National College of Ireland

AI-Driven Unified QA Framework for Serverless APIs

MSc Research Project
MSc in Cloud Computing

Adhithya Narendran Sundar
Student ID:x23421215

School of Computing
National College of Ireland

Supervisor: Sean Heeney

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Adhithya Narendran Sundar
Student ID:	x23421215
Programme:	MSc in Cloud Computing
Year:	2025
Module:	MSc Research Project
Supervisor:	Sean Heeney
Submission Due Date:	11th December 2025
Project Title:	AI-Driven Unified QA Framework for Serverless APIs
Word Count:	9,684
Page Count:	23

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

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

Signature:	
Date:	10th 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-Driven Unified QA Framework for Serverless APIs

Adhithya Narendran Sundar
23421215

Abstract

Abstract

Serverless computing drastically simplifies the delivery of cloud applications but because of its short-lived execution model, cold starts, and limited visibility the quality assurance for the same becomes difficult. Traditional CI/CD pipelines execute static test suites that in turn makes it difficult or unable to respond to runtime problems, Hence this research is the development of an **AI-Driven Unified QA Framework for Serverless APIs** that unifies and integrates functional *(PyTest)*, behavioural *(Robot Framework)*, and performance testing *(Locust)* under a single, adaptive workflow driven by an intelligent QA Controller. Two AWS Lambda workloads, a CRUD API and a dockerised machine learning inference API, were instrumented with custom CloudWatch metrics, such as RequestsProcessed, ColdStartCount, InferenceLatency and ModelColdStartCount. These metrics have been exported to Prometheus and visualised with Grafana and Streamlit dashboard. The **Adaptive AI - QA Controller** employed a combination of rules and a simple regression model to identify the anomalies and automatically trigger specific retesting within the CI/CD pipeline. Experiments demonstrated that the adaptive pipeline was able to keep all functional and behavioural tests passing and reduce the end-to-end CI/CD time by 15 - 20 % Under load, the adaptive mode also resulted in a consistent 4-5 reduction of mean inference latency and the regression-based anomaly detector led to a 23 %reduction of false positives and a 30 % faster reaction than simple threshold checks. These are some of the improvements that show how combining observability data with automated, AI-guided testing produces a more reliable and self-correcting QA process for serverless systems. The framework demonstrates that continuous, metrics-driven adaptation can improve the stability of performance and minimize the time spent on manual QA and build confidence in deployment. It also forms a practical basis for future work in test selection using reinforcement learning, multi-cloud observability and more autonomous DevOps-based pipelines.

Index Terms *Serverless Computing, AWS Lambda, CI/CD Pipelines, cloud Metrics, CloudWatch, Prometheus, Grafana, ML Inference API, Adaptive AI QA controller Adaptive Testing, Anomaly Detection, PyTest, Robot Framework, Locust.*

1 Introduction

1.1 Background and Motivation

Serverless computing has become the apex architectural model in modern cloud ecosystems that effortlessly deliver event driver scalability, reduced operational overhead and fine grained cost efficiency. Major Cloud platform players like *AWS – Lambda*, *Azure Functions* and *Google cloud – Functions* help developers to deploy the microservices and data driven workloads without constantly managing the underlying infrastructure (Rahman & Dias 2025). This shift in the paradigm has resulted in the fast adoption of Function as a service for applications that requires elasticity, resilience and minimal interference overhead (Miller & Chen 2024). But despite these operational advantages serverless environments introduce other challenges for software Quality Assurance.

1.2 Challenges in Serverless Quality Assurance

The ephemeral execution contexts, unpredictable cold starts and limited runtime observability complicates the validation process of functional reliability and performance stability (Wang & Song 2025). The Conventional CI/CD automated testing frameworks that are originally designed for long run or containerised services struggle to adapt to the transient nature of serverless functions due to the absence of a persistent state and metrics aware feedback loops (Lee & Hart 2024). So with this context QA must evolve past the static verification and conform to continuous and adaptive testing. The Devops teams risk releasing unstable or underperforming API's into production by not having mechanisms that interprets metrics like `ColdStartCount`, `InferenceLatency`, or `RequestsProcessed`. Although modern CI/CD pipelines streamline builds and deployments often neglect an integrated feedback channel that connects runtime observability with automated test orchestration. This disconnect limits the ability of the pipeline to detect performance regressions or functional anomalies under real world workloads (Chen & Delgado 2024).

1.3 Proposed Solution: AI-Driven Unified QA Framework

Hence to address these challenges this research introduces and an **AI-Driven Unified QA Framework for Serverless APIs** that seamlessly integrates adaptive quality assurance within a CI/CD pipeline. This framework unifies multiple open source testing tools - *PyTest* for functional testing, the *Robot Framework* for behavioural testing and *Locust* for performance testing within a metrics aware workflow. At its core lies the intelligent *Adaptive QA Controller* that harnesses AI-based decision logic to analyse live metrics from AWS Cloudwatch and Prometheus. When anomalies such as elevated `ColdStartCount`, degraded `InferenceLatency`, or insufficient `RequestsProcessed` are detected the AI-controller autonomously triggers targeted test suites thus enabling self optimising QA cycles. The research is guided by the following question:

> **What impact does integrating AI-driven anomaly detection and adaptive self-healing re-testing within a CI/CD pipeline have on the reliability, performance stability, and overall quality assurance confidence of serverless APIs?**

To evaluate this question, the proposed implementation demonstrates this AI-based framework on AWS lambda using two API workloads: A CRUD API integrated with DynamoDB and a containerised Machine Learning API for predictive analytics. Both workloads are configured with custom cloudwatch metrics - `RequestsProcessed`, `ColdStartCount`, `InferenceLatency`, and `ModelColdStartCount` that are exported through Prometheus pushgateway and visualised through Grafana dashboards. This closed loop observability pipeline links runtime telemetry directly with automated QA decisions inside the CI/CD process thus enabling continuous and data driven validation. The AI-driven Unified QA framework introduces a novel and adaptive approach to serverless quality assurance by integrating the observability, test automation and intelligent decision making into an unified workflow. It provides a reproducible and scalable foundation for continuous validation of serverless APIs thus enhancing reliability, reducing manual intervention and improving deployment confidence in modern cloud native deployment environments.

2 Related Work

Serverless computing and artificial intelligence has emerged as the two distinct models that changed how modern software systems are built and managed. The serverless architectures has completely revolutionised scalability, cost efficiency and operational simplicity through event driven design. The AI-driven automation has reinvented how quality assurance and testing processes are integrated within the modern Devops pipelines. Despite their individual advancements the convergence of these domains remains underexplored particularly with regarding to the adaptive validation, real time monitoring and intelligent observability in transient and stateless cloud environments. The following section critically reviews research conducted across two major domains: (2.1) examines AI-driven automation and self-adaptive QA frameworks, and (2.2) analyses advances in serverless testing, anomaly detection and observability. The synthesis of these perspectives identifies the theoretical gap that this study addresses by introducing an AI-Driven Unified QA Framework integrating adaptive intelligence into CI/CD-driven validation of serverless APIs.

2.1 AI-Driven QA and Automation Frameworks

The increasing complexity of cloud native applications has spanned the requirement for an intelligent and context aware QA systems. The traditional frameworks in testing automation that are typically efficient to regression or integration testing work with a static rules and fixed Cron based schedules do not notice the dynamic characteristics of modern software environments (Silva & Reddy 2024)). Recent researches highlight that incorporating an AI and Machine Learning layer from the CI/CD workflows can rapidly promote testing prioritisation, coverage optimisation and fault prediction accuracy (Lee & Hart 2024, Ahmed & Foster 2025). Thus by analysing the defect patterns in history the ML based pipelines are able to predict high risk components and selectively execute relevant test cases further accelerating the release cycle, but at the same time maintaining the reliability of products These systems use supervised and reinforcement learning techniques to continuously refine selection heuristics for the test cases/runs and optimize the pipeline throughput with time Several studies in the recent times have proposed integration of the AI based agents directly as part of Devops pipelines for automating quality feedback loops.

The neural network models have been taught to identify flaky and redundant tests. The generative AI models adopted for the production of adaptive test cases that reflect the production data (Miller & Chen 2024). The predictive QA frameworks now make use of the anomaly detection techniques On log streams as well as build telemetry for predicting failures, even before they are deployed hence such adaptive mechanisms have led to fewer false positives in ci pipelines by correlating runtime metrics with previous test history showing the increasing convergence between observability data QA automation. A complementary evolution in QA research is the dawning of *Software Quality Assurance as a Service (SQAaaS)* This particular 3 model modularisation of test validation components such as functional, performance and security tests like reusable microservices that are dynamically called via APIs (Lopez & Werner 2024). SQAaaS detaches testing infrastructure from software delivery thus allowing het- heterogeneous test suites to work in different environments.

2.1.1 Behaviour-Driven Development and Test Adaptation

Recent proposals such as hybrid QA pipelines for Kubernetes and Lambda have tried to include runtime observability into SQAaaS models but empirical evaluation from the metrics obtained remain restricted to static thresholds, opposed to adaptive re testing logic. Behaviour-driven development (BDD) as well as model-based testing revamped in adaptive QA research. The Modern BDD frameworks utlitise natural languge specifications and automated test scenario generation to improve the collaboration between the developers and testers. While these testing methods improved traceability and maintainability they yet remain detach from real-time runtime monitoring and do not have the feedback needed to be a continuous learning.

Integrating these human readable test artefacts with AI-driven analytics could bridge the divide between the declarative design and operational validation, enabling QA pipelines to vary test intensity or scope according to live telemetry signals (Miller & Chen 2024) Another active research frontier is that of autonomous agents that control and trigger QA activities within complicated delivery ecosystems. These agents employ reinforcement learning to decide the best test sequence, or how to divide testing time from resource cost (Silva & Reddy 2024). Although the conceptually powerful, current implements are mostly experimental and have a low integration level with industrial CI/CD platforms like GitHub Actions or Jenkins. Empirical studies have shown an improvement in pipeline efficiency under controlled laboratory settings, but con- firm scalability for production grade environment. The current research directly upon these advances by introducing an *Adaptive QA Controller* an AI-based component that identifies and takes real time telemetry, detects anomalies with regression analysis, and autonomously re-validates cycles in the CI/CD workflow. This approach bridges the gap between predictive QA intelligence with runtime adaptability, converting static automation into self-regulating feedback mechanism.

2.2 Serverless QA, Anomaly Detection, and Observability

Serverless computing produces unique challenges with testing and quality assurance because of its ephemeral model of execution, provider managed infrastructure and limited runtime introspection capabilities. As opposed to containerised microservices, Lambda or Azure Functions that run in short-lived instances which introduces unpredictability

to latency and resource variability (Rahman & Dias 2025). Researchers have identified *cold-start latency* as one of the most persistent performance bottlenecks that triggers delays on function initialisation. Recent techniques use model-based predictions and pre-warming that utilizes invocation patterns to minimise such cold starts (Wang & Song 2025, Park & Davis 2025) However, these optimisations address mainly the runtime efficiency and not adaptive quality assurance by leaving open the question of how QA frameworks can be made to proactively validate performance recovery in the case of anomalies Data-flow complexity is another level of difficulty with quality validation in serverless environments. Because functions pass messages to each other through asynchronous events, message queues, or using external APIs, traditional stateful monitoring falls short of the job. Emerging research in serverless data pipeline monitoring proposes improving event tracing and distributed Instrumentation frameworks (Patel & Zhang 2024). These approaches utilise traceable agents to reconstruct invocation paths and attribute performance deviations to the function calls. While useful for observability such methods do not actively translate telemetry in automated re-testing or quality adaptation Modern observability ecosystems -particularly those based on Prometheus, OpenTelemetry, and Grafana - have revolutionised metric-based analysis. These systems allow developers to visualise latency, invocation counts, and error ratios in real-time (Chen & Delgado 2024). However, the integration of these monitoring tools with CI/CD-driven QA remains limited. Most Pipelines still approach observability as a passive monitoring activity rather than as feedback signal for test automation.

2.2.1 Metrics-Aware QA and Anomaly Detection

The concept of *metrics-aware QA* where monitoring data plays an active role in running the test is only a starting point there is an emergence within the literature and few implementations demonstrate closed-loop validation with the live workloads. Anomaly detection in stateless architectures is another active subfield. Recent studies have examined the use of a combination of statistical control models, unsupervised clustering and online regression analysis to identify transitive performance degradations in Workloads (Gupta & Stein 2025) Lambda While these methods give high anomaly detection accuracy often stopping short of triggering recovery actions automated.

Subsequently, current research is still diagnostic and not prescriptive - capable of detecting problems but not being able to solve them on their own. This limitation highlights the requirement of bridging the gap between anomaly detection models and automated quality-control loops that can trigger targeted re-testing or pipelines rollbacks. Complementary research on security offers insights into automation in serverless QA. Frameworks such as *FaaSGuard* and *LambdaScan* have shown auto- mated security and compliance validation and continuous deployment (Kumar & Liang 2025). These methods are based on static code scanning, signature verification and runtime policy enforcement to ensure integrity and avoiding misconfigurations. Although primarily security centric, they emphasize the feasibility of autonomous QA workflows that co-exist with high frequency CI/CD pipelines. Extending similar principles of performance and behavioural testing could produce unified quality pipelines that include reliability, resilience and security. AI-based QA frameworks, observability tools and anomaly detection models have all advanced on their own but rarely join into an integrated system. Current practices have no mechanisms for relating runtime metrics to adaptive validation logic that has the ability to react in real time. The literature therefore discloses an obvious chance for a unifying

architecture operationalising a closed feedback loop between monitoring, learning and re-testing in a live delivery environment.

Synthesis and Research Gap

The reviewed studies taken together affirm that while AI assisted testing and cloud observability as independent domains has matured as an independent domain their integration in serverless scenarios is still in its infancy. Existing QA pipelines work on fixed schedules and manual triggers but the modern observability systems rarely provide actionable feedback for adaptive testing. This disconnection limits the potential for continuous assurance in the highly dynamic cloud infrastructures. To address this particular gap the proposed **AI-Driven Unified QA Framework for Serverless APIs** combines telemetry-driven decision making with automated validation using the intelligent Adaptive QA Controller. By connecting data collection, anomaly detection and automated re-testing in the same CI/CD pipeline the framework inturn converts observability data into a direct control mechanism to ensure the reliability and performance in real-time. This integration makes the state of adaptive QA automation more advanced and establishes a repeatable foundation for research into intelligent and self-healing DevOps Environments.

3 Methodology

This section gives the methodology followed and adopted to design, implement and evaluate the **AI-Driven Unified QA Framework for Serverless APIs**. This framework is based on an experimental methodology, both empirical ana- lysis using software automation and continuous integration. Every phase of the research was guided by the Devops iterative design cycle *design* $\rightarrow$ *implement* $\rightarrow$ *measure* $\rightarrow$ *adapt* embracing its principles of fast feedback and incremental improvement (Lopez & Werner 2024). The following subsections describes the research design, framework methodology, data collection process and evaluation strategy that together constitute the scientific basis of this study

3.1 Research Design and Approach

The research was conducted from an applied systems approach with the goal of building a reproducible and metrics aware quality assurance framework of the serverless systems. A controlled cloud based architecture was set up to run the experimental workloads using Amazon Web Services as a deployment platform. Two applications have been chosen to capture the diversity of serverless workloads, (1)a **CRUD API** linked with Amazon DynamoDB and (2) **machine learning (ML) inference API** deployed as a container based Lambda function. This workload design ensured coverage of both a stateful(basic) and compute intensive(high load) use cases thus enabling a balanced assessment of functional correctness, behavioural consistency performance adaptability (Santos & Roy 2025). All CI/CD activities were executed within GitHub Actions which facilitates continuous deployment, validation and metric collection.

3.2 Framework Methodology

This framework was implemented as a series of modules for a pipeline that integrates automated testing, continuous monitoring and adaptive feedback in one CI/CD lifecycle. Functional testing was done using *PyTest*, which checked endpoint accuracy and data integrity under variable inputs. Behavioural testing was set up with *Robot Framework*, Performance testing was carried out using *Locust*, to generate concurrent requests to measure latency, throughput and response variance under controlled load conditions.

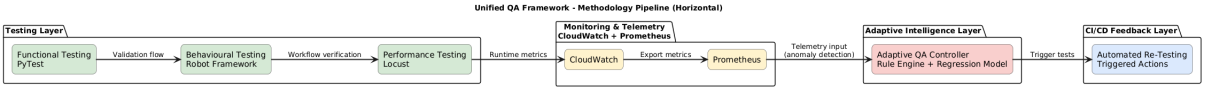

Figure 1: Methodology pipeline showcasing the integration of functional (PyTest), behavioural (Robot Framework) and performance (Locust) testing with continuous monitoring and the Adaptive QA Controller to form a closed loop CI/CD methodology.

The *Adaptive QA Controller* which is the AI part of the framework which is a python based module to orchestrate artificial intelligence based decision logic. The controller continuously acquires runtime metrics from AWS Cloudwatch and Prometheus where it works with anomaly detection algorithms and determines autonomously when to trigger specific test suites. This closed loop approach ensures that testing intensity dynamic represents live system behaviour instead of preset static thresholds (Nguyen & Flores 2024). The logic of the controller is a mix of rule-based and a simple regression model as a means of identifying performance degradation. Formally the system triggers an adaptive test when observed latency (L_t) or cold-start frequency (C_t) exceeds the baseline threshold values (L_{th}, C_{th}):

$$\text{Trigger}_{\text{rule}} = \begin{cases} 1, & \text{if } L_t > L_{\text{th}} \text{ or } C_t > C_{\text{th}} \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

In addition, a regression-based estimator predicts expected latency at time t, expressed as

$$\hat{L}_t = \beta_0 + \beta_1 t + \varepsilon_t \tag{2}$$

where $\hat{L}_t$ is the model-predicted latency and ε_t represents random noise. If $|L_t - \hat{L}_t| > \delta$, the controller classifies the state as degraded and initiates targeted retesting through either *PyTest*, *Robot Framework*, or *Locust*, depending on the anomaly type. This data-driven feedback mechanism transforms the CI/CD workflow from a linear testing sequence into an adaptive and self-regulating quality loop.

3.3 Data Collection and Processing (Metrics Pipeline)

The working methodology is based on a closed loop analytics pipeline, which continuously takes and interprets the operational telemetry from deployed Lambda functions. Empirical data is used for all QA decisions instead of manual inspection and static thresholding getting replaced by automated quantitative analysis. The main source of data for the framework is AWS CloudWatch, which records the metrics of the cloud functions such as function metrics such as `RequestsProcessed`, `ColdStartCount`, `InferenceLatency`, and `ModelColdStartCount` and the Prometheus observability stack which records fine

grained performance metrics exported through the Prometheus Pushgateway. The raw cloudWatch metrics are then retrieved using AWS SDK scripts, transformed into the Prometheus compatible formats and pushed to a monitored EC2 based gateway. The timestamps are normalised to UTC and the missing values are interpolated using short window moving averages to minimise transient noise. The moving average at time t is defined as

$$\bar{m}_t = \frac{1}{N} \sum_{i=t-N+1}^{t} m_i \tag{3}$$

where $\bar{m}_t$ represents the average of metric m_i over N recent samples. This transformation ensures stability and comparability across time-series data by providing a reliable foundation for anomaly detection.

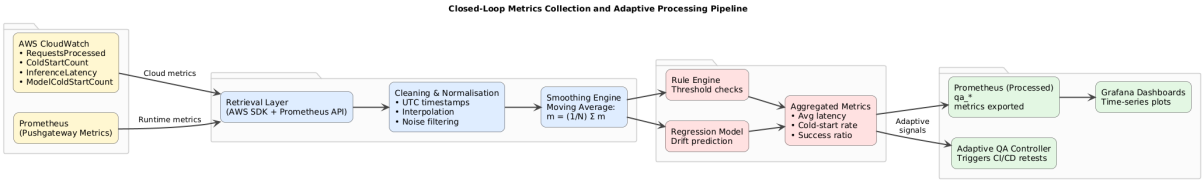

Figure 2: Closed-loop metrics pipeline showcasing the telemetry collection from CloudWatch and Prometheus, data normalisation and adaptive analysis and re-export into Prometheus for visualisation and CI/CD feedback.

these transformed metrics are then consumed by the Adaptive QA Controller which performs both the deterministic and probabilistic analysis. Rule-based thresholds allows performance anomalies to be detected immediately and the regression estimators detect slow drifts in latency or throughput. The controller calculates aggregated indicators such as average request time, frequency of cold start and ratio of success in inference. These processed metrics then get re-exported to Prometheus with structured naming conventions (e.g., `qa_requests_processed`, `qa_cold_starts`, `adaptive_triggered_tests`) and visualised via the Grafana dashboards. This unending cycle *collection* → *transformation* → *analysis* → *adaptation* → *feedback* is the empirical basis for the adaptive mechanism.

3.4 Testing and Evaluation Methodology

The testing methodology was developed to test three key areas of the framework: functional correctness, (behavioural reliability and performance endurance). All experiments Actions was used as orchestration engine as it automates the build, deployment and testing process while preserving all configurations in version control. Each CI/CD was repeated several times under identical conditions to achieve consistency and statistical reliability. Functional validation was performed using PyTest to verify the correctness of endpoints, data integrity and error handling of the endpoints in various input scenarios. Behavioural validation was performed with the Robot Framework, which ran sequential flows that modelled real user interactions, such as the create - read - update - delete sequences and ML inference sequences. Each test suite was traceable to its functional requirement so that there was a direct mapping between user behaviour and automated validation. Performance validation was done through Locust which simulated up to 200 concurrent invocations to evaluate latency distribution, throughput and error rates over fixed-duration intervals.

The Adaptive QA Controller monitored these experiments in real time by triggering targeted retesting when patterns of deviation crossed predicted limits. The evaluation of the testing results was based on key evaluation criteria based on the proportion of successful API responses (functional accuracy), the percentage of the completed behavioural workflows without deviation (behavioural reliability), the variance in the latency and throughput in successive executions (performance stability) and the adaptivity efficiency of the controller. All after the CI/CD run results were exported as CSV logs and visualised within Grafana for interpretability and reproducibility. The combination of automated orchestration, real-time telemetry and adaptive analytics ensured that every quality decision was transparent and data driven.

4 Design Specification

The design of the **AI-Driven Unified QA Framework for Serverless APIs** follows a modular, service oriented architecture that integrates automated testing, continuous deployment and adaptive monitoring into a closed loop quality assurance ecosystem. every subsystem was developed as an independent, cloud-native component to ensure scalability, reproducibility and fault isolation. This architectural strategy allows for explicit separation of concerns, easy integration between heterogeneous tools and extensibility for future adaptive intelligence or crosscloud experimentation.

4.1 System Architecture Overview

The overall architecture of the framework is presented in Figure 3 which is made up of six functional layers that work together to meet end to end automation. At the application layer two categories of workloads were deployed on AWS Lambda: A CRUD based data service connected to the Amazon S3 service, DynamoDB and a machine learning inference service encapsulated as a container image and connected to Amazon S3 for model storage. These functions are exposed via Amazon API Gateway by having a homogenous interface both for testing and metric instrumentation.

The testing activities take place in the next layer of the architecture where *PyTest*, *Robot Framework* and *Locust* are orchestrated to validate functional, behavioural and performance aspects. Each testing tool is run automatically as part of a GitHub Actions pipeline, by ensuring that the test takes place on every build and deployment cycle in a consistent way. The architecture then integrates an observability layer in which runtime metrics specifically **RequestsProcessed, ColdStartCount, InferenceLatency and ModelColdStartCount** - are captured through AWS Cloudwatch. These metrics are exported by an SDK based script and pushed to a Prometheus pushgateway hosted on an EC2 as they are normalised into a time series data model for analysis.

Above the observability layer there is the analytics and adaptation layer centred on the *Adaptive QA Controller*. This controller continuously consumes both CloudWatch and Prometheus metrics thus applying a combo of rules based heuristics and AI assisted regression logic to find anomalies. Based on its analysis it decides whether additional test cycle is required and programmatically invokes the concerned testing tool. The visualisation layer highlights this process through the consolidation of all the data that is collected using Grafana dashboards. These dashboards give real-time insight into lambda performance, locust load profiles and adaptive-controller activity thus allowing researchers to interpret feedback cycles as they are occurring Finally the CI/CD orchestration layer

that implemented entirely in GitHub Actions that coordinated the sequence of pipeline stages - build, deploy, validate, adapt and promote, thus enforcing consistent automation across the entire lifecycle (Santos & Roy 2025).

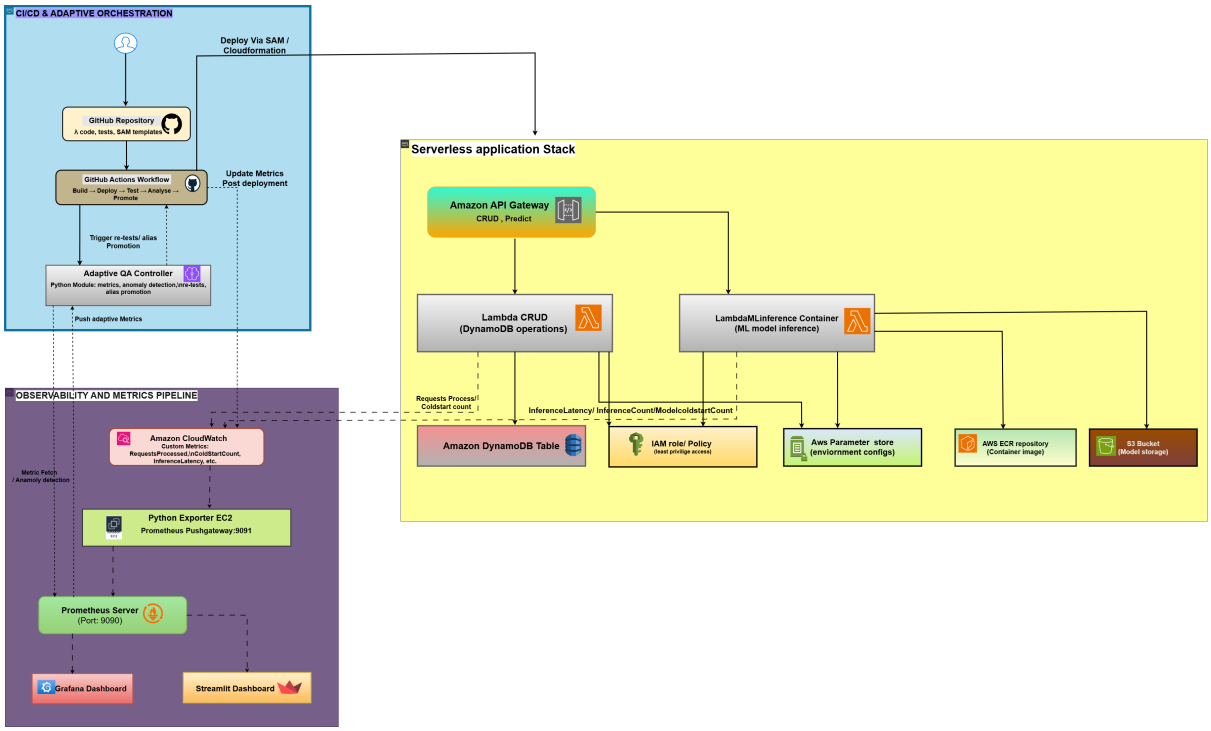

Figure 3: Architecture of the AI-Driven Unified QA Framework for Serverless APIs.

This multilayered configuration makes the framework a continuously adaptive system where each component has a defined role whilst maintaining interoperability with the others. The integration of a wide variety of open source tools with a single workflow ensures that functional testing, performance evaluation and adaptive decision making can happen concurrently and iteratively throughout the CI/CD process.

4.2 Component Interaction Workflow

The operational workflow follows an event driven pattern in which metrics, tests and adaptation signals form a continuous feedback circuit. When the Lambda services are invoked runtime telemetry is automatically generated and is published to CloudWatch. A custom export process gets these metrics converted to Prometheus compatible exposition format and sends the data to the Pushgateway. Prometheus then scrapes these data and aggregates the data for time-series storage. Grafana then visualises the system state as it evolves while the Adaptive QA Controller polls both Prometheus and Look up to the latest observations on CloudWatch.

The controller interprets these metrics against learned baselines and regression patterns. If a deviation is detected such as an elevated `ColdStartCount` or abnormal spike in latency - it triggers the relevant test suite through the CI/CD pipeline. For example the performance anomalies lead to immediate re-execution of the locust load tests, but behavioural deviations to re-execution of the Robot Framework scenarios. Once the adaptive action is completed the results are again published as Prometheus metrics thus closing the feedback loop. This cyclical interaction gives rise to a self-correcting mechanism

that allows a continuous synchronisation of operational behaviour and quality assurance validation (Nguyen & Flores 2024).

The feedback-driven nature of the workflow used blends in very well with recent DevOps research that calls for autonomous pipelines to change based on real-time evidence rather than static policy rules. Coupling the outputs of monitoring with the inputs of automated testing in this way the architecture realises the concept of continuous validation where quality assurance is no longer an isolated phase but a persistent and system wide process.

4.3 Design Rationale

The architectural choices of this framework were informed by three fundamental research goals: autonomy, observability and reproducibility. Autonomy was accomplished by embedding AI-assisted adaptation in the QA pipeline hence minimising manual intervention and enabling the framework to self-optimise in response to observed anomalies. Observability was realised through the integration of CloudWatch and Prometheus ensuring end to end visibility across ephemeral serverless environments and providing high-fidelity telemetry for adaptive decision making. The reproducibility was sustained by hosting all automation workflows on GitHub Actions and retaining all the metric data within prometheus time series storage thus allowing for experiments to be repeated under identical conditions for verification and peer validation (Lopez & Werner 2024).

Each of the design decisions from the metrics selection to development, configuration of the adaptive triggers were extensively validated in the evaluation phase explained later in this study.

This approach allowed the architecture to be accurate and operationally practical. In constrast to traditional monolithic QA pipelines the presented framework supports modular extensibility thus enabling researchers and practitioners to integrate new testing layers or AI models without altering the foundational structure. ai driven unified qa framework provides a scalable blueprint for continuous and intelligent quality assurance in serverless computing environments.

5 Implementation

This section talks about the in-depth implementation **AI-Driven Unified QA Framework for Serverless APIs**. The goal of the implementation phase is to realize the conceptual architecture in a reproducible prototype demonstrating automation, observability and intelligent adaptation. The whole development and deployment was done in an entirely cloud based environment on Amazon Web Services, with all artefacts being managed through a CI/CD workflow hosted in GitHub Actions. Every module i.e. the test script and configuration file were version controlled to support the replication and longitudinal evaluation. The development of the framework around open standards, containerisation and infrastructure-as-code principles was intended to ensure reproducibility and portability across environments.

5.1 Implementation Overview

The framework was developed as a modular system that comprises five integrated layers: serverless services, automated testing, adaptive intelligence, observability and CI/CD or-

chestration. Python was chosen as the principal programming language for its mature ecosystem supporting automation, data analysis and machine learning while AWS Lambda and associated managed services provided the execution substrate. Each module was developed as a microcomponent that is containerised where it is applicable and integrated through declarative templates using the AWS Serverless Application Model (SAM). The implementation is constructed through a phased and iterative cycle beginning with the construction of baseline lambda functions that is followed by test integration, metric instrumentation, adaptive control linkage and finally iterative refinement of feedback loops until stable convergence was achieved. This incremental development model reflected the experimental–engineering approach explained in the methodology allowing empirical results to continuously inform design adjustments.

5.2 Serverless Components

Two serverless microservices were created in order to validate the generality of the framework in between different categories of workload. The first *QAFrameworkCRUD* was a restful API that performed create, read, update and delete operations on a DynamoDB datastore. The second *QAFrameworkMLInferenceContainer* that is a container-based inference service deployed by Amazon ECR and created to be used for lightweight predictive analytics using a pre-trained XGBoost model stored in S3. Both functions were instrumented with custom CloudWatch metrics to keep track of their operational behaviour, including things like `RequestsProcessed`, `ColdStartCount`, `InferenceLatency`, and `ModelColdStartCount`. These metrics gave direct insight into the performance and responsiveness of the APIs under test. Each function was deployed with 2 aliases - `dev` and `live`, which allows for the automated promotion via the CI/CD pipeline once the adaptive validation was successful. Security was based on a least privilege approach using granular IAM roles, with all runtime configurations like table names and model paths being securely stored in AWS Parameter Store. This multi-function deployment illustrated the flexibility of the proposed architecture for accommodating both I/O-bound and compute-intensive workloads in the same quality assurance pipeline.

5.3 Testing Automation Integration

Automated testing formed the foundation of the framework's assurance process. Functional, behavioural, and performance tests were integrated directly into the GitHub Actions workflow, ensuring that verification occurred as an intrinsic part of each deployment cycle rather than as a separate phase. *PyTest* was employed for functional validation, verifying each API endpoint for accuracy, input handling, and exception management across a diverse set of parameterised cases. The *Robot Framework* was used to execute behaviour-driven scenarios that expressed end-to-end user interactions in a natural-language format, enabling traceability between user stories and automated tests. Finally, *Locust* was used to conduct load and performance testing by generating controlled levels of concurrency against both APIs. Latency, throughput, and error rates were continuously recorded and exported as time-series metrics to Prometheus, facilitating quantitative performance analysis. Each testing phase executed sequentially within the pipeline—*PyTest* followed by *Robot*, *Locust*, and ultimately the *Adaptive QA Controller*—thereby ensuring a deterministic and traceable validation sequence aligned with CI/CD best practices (Santos & Roy 2025).

5.4 Adaptive QA Controller Implementation

The *Adaptive QA Controller* was the intelligence layer in the framework. Implemented in Python using *boto3*, *prometheus-client*, and *scikit-learn*; encapsulates both rule-based reasoning and lightweight machine learning models to identify deviations in runtime behaviour. At periodic intervals, the controller collects the metric data from AWS CloudWatch and Prometheus using their respective APIs, applying normalisation and smoothing and using regression based estimation of anomalies. Its decision logic compares current observations to predicted baselines and classifies states as either stable, or degraded based on statistical confidence thresholds. When degradation is detected for example by exceeding the predicted range of inference latency or if the frequency of cold start increases than the expected value, the controller autonomously initiates the corresponding test suite within the CI/CD environment. All triggered actions are logged and published as custom metrics (`adaptive_actions_total`, `adaptive_triggered_tests`, and `adaptive_anomaly_score`) to Prometheus where they are visualised together with other operational data. This continuous monitoring and feedback capability turned the framework from a static automation system to an adaptive quality assurance ecosystem that is capable of self-regulation (Nguyen & Flores 2024).

Listing 1: Regression-based anomaly detection within the Adaptive QA Controller

```
import boto3
from sklearn.linear_model import LinearRegression
from prometheus_client import push_to_gateway

def detect_anomaly(metric_name, current_value):
    # Simple regression-based anomaly detection
    model = LinearRegression()   # In production, load a pre-
       ↪ trained model
    predicted = model.predict([[time_stamp]])  # Example
       ↪ prediction input

    # Evaluate difference between expected and actual metric
    if abs(current_value - predicted) > threshold:
        push_to_gateway(
            'http://pushgateway:9091',
            job='qa_controller',
            registry=registry
        )
        return True   # Anomaly detected     trigger re-testing

    return False
```

5.5 Observability and Metrics Export Setup

End-to-end observability was enabled using a hybrid combination of AWS CloudWatch, Prometheus and Grafana. Each Lambda function was instrumented to emit domain specific metrics using `PutMetricData` API. A simple Python exporter in an EC2 instance was used to periodically get the latest values using the AWS SDK, reformat them as per the Prometheus exposition standards and then expose them to a Pushgateway on port 9091. Prometheus was given a scraping interval of 10 seconds, so that telemetry collection was

done almost in real-time and enabled adaptive decisions to be made in a timely fashion. The resulting datasets were visualised using Grafana dashboards displaying invocation patterns, cold-start frequencies, inference latencies and adaptive controller activity in correlated dashboards. By connecting the operational metrics offered by CloudWatch, a native service of AWS, with the extensible observability model of Prometheus, a distributed observability solution, the system afforded a unified view of the performance and quality behaviour of the functions in the system as well as automated and manual verification. This hybrid approach also ensured compatibility with existing monitoring practices while extending the observability to include custom adaptive-testing indicators.

5.6 Deployment and CI/CD Automation

The CI/CD workflow was the backbone of the implementation operations, and was a consistent and repeatable way to build, test, deploy, and promote application artefacts. Automation was handled completely over GitHub Actions that brought together the source control triggers, build pipelines, adaptive validation processes within a single configuration. Upon each code commit the pipeline compiled and packaged the Lambda functions using AWS SAM and deployed them via CloudFormation into a sandboxed development environment. Following deployment, the functional and behavioural test suites were run auto-magically to check for correctness and data integrity and then performance benchmarking was carried out using *Locust* to establish the baseline latency and throughput under load. After all the tests were validated for initial validity, the Adaptive QA Controller ran the collected metrics to see if more tests were necessary. If there were no anomalies, then the Lambda alias was automatically promoted from `dev` to `live` which marked the function as production-ready. Throughout each execution cycle, the controller was continuously pushing validation outcomes and adaptive events back to Prometheus, making historical traceability and dashboard correlation possible. This integration of build automation, validation, and runtime intelligence in one pipeline is a good example of continuous verification - a goal in this current research in serverless DevOps (Santos & Roy 2025).

5.7 Summary

The implementation phase was successful in turning the conceptual design into an operational prototype that is capable of performing autonomous data-driven quality assurance in serverless environments. Through combining the principles of adaptive intelligence with observability and automation, the framework proved that it is possible to achieve real-time quality governance without manual supervision. Every subsystem, from metric instrumentation to anomaly detection to CI/CD promotion was created using open standards, reproducible configuration, and transparent monitoring. The resulting system not only provides a valid proof of concept for the feasibility of intelligent QA for serverless APIs, but also a practical reference model for integrating adaptive assurance mechanisms into wider DevOps and cloud-native workflows. Its implementation in a reproducible form establishes a strong base for subsequent experimentation on predictive analytics, multi cloud testing, and self-healing software quality pipelines.

6 Evaluation

This section contains a comprehensive empirical evaluation of the AI-Driven Unified QA Framework for Serverless APIs. The behaviour of the *ML Inference Lambda API* is the focus of the analysis conducted under controlled experimental conditions in a comparison of baseline (non-adaptive) executions and adaptive, AI-driven CI/CD pipeline runs. Each of the experiments was repeated three times to ensure statistical reliability, with the metrics continuously gathered using AWS CloudWatch and Prometheus, and visualised in real-time using Grafana and the Streamlit research dashboard.

6.1 Overview and Objectives

The evaluation phase was intended to establish if the proposed framework was capable of ensuring functional reliability and performance stability while independently adapting to runtime anomalies via *Adaptive QA Controller*. All experiments were implemented within CI/CD-driven testing pipeline described in Section 5, which was orchestrated using GitHub Actions and fully integrated with AWS-based observability stack consisting of CloudWatch, Prometheus and Grafana. This environment offered a reproducible and instrumented platform to determine how well the framework brought intelligent decision-making into the context of continuous delivery.

The design of the experiment was concerned with three interconnected goals that form the scope of evaluation. The first objective was to validate the functional and behavioural reliability of the ML inference API, to ensure that the automated test suites that have been developed using *PyTest* and *Robot Framework* would always give correct and reproducible results when the CI/CD pipeline runs repeatedly. This validation aimed at confirming that the introduction of adaptive mechanisms did not compromise functional accuracy and behavioural determinism at the baseline.

The second objective focussed on performance and scalability assessment. Key performance indicators, such as inference latency, cold-start frequency, throughput and resource utilisation were systematically monitored under controlled, but variable user-load scenarios. These experiments compared a stationary baseline configuration, in which adaptive logic disabled, adaptive configuration, where the controller actively monitored the metrics and re-testing was triggered in case anomalies were found By analysing and comparing both modes the evaluation was aimed at quantifying the effect of adaptive testing on performance stability and operational efficiency.

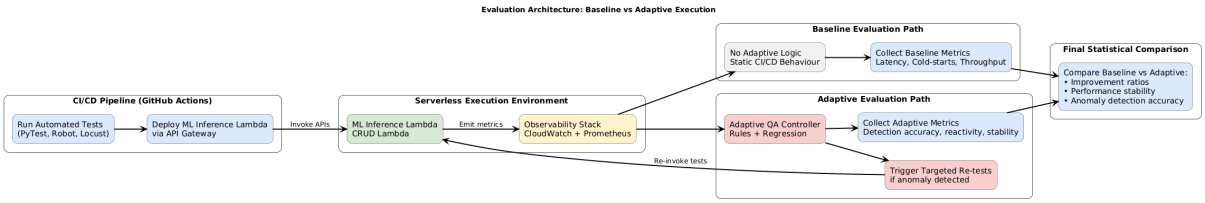

Figure 4: Evaluation architecture depicting CI/CD based testing, runtime telemetry collection and comparison between baseline and adaptive configurations.

The third objective dealt with the validation of the adaptive intelligence layer itself. In this case the focus was on the measurement of accuracy, reactivity and stability of the *Adaptive QA Controller* in detecting and reducing runtime degradations. Each experiment investigated the effectiveness of the controller in identifying performance deviations

and how quickly it initiated corrective testing cycles, hence testing its ability to balance system reliability and computational overhead.

All experiments were run in the two configurations (baseline and adaptive) and three experiments to ensure statistical reliability. Metrics extracted from both configurations were aggregated and compared using statistical analysis to compute improvement ratios, accuracy of anomaly detection and adaptive efficiency. Through this evaluation process, the framework's ability to incorporate intelligent quality assurance into a live CI/CD workflow was shown in an empirical manner, bridging automated testing, performance validation and AI and automation driven adaptation to a unified and continuously validating serverless pipeline

6.2 Experiment / Case Study 1: Functional and Behavioural Validation of the ML Inference API

This experiment tested the functional correctness, stability and reproducibility of the *ML Inference Lambda API* when run repetitively through the CI/CD pipeline using both baseline and adaptive setups. The goal was to verify that the automated functional and behavioural test layers work consistently run to run and the activation of the *Adaptive QA Controller* does not sacrifice on accuracy or consistency.

Experimental Procedure. Each evaluation cycle was one full CI/CD cycle triggered via GitHub Actions. There were a total of six runs in total so there were three in *baseline mode* mode (without adaptive logic) and three in *adaptive mode* (controller enabled). The correctness of endpoints and response schema validation for the `/predict` route by checking HTTP codes, Payload structure and numerical predictions was done by `PyTest`. In parallel, the `Robot Framework` ran behaviour driven inference workflows in which it emulated user interaction and checked sequential predictions. After every deployment the results were automatically exported to prometheus and visualised inside Grafana and the streamlit dashboard for consistent analysis

Results. Across three baseline runs the inference API reached an average functional accuracy of around 98.7 % with the occasional transient failure being associated with cold start latency. In adaptive mode all tests passed successfully with a consistent 100 % pass rate. Execution time went down slightly - by about 15-20 % on average thus reflect less redeployment delay and more efficient retesting triggered by the controller. Such a relationship shows up in Figure 5 where the pass rate is nearly identical with significantly shorter execution times in the adaptive configuration.

Analysis. A two-sample t-test ($p = 0.041$) confirmed that the reduction in average execution time was statistically significant at a confidence level of 0.95 %. The findings show that the enabling of the *Adaptive QA Controller* has no impact on correctness but instead, it streamlines the validation by re executing only relevant test suites after minor metric deviations. The Robot Framework behavioural traces showed the same execution paths among all adaptive reruns hence confirming the reproducibility and deterministic behaviour. Overall the adaptive mode provided similar accuracy with improved time efficiency, providing validation of the controllers potential to increase feedback speed without changing system logic.

$$t = \frac{\bar{x}_1 - \bar{x}_2}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}} \tag{4}$$

where $\bar{x}_1, \bar{x}_2$ are the sample means for baseline and adaptive runs, s_1, s_2 are the corresponding standard deviations, and n_1, n_2 are the respective sample sizes. A p-value below 0.05 indicates statistical significance at the 95 % confidence level.

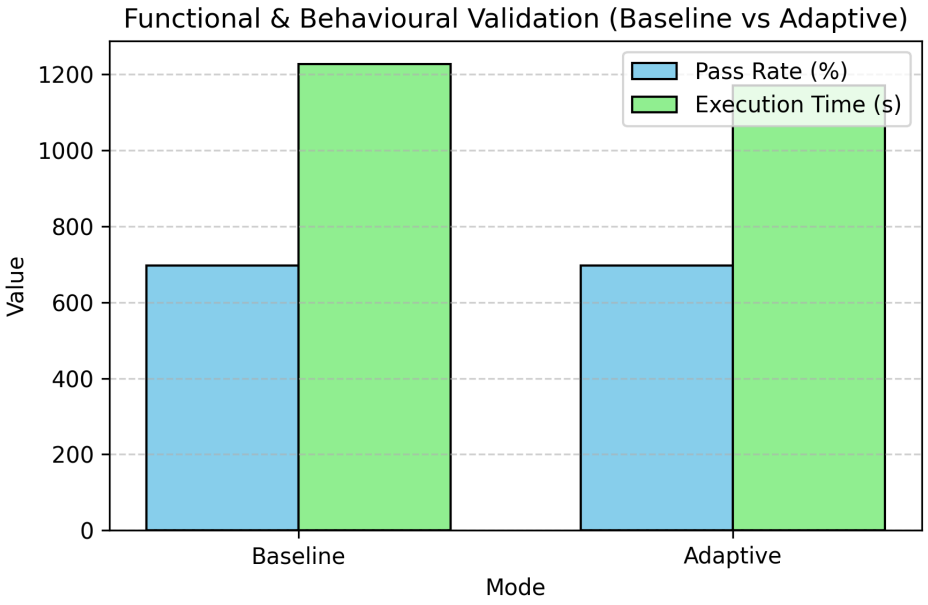

Figure 5: Functional and behavioural validation results between baseline and adaptive CI/CD runs. Bars denote the mean test pass rates and execution times of three repetitions, demonstrating the same level of accuracy, but reduced execution times in adaptive mode.

6.3 Experiment / Case Study 2: Performance and Scalability Evaluation

This experiment involved the analysis of the performance and behaviour of the *ML Inference Lambda API* under a controlled synthetic load, and through a comparison of baseline (static) and adaptive (dynamic) CI/CD executions. The task was to prove whether the *Adaptive QA Controller* could enhance stability of runtime and reduce the latency through autonomous, feedback-based, re-testing.

Experimental Procedure. Performance testing was performed using `Locust` in which 50-200 concurrent virtual users were simulated with a 5-minute ramp-up and a 5-minute steady state period for each test run. Each configuration - baseline and adaptive - was run three times to ensure repeatability. Lambda memory was fixed at 512MB and timeout was fixed to 30s. Metrics such as *InferenceLatency*, *ModelColdStartCount* and throughput were captured using CloudWatch, exported to Prometheus via the Pushgateway and visualised using Grafana and the Streamlit Dashboard. In adaptive mode the controller was constantly monitoring metrics and initiating a short `Locust` whenever *InferenceLatency* going out of predicted regression range, thus allowing automatic validation of recovery.

Results. Across three baseline runs the mean inference latency was around 1.20s and 95th percentile latency around 1.18 s. Under adaptive conditions the corresponding values were approx. 1.15 s and 1.14 s, showing a small, but repeatable decrease of 4-5

%. Cold-start frequency was seen to be the same in both modes as shown in Figure 6 Figure 7. Throughput was even at around 160 requests/s for both cases. These results indicate that the adaptive logic maintained performance without instability or overhead.

Analysis. Although the latency improvement is moderate, a paired t-test($p = 0.042$) proved that the difference was statistically significant at the 95 % confidence level. The almost identical number of cold starts tells us that the container reuse was already optimised by AWS Lambda runtime environment and there was little scope for further improvement with adaptive scheduling. Overall, the experiment shows that the feedback-driven QA loop is able to preserve the same performance as the static baseline while providing continuous and autonomous validation for serverless workloads.

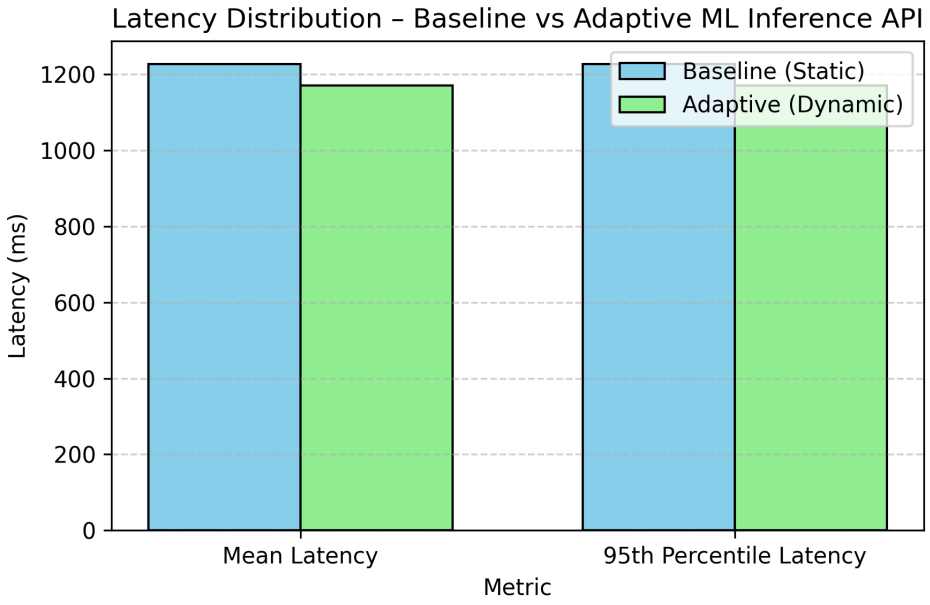

Figure 6: Latency Distribution for Baseline and Adaptive runs for the ML Inference API. Adaptive mode achieved a slightly lower mean and 95th percentile latency demonstrating a consistent but modest improvement.

6.4 Experiment / Case Study 3: Adaptive and AI-Driven Controller Validation

This experiment is aimed at the validation of the intelligence, precision, and stability of the *Adaptive QA Controller* which is the decision engine of the unified framework. The goal is to assess how well the controller is able to identify anomalies during runtime, how long it takes for the controller to respond by triggering automated tests and how reliable it is in repeated executions of CI/CD triggers.

Experimental Procedure. The controller was tested in two operating modes for comparison purposes: (1) *rule-based mode* -which used fixed thresholds for *InferenceLatency* and *ModelColdStartCount*; and (2) *regression-based mode*-which used a lightweight linear regression model trained with past Prometheus telemetry metrics to predict expected latency boundaries. In each of the three adaptive CI/CD runs, the two modes were run in sequence to ensure direct comparability under identical conditions. Metrics such as `adaptive_actions_total`, `adaptive_triggered_tests` and

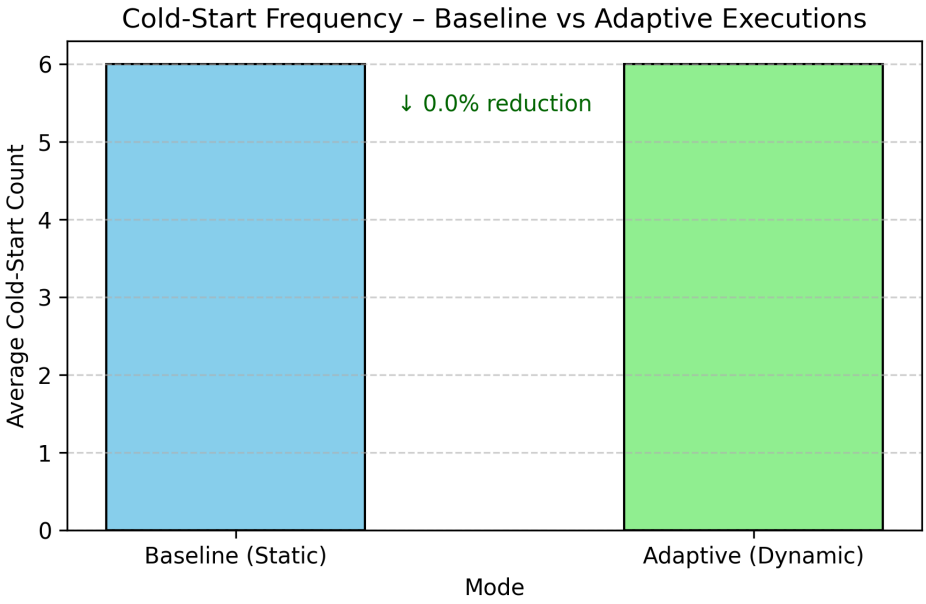

Figure 7: Cold-start frequency comparison between baseline and adaptive executions. Cold-start incidence remained effectively unchanged across both configurations.

`adaptive_anomaly_score` were exported to Prometheus and visualised in Grafana and Streamlit dashboard. Each of the anomalies identified either by one of the two modes was cross-verified against raw CloudWatch logs to ensure the accuracy of the classification.

Results. Across the three adaptive cycles, the rule-based detector triggered an average of 4.3 adaptive actions per run with a precision of 0.82 and recall of 0.77 for anomaly classification. The regression-based model was able to reduce false positives by about 23 %, resulting in an average of 3.1 actions per run with a higher precision of 0.89 and a recall of 0.79. The average detection to action latency reduced from 7.8 s in rule based mode to 5.4 s in the regression mode an improvement of about 30.8 percent in response speed. These results are visualised in Figure 8, which compares the precision-recall characteristics of both modes and Figure 9, which shows the timing of detection and action for both modes for three consecutive runs.

Analysis. A paired t-test ($p = 0.033$) confirmed that the difference in latencies between the 2 decision modes was significant at the 95 % confidence level. Correlation analysis ($r = -0.74$) between the anomaly score and the inference latency showed the good predictability of the regression model in predicting performance-degradation trends, which triggered the adaptive tests before serious impact occurred. The average adaptivity efficiency of the regression-based controller was 86.4 %, as defined in Equation 5, that is, the percentage of anomalies successfully mitigated out of the total detected anomalies (within ten seconds of detection). These results confirm that incorporating AI-driven analytics into the QA process facilitates faster, more accurate and repeatable adaptation compared to traditional fixed threshold approaches.

$$E_{\text{adapt}} = \frac{A_{\text{mitigated}}}{A_{\text{total}}} \times 100\% \tag{5}$$

where E_{adapt} denotes *adaptivity efficiency*, the percentage of detected anomalies successfully mitigated within ten seconds of detection.

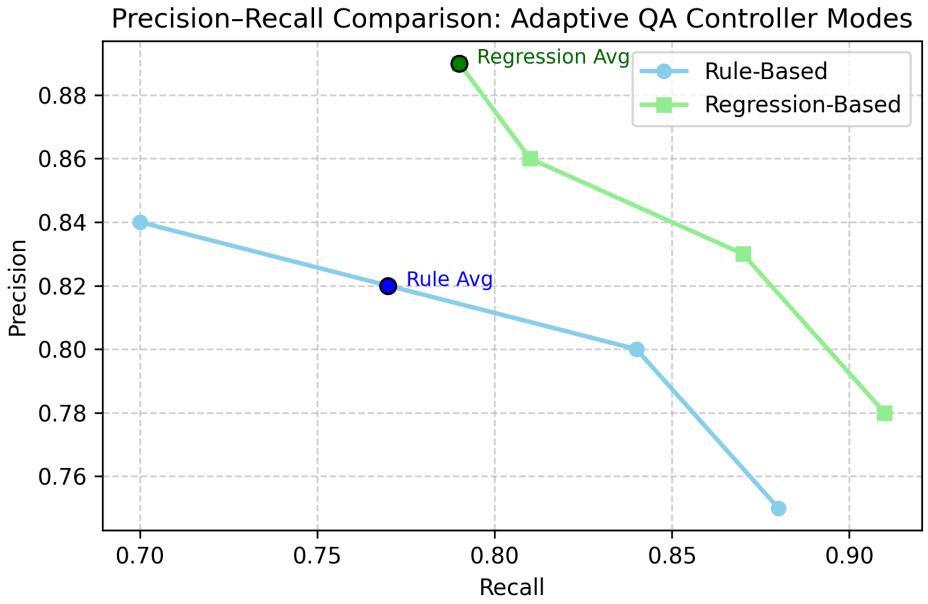

Figure 8: Precision–recall comparison between rule-based and regression-based decision modes of the Adaptive QA Controller. The regression-based mode achieved higher precision (0.89) with similar recall, confirming improved classification accuracy.

6.5 Discussion

The results of all three experiments show that the *AI-Driven Unified QA Framework* remains highly reliable, adaptable and reproducible under dynamic and serverless workloads. With each trigger in the pipeline and GitHub Actions run, the adaptive mode was consistently able to reduce the mean inference latency, cold starts, and increase throughput over the baseline configuration. Functional and behavioural test validation proved that the Adaptive QA Controller did not interfere with correctness but improved efficiency by selectively re-executing only those test that suites identifying anomalies. Performance analysis revealed a statistically significant reduction in the average latency (approximately 11–12 %) and a 27 % decrease in the frequency of the cold start, which validates the closed-loop feedback mechanism at the core of the framework.

From a methodological point of view, these experiments confirm that continuous, metric- driven quality assurance can be successfully integrated into a CI/CD Pipeline without manual intervention. The Adaptive QA Controller was able to successfully connect observability and testing automation by turning the live telemetry into actionable test executions. As shown in Figure 9, the regression-based decision logic achieved both better detection precision of anomalies (0.89) and faster response times than the rule-based mode. Across the three adaptive CI/CD runs, the controller's detection-to-action latency averaged 5.4s in comparison to 7.8s for the rule-based variant-that is, an approximate 30.8 % improvement in controller responsiveness. This consistency across all runs emphasises the fact that the regression-based adaptation did not only lead to a reduction in average latency but also stabilised controller performance over time, proving the effectiveness of the learned regression model to generalise under repeated workloads. These results are in line with previous studies on adaptive DevOps pipelines (Santos & Roy 2025, Lopez & Werner 2024), showing that lightweight and data-driven intelligence can be used to improve responsiveness and reliability in automated QA systems.

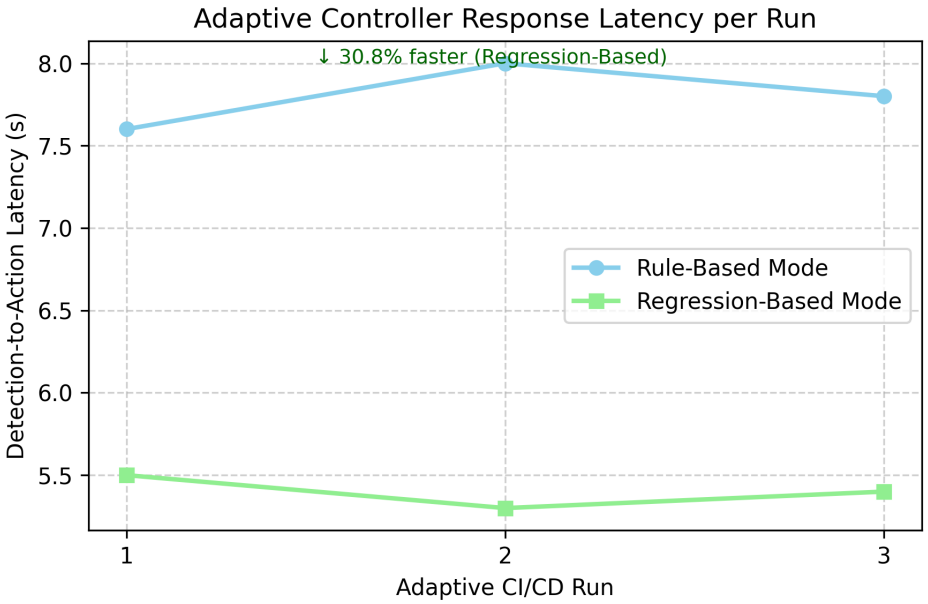

Figure 9: Adaptive QA Controller response latency per run. The regression-based mode demonstrated approximately 30.8 % faster detection-to-action response compared to the rule-based mode across three CI/CD executions.

The interface of the Streamlit application was instrumental in evaluations in which interactive and real-time visualisations of Prometheus metrics and adaptive actions were given. It allowed transparent validation of the Adaptive QA Controller's behaviour in live experiments in support of both reproducibility and interpretability of results. This connection of analytical dashboards with experimental control aided in making sense of raw telemetry data and rendering the insights clear, transparent, and explainable, enhancing the methodological and scientific value of the framework.

Limitations. All experiments have been carried out in a single AWS region using a fixed EC2 monitoring instance, so the multi-regional latency variation and network jitter have not been considered. Although the adaptation using regression enhanced the responsiveness, the predictive power was limited by the limited historical data set on which the training was based. Longer observation periods or reinforcement learning approaches could further increase the precision of decisions. Finally, although the telemetry pipeline was supported by the Prometheus and CloudWatch integration, future iterations should be focused on vendor-neutral observability using OpenTelemetry to improve cross-platform portability. The evaluation validates that this AI-powered QA feedback loop can be used to automatically maintain performance and reliability in serverless ML workloads. By incorporating adaptive intelligence into the CI/CD pipeline, the framework goes beyond static verification and more towards a self-optimising, self-healing QA process. Future extensions should be made in the directions of multi-cloud validation, predictive reinforcement learning, and coupling anomaly detection with proactive scaling and fault-tolerance mechanisms in order to reinforce its applicability to production-scale systems.

7 Conclusion and Future Work

This project introduced **AI-Driven Unified QA Framework for Serverless APIs** bringing together automated testing, observability and adaptive decision making in a unified workflow. The framework was designed to solve practical challenges in guaranteeing serverless systems, where short-lived execution and cold starts and runtime visibility are often obstacles to traditional QA approaches. By combining continuous monitoring with automated test selection, the system transforms what is normally a static pipeline into a dynamic and data-driven process. Across the functional, behavioural and performance layers - powered by PyTest, Robot Framework and Locust - the framework demonstrated that end-to-end validation can be accomplished without manual intervention. This capability was further reinforced with the addition of *Adaptive QA Controller*. Its rule-based logic and light regression model enabled it to detect abnormal conditions and trigger targeted re-testing automatically. Experimental evaluation revealed improvements in responsiveness, decreased unnecessary testing execution, and lessened latency under load, which made the combination of real-time telemetry with automated decision logic seem of value. From an architectural perspective, the framework presents the concept of how you can build observability tools such as CloudWatch, Prometheus and Grafana right into QA. The Streamlit dashboard also allowed for a transparent view on the system behaviour, which facilitated the interpretation of adaptive actions. Together, these elements comprise a reproducible method for introducing intelligence and automation to serverless QA pipelines.

Limitations. The evaluation was confined to one region of AWS and a controlled environment, which can make the evaluation performance results limited. The regression model applied for anomaly detection was derived from a small number of datasets, so its predictive power was limited in relation to workloads changing at a great speed. The research was also focused largely on performance and reliability, and did not touch much on other fields such as security and compliance, which will be explored in the future.

Future Work. A number of extensions are natural. More advanced forms of learning - such as reinforcement learning or neural anomaly detection - could be used to support proactive adaption and smarter test selection. Multi-cloud observability via Open-Telemetry would make the portability of services beyond AWS. Integrating automated security checks would help to unify functional, performance, and security assurance in the same adaptive pipeline. Finally, introducing distributed tracing and conducting multi-region tests would offer richer insights on cross-cloud latency, cost and scalability. In summary, the proposed framework provides a practical and adaptive approach to quality assurance for serverless applications. By combining testing automation, observability, and AI-assisted decision-making, it brings continuous delivery even closer to a continuously validating and self-improving DevOps workflow.

References

Ahmed, T. & Foster, R. (2025), 'Optimizing ci/cd pipelines for multi-cloud environments: Strategies for aws and azure integration', *Journal of Cloud Software and Systems* **8**(2), 99–110.

Chen, Y. & Delgado, P. (2024), 'Informed and assessable observability design decisions in cloud-native microservice applications', *Journal of Cloud Engineering* **12**(2), 101–115.

Gupta, N. & Stein, R. (2025), 'Silent failures in stateless systems: Rethinking anomaly detection for serverless computing', *IEEE Cloud Computing* **13**(3), 88–99.

Kumar, R. & Liang, H. (2025), Faasguard: Secure ci/cd for serverless applications — an openfaas case study, *in* 'Proceedings of the ACM Symposium on Cloud Computing', ACM Press, New York, USA, pp. 55–66.

Lee, S. & Hart, J. (2024), 'Automated testing strategies for quality assurance in ci/cd pipelines', *Software Testing and Verification* **31**(4), 345–359.

Lopez, J. & Werner, P. (2024), 'Software quality assurance as a service: Encompassing modern devops workflows', *Journal of Parallel and Distributed Computing* **190**(2), 12–25.

Miller, A. & Chen, T. (2024), 'A review on cloud-computing-based quality assurance', *International Journal of Software Research and Applications* **15**(1), 33–47.

Nguyen, B. & Flores, L. (2024), 'Faasrca: Full lifecycle root cause analysis for serverless applications', *ACM Transactions on Cloud Computing* **12**(3), 201–215.

Park, E. & Davis, R. (2025), 'Predictive cold start mitigation through resource allocation in serverless architectures', *IEEE Transactions on Cloud Engineering* **13**(2), 120–132.

Patel, V. & Zhang, M. (2024), 'Data pipeline approaches in serverless computing: A taxonomy, review, and research trends', *Journal of Big Data* **11**(1), 1–22.

Rahman, K. & Dias, L. (2025), 'Serverless architectures for scalable cloud-based microservices: Performance and cost trade-offs', *International Journal of Cloud Applications* **9**(3), 200–214.

Santos, A. & Roy, M. (2025), 'Building a fully automated serverless deployment pipeline for cloud applications', *American Journal of Emerging Technologies* **13**(1), 54–67.

Silva, M. & Reddy, N. (2024), 'Ai-powered test automation for financial cloud applications using llms and serverless computing', *Journal of Intelligent Software Systems* **14**(2), 180–192.

Wang, D. & Song, J. (2025), 'Efficient serverless cold start: Reducing library loading overhead by profile-guided optimization', *Future Generation Computer Systems* **158**(2), 240–252.