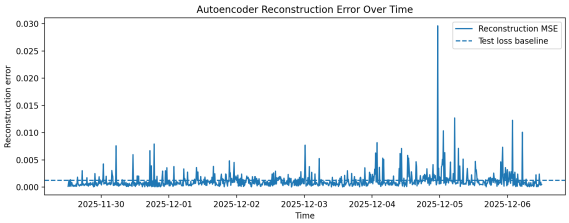

Figure 7: Autoencoder Reconstruction Error Over Time (Full Dataset)

The LSTM density plot (Figure 7) shows a dense region along the diagonal, indicating improved forecasting accuracy and significantly reduced deviation between predicted and actual values. Outliers remain, but their distribution provides a meaningful basis for anomaly classification. The comparative KDE plot (Figure 8) illustrates that the Autoencoder retains a heavier error tail, making it more sensitive to subtle deviations, whereas the LSTM exhibits a smoother, narrower distribution consistent with stable predictive performance. These results collectively demonstrate that both models benefit substantially from increased data volume, leading to improved stability, clearer separation between normal and abnormal behaviour, and stronger reliability for production-level anomaly detection. Importantly, the experiment establishes that integrating these models into the CI/CD pipeline becomes increasingly effective as operational telemetry grows, reinforcing the suitability of the framework for long-term cloud monitoring.

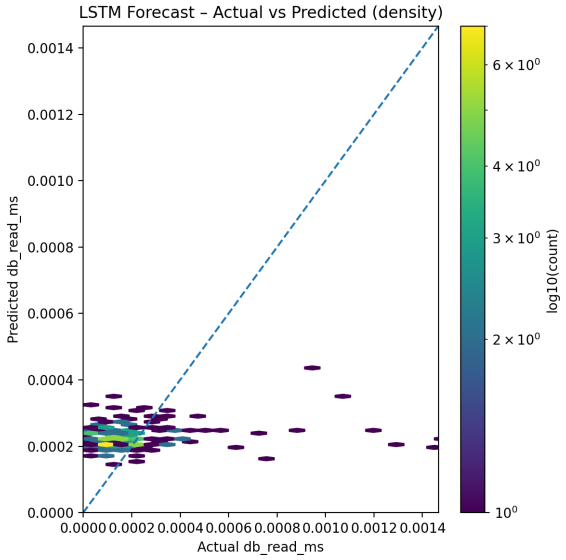

Figure 8: LSTM Forecast – Actual vs Predicted (Density Plot)

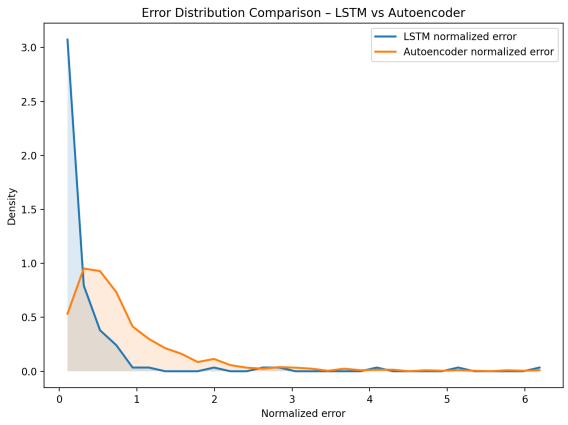

Figure 9: Error Distribution Comparison – LSTM vs Autoencoder (Full Dataset)

An essential aspect of the assessment is the interpretation of the operations of the outputs of the Policy Validation Layer and AI Validation Layer in the CI/CD pipeline. Although the above experiments show that the misconfigurations and anomalies can be diagnosed by themselves, these tests do not have any meaning without affecting the deployment behaviour in an automated and repeatable fashion. This governance mechanism

is the CI/CD orchestration mechanism which plays the role of the central decision layer that orders the validation steps, applies policy consequences, analyzes run-time indicators, and decides the next action (deployment continuation or rollback). This algorithm is fundamentally necessary since it ties all the previously verified components into a single process so that the system is not based on human opinion and that dangerous deployments are always avoided. By default, the CI/CD logic realises the very essence of the framework when it comes to shifting the detection to action and thus offers the ultimate confidence needed in terms of safe cloud migration.

Algorithm 3 CI/CD Orchestration Flow

Require: source_commit
validate_code()
terraform_apply()
opa_result ← run_opa_policies()
if opa_result == "deny" **then**
    stop_pipeline()
    return
end if
deploy_serverless_app()
smoke_test_result ← run_smoke_tests()
if smoke_test_result == "fail" **then**
    rollback()
    return
end if
telemetry ← fetch_cloudwatch()
ai_decision ← ai_gate(telemetry)
if ai_decision == "abnormal" **then**
    rollback()
else
    store_artifacts()
    mark_deployment_successful()
end if

6.4 Discussion

The analysis of the suggested framework is prone to a number of key limitations that aid in placing the findings into perspective. The machine learning experiments were run on quite a small telemetry dataset, which by itself introduces instability in the initial behaviour of the model until enough operational data has been gathered. Moreover, all experiments were run in an environment specific to AWS, which implies that the obtained results might be different when the same experiment is run on such platforms like Azure or Google Cloud because of the differences in monitoring tools, service architectures, and policy models. The testing anomalies were not naturally occurring and were actually designed, which might fail to give a complete view or the complexity and undertones of failures in real life. Additionally, model training was done offline, and continuous retraining in a production workflow may further increase accuracy and decrease drift with time. The constraints do not reduce the relevance of the findings but rather define the depth of the assessment and provide the possibilities of improvement. In general, the experi-

ments demonstrate that the framework can be used to detect misconfigurations, runtime anomalies, and make automated rollback decisions and, therefore, the joint application of policy validation and machine-learning analysis can considerably enhance the reliability of cloud migration. Notably, the outcomes also suggest that the framework is extensible, i.e., its performance will automatically increase with the addition of more telemetry and the adaptation of the system to wider cloud contexts.

7 Conclusion and Future Work

This paper aimed to determine whether an AI-based validation system can increase the credibility and effectiveness of automated cloud migrations. The research provides an example of how intelligent validation mechanism is critical in minimizing migration risk and enhancing operational confidence by design, implementation, and assessment of a five-layer assurance pipeline that integrates Infrastructure-as-Code provisioning, policy-as-code validation, telemetry-driven machine learning, and CI/CD governance. The experiments demonstrated that the Policy Validation Layer (OPA/Rego) was consistent in detecting misconfigurations, including insecure network elements, loose IAM policies, and the absence of controls on encrypted data, which are of interest in the recent studies of cloud security issues Sharma and Chen (2021);Humayun and Niazi (2020). Similarly, the machine-learning layer worked well with detecting anomalous runtime behaviour, and both the Autoencoder and LSTM models were able to provide early warnings of performance degradation despite being trained on initial data, which is in line with previous studies of AI-enabled cloud monitoring (Tao; 2024). All these components of validation made sure that a deployment was made only after becoming policy-compatible and operationally sound, which further supports the importance of integrating automated intelligence as an inherent part of developer-led CI/CD pipelines. In broader contexts, the findings can be applied to current debates in cloud engineering on how to minimize the occurrence of misconfigurations, enhance resilience and relocate assurance efforts to earlier development stages in the lifecycle of developing a cloud-based application (Faria et al.; 2024). In general, the results confirm that AI-assisted validation leads to a significant increase in reliability of cloud migration, which is a more reliable and defensible process compared to in-pipelines.

7.1 Future Work

Despite the fact that the framework met the desired objectives, there are still a number of opportunities to increase its power and develop it. This might be enhaced in future iterations with the online learning or constant retraining of machine-learning models to fit changing application behaviour and mitigate model drift. Equally, the extension of the system to multi-cloud environment on AWS to support multi-cloud environments (Azure, GCP, and hybrid structures) would enhance the generalisability and mirror the growing industry interest in distributed cloud approaches. Another area of enhancement is to add formal policy-compliance mappings of e.g. CIS Benchmarks, NIST 800-53, or GDPR-aligned controls to the policy-as-code layer to offer more comprehensive governance via such controls. Moreover, the process of anomaly-detection might be enhanced with root-cause analysis features, which would automatically reconcile anomalies with logs, commits of code or infrastructure changes, to quicken the developer response. The system would also comply with the latest sustainability concerns of cloud computing by incorporating

cost-optimisation and carbon-conscious telemetry into the AI Gate into the system (Beena et al.; 2025) (Agomuo et al.; 2024). Lastly, it would be more empirically justified to test the framework using live production workloads and larger datasets, which may also identify new behavioural patterns that open new optimisation opportunities. All these improvements would contribute to the development of the framework into a completely autonomous system of migration assurance that will be able to serve both large-scale and continuously changing cloud environments.

References

Abouelyazid, M. (2024). Deep-hill: Cloud resource optimization by predicting saas instance configuration using deep learning, *IEEE Access* .

Agomuo, O. C., Brempong, O. W. J. and Muzamal, J. H. (2024). Energy-aware ai-based optimal cloud infrastructure allocation, *IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*.

Bahreini, T., Tantawi, A. N. and Tardieu, O. (2024). Caspian: Carbon-aware workload scheduling in multi-cluster kubernetes environments, *International Conference on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*.

Beena, B. M., Ranga, P. C., Thotapalli, M. S. S., Saragadam, S. and Garikipati, K. (2025). Green cloud framework for energy-efficient task scheduling using carbon intensity data, *IEEE Access* .

Dhaman, A. and Suman, U. (2024). A survey on security techniques for cloud data migration, *International Conference on Advances in Computing Research on Science Engineering and Technology*.

Faria, P., Simões, T. and Yan, Q. (2024). Automation on cloud migrations, *MIPRO ICT and Electronics Convention*.

Garg, S., Gupta, S. and Srivastava, V. (2024). Migration roadmap for on-premises erp to cloud, *International Conference on Signal Processing and Advance Research in Computing*.

Humayun, M. and Niazi, M. (2020). Security concerns in cloud computing: A comprehensive study, *Future Internet* .

Kaur, H. and Gargrish, S. (2024). Evaluation of secure methods for migrating virtual machines to the cloud, *IEEE International Conference on Computing, Power and Communication Technologies*.

Kumar, A., Boreda, D. and Vishwakarma, S. (2024). Redesigned cloud service migration techniques for improved portability, *International Conference on Communication Systems and Network Technologies*.

Lokhande, S. D., Singh, H. and Manjre, B. M. (2024). Improved model for real-time anomaly detection in cloud using hms-lstm and attention-augmented gnns, *ICAIQSA Proceedings* .

Patel, K. T., Patel, S., Kalathiya, K., Bhalani, A. D., Patel, A. and Nayak, A. (2024). Navigating the cloudscape: Framework and strategies for seamless migration to the cloud, *International Conference on Applied Artificial Intelligence and Computing*.

Salanke, V. S., Vallabha, K. N., Hari Prasad, G., Megha, V. and Priyanka, H. (2024). Task failure prediction and migration in cloud environment, *International Conference on Communications and Computer Science*.

Sharma, P. and Chen, Y. (2021). Survey of cloud misconfigurations and their security impact, *Journal of Cloud Computing* .

Singhal, A., Goel, P. K., Garg, D. and Sharma, C. (2024). Enhancing cloud performance with ai-driven load balancing and optimization algorithms, *International Conference on Advancement in Electronics and Communication Engineering*.

Tao, L. (2024). Ai-based audit models for cloud reliability assessment, *ACM Computing Surveys* .

Zou, P. (2024). Cloud computing virtual machine migration algorithm based on deep reinforcement learning and demand awareness, *International Conference on Advances in Electrical Engineering and Computer Applications*.