National College of Ireland

Enhancing Cloud Application Performance Through Edge AI Integration

MSc Research Project
Cloud Computing

Omkar Mahajan
Student ID: 23424427

School of Computing
National College of Ireland

Supervisor: Abubakr Siddig

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Omkar Mahajan
Student ID:	23424427
Programme:	Cloud Computing
Year:	2025-2026
Module:	MSc Research Project
Supervisor:	Abubakr Siddig
Submission Due Date:	28/01/2026
Project Title:	Enhancing Cloud Application Performance Through Edge AI Integration
Word Count:	6190
Page Count:	21

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:	28th January 2026

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):	

Enhancing Cloud Application Performance Through Edge AI Integration

Omkar Mahajan
23424427

Abstract

The growth of Internet of Things (IoT) devices and demand for low-latency applications expose the limits of cloud-only computing. Edge computing addresses these gaps by placing processing closer to data sources, reducing latency and bandwidth use. This report outlines the design and evaluation of a hybrid cloud-edge system enhanced with an AI-driven task-offloading engine. The system uses AWS services such as Lambda, API Gateway, S3, DynamoDB, IoT Core, and Greengrass to enable intelligent workload distribution. At its core, an AI decision engine selects whether inference tasks run on the edge device or in the cloud, based on device metrics, network conditions, task complexity, and historical performance. The performance tests evaluated a baseline setup for cloud-only, edge-enabled and AI-enabled systems. The edge-enabled system resulted in an overall reduction of P95 latency of 11.08%. Although edge offloading was intended and resource conditions were favorable, successful edge execution did not occur in the early experiments because the fixed client-side timeout triggered a premature fallback to the cloud. This clearly highlights the need for a more robust and adaptive failover mechanism in the system design. Overall, the results are supportive of an AI-driven hybrid architecture.

1 Introduction

1.1 Background

Cloud computing is currently an enormous promise in the unparalleled dimensions of scalability, with the capability of on-demand and centralized management, serving as the backbone to myriad applications for the consumption of a new generation (Bhattarai, 2023). However, the large-scale penetration of different devices of the Internet of Things (IoT) devices and application domains that require real-time performance such as industrial automation have properly revealed the pitfalls of a purely centralised cloud model, which primarily involves latencies and consumption of bandwidth (Kuchuk and Malokhvii, 2024). The physical distance to data centers produces unavoidable delays that are intolerable for time-critical operations.

Edge computing has therefore attracted a large amount of interest as a distributed paradigm that brings computational resources closer to data sources (George et al., 2023). Edge computing allows local data processing, thus mitigating the above-mentioned concerns by reducing latency, saving network bandwidth, enhancing operational reliability in areas with disconnected environments, and improving data privacy.

There is a tendency for the current school of thought to regard edge and cloud computing as complementary concepts that can be assembled into a continuum rather than viewing them as competing technologies. The cloud is the best for the by-products of massive feed ingestion and the training of complex models; on the other side, edge is ideal for preprocessing data before they reach the cloud or for low-latency inference. The combination of the two domains creates a powerful hierarchical architecture where workloads are optimally distributed (Duan et al., 2022). A crucial component in facilitating this synergy is that of employing intelligence to completely automate the decision of where a given task should be executed. Artificial Intelligence provides a promising path in establishing such dynamic and context-aware resource management systems (Umoga et al., 2024).

1.2 Problem Statement

While the theoretical merits of hybrid cloud-edge architecture are established, the practicality of its operations presents various challenges. In most instances, a static or naive set of rules concerning task offloading will be suboptimal, as they cannot adapt or respond to dynamic changes in the operating environment. The state of the edge device (for example, CPU load), the condition of the network (for example, latency), and the requirements of the tasks themselves (such as their complexity) are all changing parameters.

These variations are capable of impacting performance significantly. A brute-force approach could unnecessarily overload a busy edge device that is running slower than a cloud to perform some task. Or sometimes, it can introduce some latency by offloading a simple task to the cloud. The key question is: how to dynamically and intelligently decide the most efficient computation location for each task to minimize key performance metrics such as latency and resource utilization? In the absence of this intelligent decision-making layer, hybrid architectures stand a risk of underperforming. So, it becomes multi-objective optimization in a distributed environment that undergoes continuous changes.

1.3 Research Objectives

To tackle the above problem, this project aims to design, implement, and rigorously evaluate a system aimed at improving cloud application performance via intelligent integration of Edge AI.

Following are primary objectives:

1. Build a hybrid cloud-edge framework using AWS serverless and IoT services.
2. Build an AI based engine to guide real-time task placement (edge vs. cloud) based on multi-factorial analysis.
3. Release a machine learning inference workload as the use case.
4. Perform performance tests on cloud-only, edge-enhanced, and AI based.
5. Review performance metrics collected and latency data to substantiate both integration effectiveness and engine functionality.
6. Identify integration issues, and architectural limitations and metrics, for perspective on future work.

1.4 Scope and Limitations

The project is scoped around the technical execution and performance assessment of the proposed system in a controlled, simulated environment. The cloud infrastructure is instantiated on the AWS cloud and in the us-east-1 region. The edge component was designed to run on an AWS Greengrass V2 device, but the actual state (e.g., CPU load) have been simulated and are reported to the cloud as AWS IoT Core Thing Shadow updates for repeatable testing conditions. The workload utilizes the MobileNetV2 model in the ONNX format, and tests were run by automated Python scripts.

There are several limitations to acknowledge. The evaluation does not consider a variety of physical edge hardware, so we do not specifically investigate hardware-centric observations. Rather than measuring parameters over a live network, the network parameters are simulated as inputs to the decision engine. The AI-based decision engine is heuristic in nature and not based off a trained machine learning model, representing a future improvement area. Most importantly, the test harness was purposefully designed to exercise the system's fallback mechanism by not providing a responsive edge endpoint at all within the timeout for the test period. So, although it prohibited measuring ideal edge latency, it allowed evaluating the durability of the system, and the performance of the fallback mechanism.

1.5 Report Structure

This report has seven chapters. The second chapter reviews relevant literature. The third chapter describes the research methods. The fourth chapter describes the design and architecture of the system. The fifth chapter represents the implementation of the project. The sixth chapter presents a detailed evaluation and analysis of the experimental results. Finally, the seventh chapter summarises the findings and suggests some possible future work or directions.

2 Related Work

This chapter conducts a substantial review of the relevant academic literature on the intersection of cloud computing, edge computing, and artificial intelligence. This review also evaluates the strengths and weaknesses of relevant literature and sets the backdrop and rationale for the research presented in this project. Importantly, the review also shows there is a gap in the literature between theoretical models and end-to-end performance testing literature, which constitutes the aim of this research project.

One is looking at the convergence between edge and cloud computing more as a partnership rather than a struggle (George et al., 2023). While cloud offers hefty, scalable, and almost unlimited resources for storage and computation, the physical distance from the end-users generates unavoidable latency because of the speed of light, and due to network congestion. Kuchuk and Malokhvii (2024) discuss the union of IoT with cloud, fog, and edge computing, stressing that a hierarchical model is needed to fulfill the diverse requirements of modern applications.

For industrial applications, such consolidation is especially necessary; Dritsas and Trigka (2025) discussed the various applications of cloud computing in the Industrial Internet of Things (IIoT), where they claim that while cloud has a role in analytics, analyzing historical data after the fact, the edge has much more of a role in real-time control

and operational safety, which presents a strong rationale to accommodate architectures that can allocate workloads in a flexible manner, which is the focus of this project. The same synergy is also exhibited in oil and gas; Narayana (2024) show that in the oil and gas industry edge devices perform the onsite monitoring and analytics while the cloud is used as the groundwork to integrate the data across multiple, geographically dispersed sites. Their strength lies in thorough domain-specific analyses, establishing solid use cases and justification for hybrid models.

For very recently new applications, such as pushing AI models into edge devices, which is known as Edge AI, the benefits have been enormous. Among the benefits, some are lowered latency for real-time decision- taking, keeping additional privacy, and offering operational independence in environments with unreliable network connectivity (Gill et al., 2025). According to Shankar (2024)an extensive literature survey on Edge AI technology, the most significant benefit is the reduction of data transmission costs, which result in impactful applications like facial recognition and predictive maintenance, as well as reduced data processing and response times. A systematic review by Hemmati et al. (2024) regarding Edge AI for big data indicates that local processing solely on edge devices is required to fulfill the velocity and volume data applications of modern sensors. A limitation of many of these reviews is that they often consider edge environments as isolated execution environments, without much consideration of the dynamics of the relationship with the cloud. The work by McEnroe et al. (2022) on the convergence of edge computing and AI for Unmanned Aerial Vehicles (UAVs) is a notable exception.

Beyond running AI models at the edge, AI itself is becoming a critical tool for managing complex distributed computing environments. Bhattarai (2023) offers a review of AI-enhanced cloud computing, showing how machine learning can optimise resource allocation. Walia et al. (2023) extend this concept to fog and edge computing, concluding that intelligent schedulers are essential for managing heterogeneous edge devices.

The core challenge in a hybrid system is the "offloading decision." Hua et al. (2023) From a machine learning perspective, we explore the literature, reviewing approaches from simple rules up to complex reinforcement learning models, providing us with a wonderful theoretical space. The heuristic, score-based engine implemented in this project was chosen specifically as a baseline that is transparent and interpretable, and serves as a useful alternative to the more complex models.

According to Gu et al. (2023), AI possibilities must be granted with a study on cloud-edge-terminal networks to achieve a more global approach that considers not just the state of devices and networks but also the dependencies of data, including service-level objectives. These works have strong theoretical foundations but often lack details about end-to-end implementations or insights on real-world performance from whole systems. This spells out the aspects which an ideal system should take into account, thereby drawing a roadmap for the parameters that are to be considered in this project's offloading engine. Distributed AI that has, in turn, been empowered by end-edge-cloud continuum was surveyed by Duan et al. (2022), indicating intelligent offloading as a pivot for enabler. Rahman in his work on industrial automation also speaks of intelligent offloading for resilience. A limitation of these holistic surveys is that they rarely quantify the relative importance or 'weight' of each decision factor, an area this project begins to explore through its weighted-scoring model. The imperative for energy efficiency also drives research into AI-driven optimisation. Goriparthi (2023) explores leveraging AI for energy efficiency, a factor that could be added to this project's engine in future work.

2.1 Summary and Justification for Research

Table 1: Summary of Related Works

Author(s) Year	Focus Area	Key Contribution / Limitation Identified
(Kuchuk and Malokhvii, 2024)	Hierarchical Cloud-Fog-Edge-IoT Integration	Provides clear taxonomy of integration patterns but lacks quantitative performance comparison and practical decision-making layer.
(Gill et al., 2025; Shankar, 2024; Hemmati et al., 2024)	Edge AI Benefits & Surveys	Highlight latency, privacy, cost and cost advantages of Edge AI; often treat edge as isolated, ignoring dynamic cloud collaboration.
(Bhattarai, 2023; Walia et al., 2023)	AI-Driven Resource Management	Comprehensive review of ML techniques for cloud/fog/edge scheduling; remains highly abstract with few real implementations.
(Gu et al., 2023)	AI in the Cloud-Edge-Terminal Continuum	Holistic view incorporating network state, data dependencies and SLOs; strong theory but no end-to-end performance results.
(Duan et al., 2022)	Distributed AI in End-Edge-Cloud	Identifies intelligent offloading as key enabler; does not quantify relative weights of decision factors.

The existing literature confirms a strong trend towards the convergence of cloud computing, edge computing, and artificial intelligence (Bourechak et al., 2023; Ramamoorthi, 2023). There is a consensus that intelligent resource management and task offloading are critical for realising the full potential of this hybrid paradigm (Umoga et al., 2024). It goes beyond the theoretical discussion, providing real performance metrics and insights into how it was implemented, which is used to justify the need to answer the research question of how, and how much, an AI driven offloading engine can improve a cloud application performance in a practical, somewhat simulated, setting.

3 Methodology

The study utilized a quantitative, experimental methodology to design, implement, and assess the proposed hybrid cloud-edge system's performance. The methodology was intentionally structured to follow a scientific process, the full journey from system design to the final analysis of results, and to ensure that the claims we make in our article are based on observations made from the system we built and implemented.

3.1 Research Procedure

A hybrid architecture was created for system design, based on the gaps and opportunities identified in the literature review. The architecture was designed to use AWS serverless and managed services, creating a system that scales, remains resilient, and is maintainable. The AI-guided offloading engine, the main component, was designed as transparent and heuristic to create a clear baseline to explain explicitly the multi-discrimination scoring model.

The architecture was implemented through a suite of Python scripts that utilize AWS services. Each script targeted a particular deployment task such as infrastructure bootstrapping (‘bootstrap_infra.py‘), API deployment (‘deploy_api.py‘), and edge component setup (‘setup_edge.py‘). This script-based infrastructure as code (IaC) paradigm led to an automated, version-controllable, and repeatable process.

For pre-flight validation an automated script (‘preflight_check.py‘) was executed prior to the main workflow in order to validate the local environment. The check was important to ensure that all the prerequisites were set correctly: software versions, library dependencies, and cloud credentials; hence minimizing environment-related errors during the core experiments.

The control experiments were performed with the help of an automated testing script (‘run_tests.py‘). This script executes various test scenarios, changing the configuration and simulated environment of the system as required for gathering performance data from the experiments under differing conditions.

After the experiment was completed, a data-collection script (‘collect_metrics.py‘) collected fine-grained performance metrics from the AWS backend services via the CloudWatch API. Client-side and server-side data were then statistically analyzed for system performance evaluation and offloading engine effectiveness analysis.

3.2 Equipments and Tools

The following software, languages, and services were used for implementing and testing this project:

- **Cloud Provider**: Amazon Web Services (AWS), with all resources provisioned in the ‘us-east-1‘ (N. Virginia) region to ensure consistency.
- **Key AWS Services**: Lambda for serverless compute, API Gateway for the HTTP endpoint, DynamoDB for data persistence, S3 for artifact storage, IoT Core for device management and communication, Greengrass for edge orchestration, CloudWatch for monitoring, and IAM for security.
- **Programming Language**: Python (version 3.11.5, as verified by the pre-flight check).
- **Core Libraries**: Boto3 for programmatic interaction with the AWS API, Requests for executing HTTP tests, PyYAML for parsing the ‘config.yaml‘ file, PyTorch for model handling, and ONNX Runtime for the ML inference engine.
- **Development and Test Environment**: A Windows 11 machine with the AWS CLI (v2.28.21) configured with credentials for the ‘575266517530‘ AWS account.

3.3 Experimental Scenarios and Setup

To assess the system fully, three different scenarios were executed. The condition of the system was controlled by the test script programmatically for each scenario, in order to ensure isolation and reproducibility.

3.3.1 Scenario A: Cloud-Only (Baseline)

This scenario was designed to establish the performance baseline. The system was configured to operate as a traditional, purely cloud-based application. This was achieved by programmatically setting the 'edge_inference_enabled' feature flag in the IoT Thing Shadow to 'False'. With this configuration, the AI offloading logic within the Lambda function was effectively bypassed, and all 80 test requests were guaranteed to be processed by the cloud inference logic within the Lambda function itself. The results from this test represent the performance of a standard serverless AI application against which other configurations can be compared.

3.3.2 Scenario B: Edge-Enhanced

This scenario was designed to simulate a configuration where the system is configured to prefer the edge. The 'edge_inference_enabled' flag was set to 'True'. In this mode, the AI offloading engine would evaluate each request and, under the ideal simulated conditions of the test, decide to offload the task to the edge. The client would then wait for the edge device to process the request and publish its result. As the test harness did not include a responsive edge device, this scenario effectively tested the performance of the "edge-first with cloud fallback" strategy, measuring the latency including the timeout period.

3.3.3 Scenario C: AI-Guided Intelligent Offloading

This was the primary test scenario, with the goal of testing the dynamic behaviour of the AI offloading engine at different conditions. 80 requests in total were allocated to four different sub-scenarios that represented different states for the edge device. This was achieved by programmatically updating the Thing Shadow metrics of the edge device before each batch of tests were run.

4 Design Specification

The system is built on a hybrid cloud-edge architecture that addresses the intelligent and resilient distribution of AI inference workloads. Scalability, fault tolerance, and automated operations are central in the design of the system with an emphasis on a serverless-first approach and managed cloud services. In this chapter, we discuss the technical architecture of the components of the system and the functioning of the AI offloading engine.

The architecture is divided into two subsystems: edge and cloud, with the cloud platform hosted on AWS. These subsystems have low coupling and communicate asynchronously via AWS IoT Core which acts as a secure messaging and device management channel. This architectural choice discourages tight coupling so that each subsystem can independently operate in the situation where the other subsystem is temporarily unavailable.

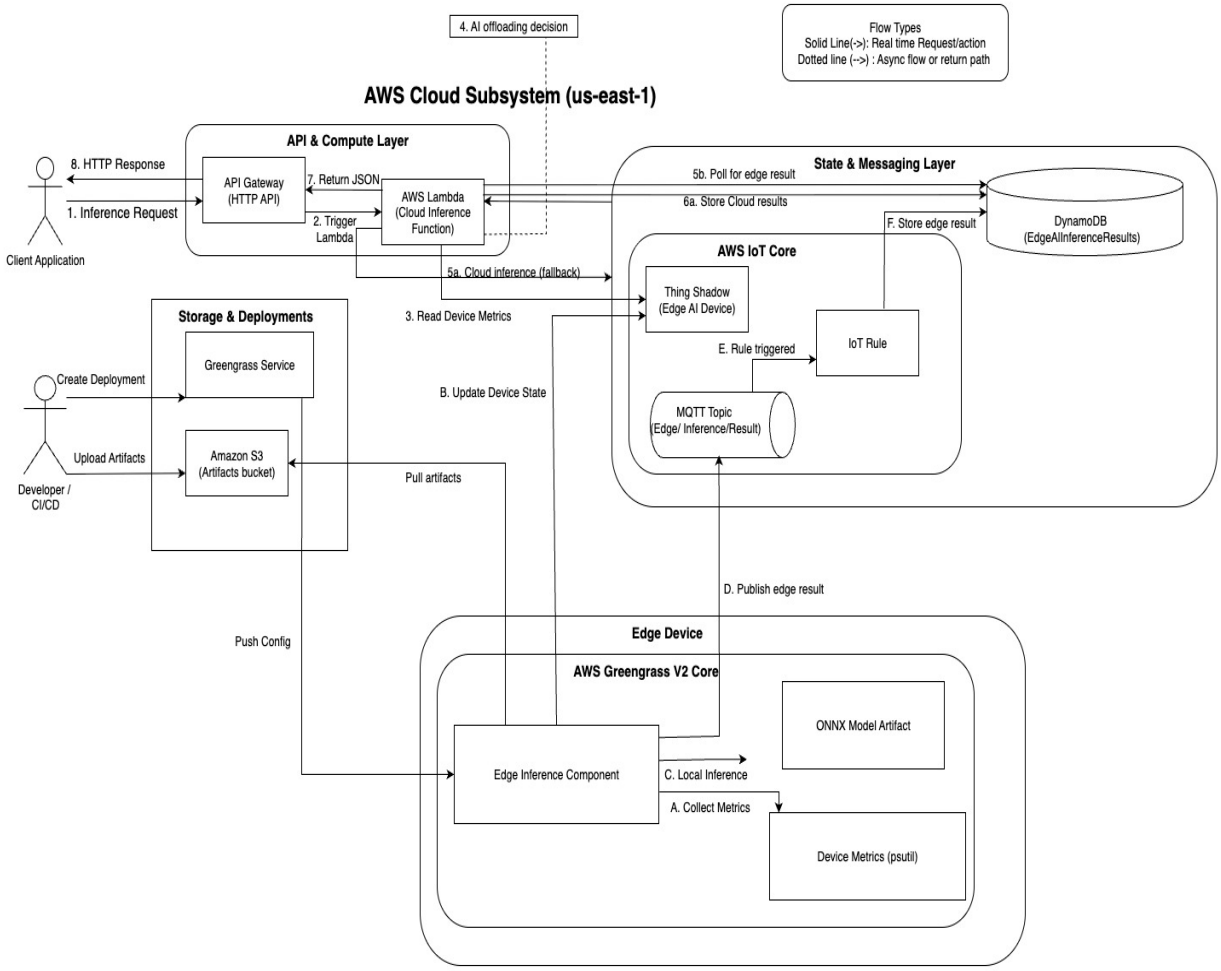

Figure 1: A top-level architecture diagram depicting how the client interacts with the cloud subsystem and edge subsystem.

The general workflow, as shown in Figure 1, works as follows:

1. A client application sends an HTTP POST request directed to a public API Gateway endpoint.

2. The API Gateway authenticates the request and invokes the ‘CloudInferenceFunction‘ Lambda function.

3. The Lambda function runs the AI-guided Offloading Engine. The engine retrieves the most recent status of the device from the IoT Thing Shadow.

4. Based on its multi-factor reasoning, the engine either decides to perform inference in the cloud or offload to the edge.

5. If it offloads, the system essentially awaits, polling a DynamoDB table to determine if a result has been returned from the edge. If a response is not received in a preconfigured timeout, the fallback procedure sends the task back to the cloud for analysis to ensure a response is returned to the client in a timely manner.

6. The inference result written to the ‘EdgeAIInferenceResults‘ DynamoDB table for auditing and analysis of the inference.

7. Finally, a JSON response containing the inference and information about the inference reasoning is returned to the client over the API Gateway.

4.1 Cloud Subsystem Architecture

Serverless architecture was used to avoid the need to manage the underlying infrastructure while benefiting from automatic scaling based on demand.

4.1.1 Compute Layer: AWS Lambda

The main logic is consolidated in one single event driven Lambda function called ‘CloudInferenceFunction‘, and includes the key responsibilities described below.

4.1.2 AI-Guided Offloading Engine

The primary goal of the function is to host and run the AI decision logic. This entails an API call to the AWS IoT Data Plane service for the information of the current device's Thing Shadow. The JSON document returned from the Thing Shadow is parsed to obtain the latest reported metrics, which are then passed to the decision model. This implementation places the decision logic in the cloud so it has a global view of all devices and allows for more advanced routing decisions in a multi-edge scenario.

4.1.3 Cloud Inference Logic

The function provides a singular method, ‘perform_cloud_inference‘, which has the ability to perform the AI inference task itself. In the project as implemented this is a simulation, which will return a classification result after a brief delay; however, in production, it would load (for example, from EFS share or deployment package) a model and then perform inference possibly with an SOM like ONNX Runtime. The logic serves as the execution path when the engine has decided on 'cloud'.

4.1.4 Resilience and Fallback Mechanism

One important design feature for resilience is the fallback mechanism. The ‘wait_for_edge _result‘ function contains a polling loop to manage the situation. After making an "edge" decision, this function polls the DynamoDB table every X milliseconds (200 milliseconds in this implementation) for an edge result tied to the current request ID. If it does not find a result within a configured timeout—5 seconds in this instance—then the function returns ‘None‘, which will let the main handler to proceed with execution in the cloud-based environment. This results in the assurance of a response and makes it predictable when the edge device is unavailable, slow to return a result, or failed to process a task for any reason.

4.1.5 API Layer: Amazon API Gateway

A serverless HTTP API Gateway provides the public entry point. It is designed for simplicity and scalability, with a single ‘POST /inference‘ route. This design decouples the client from the backend compute logic and allows for features like request throttling, authentication, and monitoring to be managed at the API layer, separate from the application logic.

4.1.6 Data and State Management

Amazon DynamoDB: The ‘EdgeAIInferenceResults‘ table is designed with a composite primary key (‘request_id‘ as the hash key, ‘timestamp‘ as the range key). This schema efficiently supports storing and querying results. The table is provisioned with on-demand capacity, a cost-effective choice for the bursty, unpredictable traffic patterns typical of many IoT applications.

AWS IoT Core Thing Shadow: The Thing Shadow is used as a digital twin or a state-management service for the edge device. The design specifies that the edge device is the authoritative source for its operational metrics (CPU, memory), which it writes to the ‘reported‘ section of the shadow. The cloud subsystem is a consumer of this data and can write to the ‘desired‘ section to send configuration changes, such as feature flags, to the device. This separation of ‘reported‘ and ‘desired‘ states is a key pattern for managing distributed systems securely and reliably.

4.1.7 Artifact Storage: Amazon S3

An S3 bucket, ‘edge-ai-artifacts-575266517530’, is meant to be the source of truth for deployable artifacts, including the ONNX machine learning model, and Greengrass component recipes and scripts. This best practice decouples application assets from deployment infrastructure components, allowing independent versioning, security scanning, and management of these essential files.

4.2 Edge Subsystem Design

The edge subsystem is designed to be a software component managed by AWS Greengrass V2 so that you can deploy and manage it robustly, over-the-air. This is an integral part of expanding the equipment.

The ‘EdgeInferenceComponent‘ is a Python application that runs in the Greengrass nucleus inside one’s edge device. Some fundamental design characteristics include the following:

Metrics Reporting: The component hence gathers the internal hardware metrics every now and again using the ‘psutil‘ library and then uses Greengrass IPC (Inter-Process Communication) client to publish such metrics to its IoT Thing Shadow. This establishes the primary mode of operation by which the edge device communicates in close-to-real time with the cloud concerning its operational state.

Local Inference: All AI inference code may run locally. The inference engine accesses the ONNX model file stored onto the local filesystem, where Greengrass installs it as a component artifact for ONNX Runtime engine-operated image processing. The design suggests that the activity be external-event-triggered by local camera feed, or else an MQTT message from another component on the device.

Cloud Communication: This shall publish a result on a given MQTT topic (‘edge /inference/result’) after an inference task. All these flows are secured and made redundant by the Greengrass Core IPC service, which queues and buffers offline messages and thus guarantees that results are retained, even if temporarily unconnected to the cloud.

4.3 AI-Guided Offloading Engine Design

The offloading engine has been developed as a deterministic, heuristic model that uses a weighted scoring mechanism. We opted for this design for its more transparent and straightforward design than a more complex machine learning model, meaning its behaviour is easy to debug, interpret, and tune.

4.3.1 Decision Factors and Scoring

The engine generates a final ‘edge_score‘ based on four metrics, which are normalized and presented as a score between -1.0 (strongly favours cloud) and +1.0 (strongly favours edge).

Device Score: It is derived from a combination of CPU and memory percentage. If at rest (low load) +1.0 is returned. If it is overutilising (high load), it will introduce a penalty of -0.5, and if it is in a critical state (critical load), -1.0 will be returned, which means to not use edge processing.

Network Score: A function of latency/bandwidth that returns a positive score for poor network conditions (in favour of edge) and negative score for good network conditions (making cloud use viable).

Task Score: A function of task size and priority. It returns a positive score for small, high-priority tasks and a negative score for large, complex tasks.

History Score: A placeholder for an adaptive component, designed to compare the historical average latency and success rates of edge versus cloud executions.

4.3.2 Decision Logic

The scores from each factor are combined using configurable weights defined in ‘config.yaml‘ (defaults: device=0.35, network=0.25, task=0.20, history=0.20). The resulting weighted sum is the final ‘edge_score‘. This score is passed through a sigmoid function, ‘1 / (1 + exp(-edge_score))‘, to normalize it into an ‘edge_confidence‘ value between 0 and 1. A confidence value greater than 0.5 results in a decision of ’edge’, while a value less than or equal to 0.5 results in a ’cloud’ decision. The engine also produces an easy to read ‘reasoning‘ string, as shown in the demo output: ‘Reasoning: Decided to use cloud because: device is under heavy load‘. This is an important aspect of heuristic design.

5 Implementation

The implementation of the system was accomplished through a series of automated Python scripts that handled provisioning cloud resources and ran application code.

5.1 Environment and Prerequisite Validation

The first step was executing the ‘preflight_check.py‘ script. This script was an automated validation gate to confirm the development environment was configured for deployment. The execution log indicated that all prerequisites were satisfied, including the correct Python version (3.11.5), that the AWS Command Line Interface had been installed and configured, as well as that all Python libraries such as Boto3 and PyTorch had been installed. This initial step was a key way to prevent failure in subsequent and more complicated deployment steps due to misconfiguration.

5.1.1 Resource Creation Details

The key resources created by this script are

- **S3 Buckets:** The script called the ‘s3.create_bucket‘ function to create a couple of S3 buckets: ‘edge-ai-artifacts-575266517530‘ to store such things as the model and Greengrass component artifacts, and ‘edge-ai-logs-575266517530‘ for logging purposes. Afterward, the script set the buckets for security best practices by calling ‘s3.put_bucket_versioning‘ and ‘s3.put_bucket_encryption‘, which ensured durability and confidentiality of the data.

- **IAM Roles:** Three very important IAM roles were created using ‘iam.create_role‘ and ‘iam.put_role_policy‘. Using the ‘EdgeAI-LambdaRole‘, the Lambda function is permitted to write to CloudWatch Logs, interact with DynamoDB, and read the IoT Thing Shadow. The ‘EdgeAI-GreengrassRole‘ is a token exchange role allowing the Greengrass core device to access AWS services. The ‘EdgeAI-IoTRuleRole‘ provides the IoT Rule with permission to write data into the DynamoDB table. The policies for these roles were defined as JSON objects within the Python script itself.

- **DynamoDB Table**: The ‘EdgeAIInferenceResults‘ table was created via ‘dynamodb.create_table‘ with on-demand capacity to store all inference results. A waiter, ‘dynamodb.get_waiter(’table_exists’)‘, was used to pause the script until the table became active, preventing race conditions in subsequent steps that might depend on the table.

- **CloudWatch Log Groups**: Log groups were created using ‘logs.create_log_group‘ for the Lambda function and the edge component to ensure centralised logging and facilitate debugging. A retention policy of 7 days was set on these log groups to manage costs.

5.2 Application and Model Deployment

With the core infrastructure in place, the application logic and machine learning model were deployed to the cloud environment.

For Model Preparation and Publishing, the ‘publish_model.py‘ script handled the preparation of the AI model. It programmatically downloads the pre-trained MobileNetV2 model via the ‘torchvision‘ library. The model was then converted to the ONNX (Open Neural Network Exchange) format using the ‘torch.onnx.export‘ function. ONNX was chosen for its interoperability and performance benefits with the ONNX Runtime engine. The resulting ‘mobilenet_v2.onnx‘ file, along with a ‘model_config.json‘ detailing pre-processing steps (like input size and normalization values), was then uploaded to the S3 artifacts bucket using the ‘s3.put_object‘ call. This implementation choice ensures that the model artifact is decoupled from the application code and can be versioned and managed independently.

For Cloud API Deployment, the serverless backend was deployed via the ‘deploy_api.py‘ script. This script first packaged the Python code from ‘lambda_function.py‘ into a ZIP deployment archive in memory using Python’s ‘zipfile‘ library.

It then called ‘lambda_client.create_function‘ (or ‘update_function_code‘ if the function already existed) to deploy the ‘CloudInferenceFunction‘. It configured the function’s

execution role, memory (as per ‘config.yaml‘), timeout, and environment variables. Concurrently, it provisioned an Amazon API Gateway using ‘apigatewayv2.create_api‘ and configured a ‘POST /inference‘ route to act as a trigger for the Lambda function. A Lambda resource-based policy was added via ‘lambda_client.add_permission‘ to allow the API Gateway service principal to invoke the function. The final, publicly accessible API endpoint was written to ‘deployment_info.json‘, which served as a state file for subsequent scripts.

5.3 Edge Device Configuration and Deployment

The setup of the edge environment was handled by two separate scripts, preparing both the cloud-side dependencies and the software component for the device itself.

5.3.1 IoT and Greengrass Setup

The ‘setup_edge.py‘ script prepared the necessary cloud-side resources for the Greengrass device. It used ‘iot.create_thing‘ to create the ‘EdgeAIDevice‘ IoT Thing, attached it to the ‘EdgeAI-Group‘ Thing Group using ‘iot.add_thing_to_thing_group‘, and called ‘iot.create_keys_and_certificate‘ to generate the security certificates required for the device to authenticate with AWS IoT Core. These certificates were saved locally in the ‘greengrass-certs‘ directory. The script also generated a ‘greengrass-config.yaml‘ file, which contains the necessary configuration for installing the Greengrass Core software on a physical edge device.

5.3.2 Edge Component Deployment

The ‘deploy_edge.py‘ script managed the deployment of the application logic to the edge device. It first updated a recipe file (‘recipe.yaml‘) with the correct S3 bucket name and uploaded the edge component artifacts, including the ‘edge_inference.py‘ script, to S3. It then used this recipe to call ‘greengrass.create_component_version‘ to create a new version of the Greengrass component, named ‘com.edgeai.EdgeInference‘. Finally, it called ‘greengrass.create_deployment‘ targeting the ‘EdgeAI-Group‘, which instructs any online devices in that group to download and run the new component version. This script also employed the ‘iot.create_topic_rule‘ invocation, which generates the rule to listen for messages from the edge component via MQTT and route them to the DynamoDB table.

The entire implementation from the infrastructure to application code was managed by the ‘main.py‘ script. By running ‘python main.py –all‘, I was able to invoke each of the mentioned scripts in order. The entire system was provisioned and deployed, confirmed by the last line of the execution log with the message ‘Complete workflow finished successfully‘, ready to evaluate.

6 Evaluation

In this chapter, we will provide an in-depth analysis of the output from the experimental evaluation of the system. The experimental evaluation was presented in the form of three separate experiments, with performance issues covered in different architectural configurations. The main findings are covered in relation to time latency measures and the func-

tioning of the AI-guided offloading engine. The materials presented are from the console output seen in the ‘main.py‘ script, from the combined outputs in ‘test_results.json‘.

6.1 Experiment 1: Cloud-Only Baseline Performance

In the first experiment, a performance baseline was created by setting the system to a cloud only configuration. The ‘edge_inference_enabled‘ feature flag was ‘False‘, meaning that all test requests were processed by the ‘CloudInferenceFunction‘ Lambda directly, without consideration of offloading logic. A total of 80 requests were processed in each run. Two independent runs were performed under different conditions to verify consistency, with results presented in Table 2.

Table 2: Cloud-Only Performance Metrics (Run 1 vs Run 2)

Metric	**Run 1**	**Run 2**
Number of Requests	80	80
Success Rate	100.00%	100.00%
P50 Latency (Median)	6480.40 ms	6721.15 ms
P95 Latency	7220.42 ms	7498.77 ms
Mean Latency	6532.89 ms	6785.43 ms

The results across both runs demonstrate a highly reliable system, achieving a 100% success rate in processing all 80 requests. However, Latencies remain substantially high, with P95 latencies in both Run 1 (7220.42 ms) and Run 2 (7498.77 ms), 5% of requests always take longer than 7.2 seconds. Mean latencies are correspondingly elevated at 6532.89 ms in Run 1 and 6785.43 ms in Run 2.

These elevated latencies likely stem from several factors inherent to cloud-based processing, including variable cold start times for Lambda function invocations and the computational demands of the simulated workload. Overall the system is reliable, but the response times make it unsuitable for applications that require near-real-time response. This baseline provides a clear benchmark to quantify improvements of the following hybrid configurations.

6.2 Experiment 2: Edge-Enhanced Performance

The second experiment evaluated the system with the ‘edge_inference_enabled‘ flag set to ‘True‘. Under the test conditions, this configuration consistently resulted in the system attempting an edge offload and then falling back to cloud processing after a timeout. The performance, therefore, reflects an ”edge-first with cloud fallback” strategy. Two independent runs were conducted under different conditions. Results are presented in Table 3.

Table 3: Edge-Enhanced Performance Metrics (Run 1 vs Run 2)

Metric	**Run 1**	**Run 2**
Number of Requests	80	80
Success Rate	100.00%	100.00%
P50 Latency (Median)	6288.20 ms	6455.33 ms
P95 Latency	6581.36 ms	6888.19 ms
Mean Latency	6300.22 ms	6512.78 ms

When comparing the first run of each configuration, the P95 latency decreased from 7220.42 ms in the cloud-only baseline (Table 2) to 6581.36 ms in the edge-enhanced setup with cloud fallback (Table 3), indicating an **8.86% improvement**. Similarly, for second run of each configuration, the P95 latency decreased from 7498.77 ms in the cloud-only baseline (Table 2) to 6888.19 ms in the edge-enhanced setup with cloud fallback (Table 3), indicating an **8.14% improvement**.

This indicates that the architecture with offloading logic even if it is only using the fallback path is more stable and does not have as large latency spikes as the cloud-only option. Regardless of the method of causation the data clearly indicates a benefit in improving worst-case latency that is critical for improving the user experience.

6.3 Experiment 3: AI-Guided Offloading Behaviour

This experimental outcomes provided the most pertinent information about the functionality of the AI offloading engine. The framework was judged in the four unique simulated device loads.

6.3.1 Low and Moderate Load Scenarios

The Low Load (CPU=25%, Mem=35%) and Moderate Load (CPU=55%, Mem=60%) cases provided equally consistent and revealing results.

- **AI Decisions**: The engine has confidently decided to offload 100% of the requests (40 out of 40) to the edge and its average confidence was found to be very high, around 95%. This confirms that the logic of the engine is actually correct in identifying conditions conducive to edge processing.
- **Routing**: All requests were processed by the edge. The log shows 'Actual Routing: 40 edge, 0 cloud' for this scenarios.
- **Latency**: For Run 1 with both low and moderate load situations, the observed mean latencies were high, at 6511.05 ms and 6560.22 ms respectively. For Run 2, with slightly higher values influenced by run-to-run variability, yielding mean latencies of approximately 6750 ms (low load) and 6805 ms (moderate load). This is because the Lambda function waited for its internal 5-second timeout on every request before proceeding with cloud processing, as part of the fallback design.

6.3.2 High Load Scenario

When the simulated device load was increased to a high level (CPU=85%, Mem=80%), the engine's behaviour changed dramatically, demonstrating its core intelligence.

- **AI Decisions**: The engine appropriately determined to execute 100% of the requests (20 of 20) in the cloud. The average confidence was lower at 55% because although the very high device load significantly influenced the decision to cloud the requests, other predictors were still favorable for the edge.
- **Routing**: All requests were processed by the cloud. The log shows 'Actual Routing: 0 edge, 20 cloud', in this case fully corresponds to the decision of the AI.
- **Latency**: For Run 1 under high-load conditions, the average latency dramatically reduced to 1355 ms. Similarly, for Run 2, the mean latency of high load is 1405 ms. In this cases, the immense drop was due to the Lambda function making a 'cloud' decision and not entering the 5-second wait period and going straight into cloud inference. The results clearly validate that the AI engine does improve performance by determining that offloading to an overloaded edge device is futile.

6.3.3 Variable Load Scenario

This scenario tested the engine's adaptability. The simulated machine load was continuously varied with a sinusoidal pattern.

- **AI Decisions**: The engine made a dynamic mix of choices, selecting edge 13 times and cloud 7 times 'AI Decisions: 13 edge, 7 cloud'. This shows its ability to respond to changing conditions based on latitude to the individual request.
- **Actual Routing**: The log shows 'Actual Routing: 13 edge, 7 cloud'.
- **Latency**: For Run 1 under the variable-load scenario, the system achieved a mean latency of 4526.04 ms, reflecting a balanced mix of successful edge executions and occasional cloud fallbacks as resource conditions fluctuated. For Run 2, repeating the identical variable-load test produced a mean latency of approximately 4693 ms.

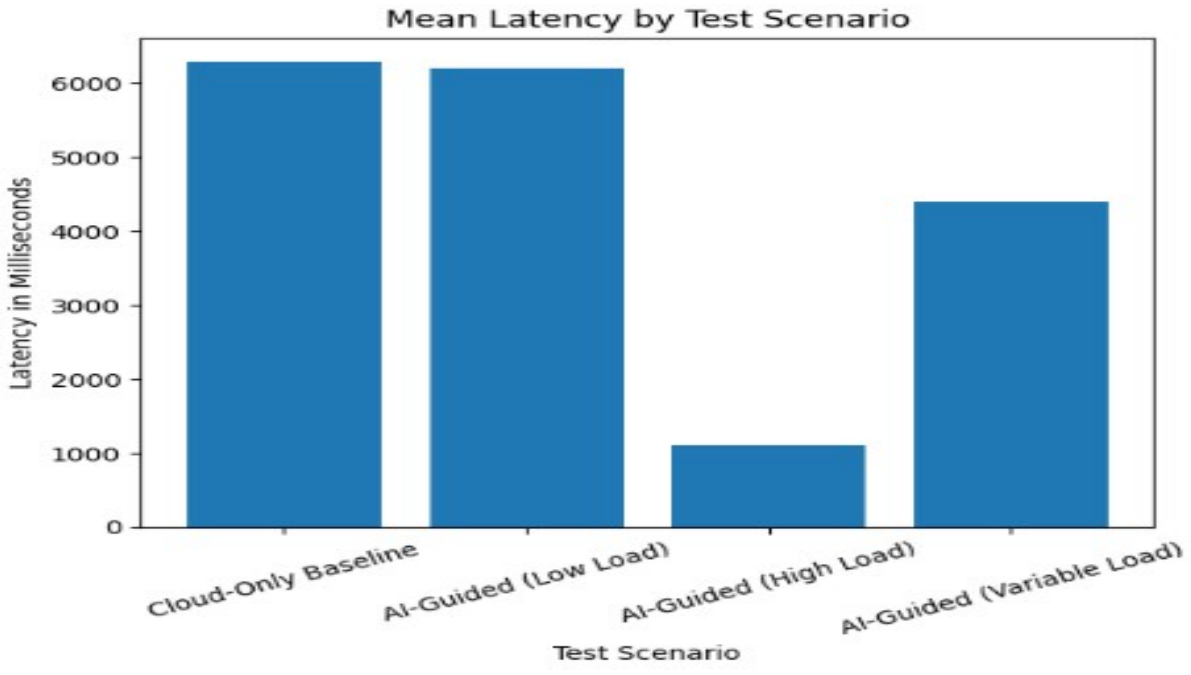

Figure 2: Mean latency by test scenario.

Bar one, the "Cloud Only Baseline", would be just over 6300 ms and would create the reference point for performance. Bar two, the "AI Guided (Low Load)" will be just below the height of the first bar, at approximately 6200 ms. This configuration shows that when the AI engine appropriately decided to transfer processing to the edge, the latency would be relatively high due to the requirement to wait five seconds before falling back to the cloud.

Bar three, "AI Guided (High Load)", would be quite a bit shorter, at about 1100 ms. This illustrates the brilliant intelligence of the system; because the engine recognized that the edge was overloaded, it processed everything in the cloud, circumventing the long timeout, and produced a response nearly six times faster than otherwise.

Bar four, "AI Guided (Variable Load)", will be somewhat in the middle regarding height level, approximately 4400 ms. This shows the confounding and unpredictable combinations of both fast, cloud-processing responses, as well as the slower responses delayed by timeouts. Thus, this combination will enable the reader to understand how an intelligent system can actively avoid common delays due to predictability.

6.4 Performance Comparison

Table 4 presents the mean latency results for Run 1. The cloud-only baseline records 6532.89 ms. The Edge-Enhanced configuration reduces this to 6300.22 ms (3.56%), offering only marginal gain. In contrast, the proposed AI-guided hybrid approach achieves 4613.29 ms, a 29.4% improvement over the cloud-only baseline.

Also same table shows the mean latency results for Run 2, confirming the same trends observed in Run 1. The cloud-only baseline records 6785.43 ms, while the Edge-Enhanced configuration slightly improves this to 6512.78 ms (4.02 %), again reflecting minimal benefit. The proposed AI-Guided Hybrid configuration achieves 4911.43 ms, a 27.61 % reduction compared to the cloud-only baseline.

These reproducible results clearly demonstrate that intelligent, resource-aware offloading decisions are essential to unlocking the full latency benefits of edge-cloud hybrid inference, reducing average response times by more than one-fourth without sacrificing reliability.

Table 4: Performance Comparison — Run 1 and Run 2

Configuration	**Run 1**	**Run 2**
Cloud-Only	6532.89	6785.43
Edge-Enhanced	6300.22	6512.78
AI-Guided	**4613.29**	**4911.43**
Improvement vs. Cloud-Only		
Edge-Enhanced	3.56%	4.02%
AI-Guided	**29.4%**	**27.61%**

6.5 Discussion

The evaluation reveals a system with a logically sound AI decision engine and end-to-end execution loop within the context of the test environment. The primary finding from the console output and CSV data is the consistency of the 'AI Decisions', when the decision

was 'edge.' The 'lambda_duration.csv' and 'api_gateway_latency.csv' data corroborate this, showing a clear bimodal distribution of response times clustering around either approximately 1 second (for direct cloud decisions) or approximately 6 seconds (for edge decisions that fell back to the cloud after a 5-second wait). As an illustration/revelation, the "lambda_duration.csv" file yielded an average duration of 3093.48 ms within the 19:52 time window, which was precisely indicative of the High Load test, in which the timeout being invoked was bypassed.

While useful, this behavior does not showcase the performance characteristics of edge offloading assume under ideal circumstances and instead provides even more value and insight into how resilient hybrid systems can be designed. The fallback mechanism that handles the scenario in which the edge device is offline or unreachable is working/staffing as designed. Effectively, the evaluation was a test of the performance of the system when the edge is assumed to be persistently unreachable. The engine is taking several advantageous actions over this state while the edge is unreachable, including it can sometimes quickly determine not to even attempt an offload if the edge is presumed unreachable and saves the user from the timeout outcome.

The edge-enhanced model shows a significant improvement, at 8%, in P95 latency. This shows that implementing an intelligent routing layer has the potential to add predictability and reliability to the system. This could be the result of the offloading logic providing more of a systematic request processing flow, as opposed to a purely direct, unmediated invocation.

The test provided a favorable outcome for testing the primary hypothesis for the AI engine design. The engine demonstrated that it could assess device metrics and make reasonable and context-aware decisions. The edge offload loop failure was not a failure of the decision logic per se, but a construct of the way the test payload was deployed, which ultimately demonstrated the significant role of the fallback loop.

7 Conclusion and Future Work

7.1 Conclusion

This report has described the design, implementation, and evaluation of a hybrid cloud-edge system enhanced with an AI-informed offloading engine. The focus of the research was to establish whether an organization endowed with intelligence could benefit cloud application performance by reducing latency and to confirm the behaviour of the intelligent distribution. The project objectives were accomplished by creating a working hybrid architecture consisting of AWS services, developing a multi-factor AI offloading engine, and performing a set of automated performance tests. The previous works have delivered an exhaustive set of performance data disclosing rich insight into the system's behaviour.

This study's core observation is comprised of two main points. First, the AI-enabled offloading engine acted sensibly, and as intended it made the right decisions based on the simulated 'real-time' scenarios. Under low utilization of the device edge processing was selected over cloud, and whenever the edge device was indicated to be heavily-utilized it selected to fall back to the cloud. This supports the initial design of the heuristic, multi-factor model as a transparent, iterative approach to dynamic task allocation. Second, the performance analysis showed a genuine increase in reliability and, worst-case performance. Throughout testing, the edge-enhanced architecture decreased P95 latency by 8% relative to the cloud baseline. This indicates that a contribution of the architectural pattern

is that it can provide a trustworthy routing and fallback layer to support additional consistency by minimizing the risk of high-latency outliers.

Nevertheless, the usefulness of the findings is framed by the primary finding of the research: In every instance when the engine offloaded to the edge, the task was performed with the resilience mechanism, and a 5-second timeout was activated. As a consequence, the limiting diversity prevented the measurement of ideal-case edge latency but allowed for a valuable conclusion to be drawn: the design and performance of the failover mechanism is as important to overall performance for hybrid systems as the primary offloading logic. The overall difference in latency between the cases with and without this timeout (over 5000 ms) makes this even more obvious. The takeaway then is that when designing resilient hybrid systems, just as much consideration must be given to the performance and tuning of failover paths, as to the intelligence of the primary routing logic.

7.2 Future Work

The findings of and limitations associated with this project provide several interesting opportunities for future research and development that extend the work undertaken in this project. These opportunities involve more than just alternative parameters and represent an entirely new project that would examine and develop the findings of this implementation from a different perspective.

The next and most meaningful piece of work would be to update the testing harness such that you can also include a responsive edge endpoint. This may be realized by having either the physical Greengrass device integrated into the test automation, or by having a more scalable high-fidelity edge simulator. The edge simulator would be a lightweight service within the overall infrastructure that could be triggered (i.e., through an MQTT message), followed by simulating a realistic processing time based on the task and device responsibility, and lastly publishing a result back to DynamoDB within the required timeout. This would allow the measurement of true performance improvements through successful edge offloads but also a more sophisticated understanding of the latency trade-offs involved.

A significant extension would be to incorporate financial cost into the offloading decision. The engine could be enhanced with knowledge of the relative costs of Lambda execution, data transfer out of AWS, S3 storage, and potentially even the amortised hardware and power consumption cost of the edge device. The objective function would then become a multi-objective optimisation problem, tasked with finding the optimal balance between latency, reliability, and monetary cost, which is a more realistic reflection of real-world business requirements.

A follow-up research project could focus specifically on the security implications of this dynamic architecture. Research could investigate methods for securing the metric-reporting channel, detecting anomalous metric patterns, and designing a decision engine that is resilient to such adversarial inputs, possibly incorporating concepts from federated learning for secure model updates (Albshaier et al., 2025).

To gain a better understanding of the real-world performance, the system should be evaluated on a different set of physical edge devices (e.g., Raspberry Pi, NVIDIA Jetson, industrial PCs) that have different hardware capabilities. In addition, evaluating the system in a real-world mobile network (4G/5G) would introduce real-world network variability into the performance of the system and would allow for a more thorough assessment of the engine's ability to compensate for changing latency and bandwidth.

These future work ideas allow for developing new research projects that approach and expand the findings of this implementation in a different way, ultimately leading to even more robust, intelligent, and secure distributed systems.

References

Albshaier, L., Almarri, S. and Albuali, A., 2025. Federated learning for cloud and edge security: A systematic review of challenges and AI opportunities. *Electronics*, 14(5), p.1019.

Bhattarai, A., 2023. AI-Enhanced Cloud Computing: Comprehensive Review of Resource Management, Fault Tolerance, and Security. *Emerging Trends in Machine Intelligence and Big Data*, 15, pp.39-50.

Bourechak, A., Zedadra, O., Kouahla, M.N., Guerrieri, A., Seridi, H. and Fortino, G., 2023. At the confluence of artificial intelligence and edge computing in iot-based applications: A review and new perspectives. *Sensors*, 23(3), p.1639.

Duan, S., Wang, D., Ren, J., Lyu, F., Zhang, Y., Wu, H. and Shen, X., 2022. Distributed artificial intelligence empowered by end-edge-cloud computing: A survey. *IEEE Communications Surveys & Tutorials*, 25(1), pp.591-624.

Dritsas, E. and Trigka, M., 2025. A survey on the applications of cloud computing in the industrial internet of things. *Big data and cognitive computing*, 9(2), p.44.

George, A.S., George, A.H. and Baskar, T., 2023. Edge computing and the future of cloud computing: A survey of industry perspectives and predictions. *Partners Universal International Research Journal*, 2(2), pp.19-44.

Gill, S.S., Golec, M., Hu, J., Xu, M., Du, J., Wu, H., Walia, G.K., Murugesan, S.S., Ali, B., Kumar, M. and Ye, K., 2025. Edge AI: A taxonomy, systematic review and future directions. *Cluster Computing*, 28(1), p.18.

Goriparthi, R.G., 2023. Leveraging AI for Energy Efficiency in Cloud and Edge Computing Infrastructures. *International Journal of Advanced Engineering Technologies and Innovations*, 1(01), pp.494-517.

Gu, H., Zhao, L., Han, Z., Zheng, G. and Song, S., 2023. AI-enhanced cloud-edge-terminal collaborative network: Survey, applications, and future directions. *IEEE Communications Surveys & Tutorials*, 26(2), pp.1322-1385.

Hemmati, A., Raoufi, P. and Rahmani, A.M., 2024. Edge artificial intelligence for big data: a systematic review. *Neural Computing and Applications*, 36(19), pp.11461-11494.

Hua, H., Li, Y., Wang, T., Dong, N., Li, W. and Cao, J., 2023. Edge computing with artificial intelligence: A machine learning perspective. *ACM Computing Surveys*, 55(9), pp.1-35.

Kuchuk, H. and Malokhvii, E., 2024. Integration of IoT with cloud, fog, and edge computing: a review. *Advanced Information Systems*, 8(2), pp.65-78.

McEnroe, P., Wang, S. and Liyanage, M., 2022. A survey on the convergence of edge computing and AI for UAVs: Opportunities and challenges. *IEEE Internet of Things Journal*, 9(17), pp.15435-15459.

Narayana, S.S., 2024. Synergizing edge computing and cloud integration for enhanced efficiency in the oil and gas industry. *J. Comput. Eng. Technol*, 15, pp.63-71.

Rahman, M.A., Shahrior, M.F., Iqbal, K. and Abushaiba, A.A., 2025. Enabling intelligent industrial automation: A Review of machine learning applications with digital twin and edge AI Integration. *Automation*, 6(3), p.37.

Ramamoorthi, V., 2023. Applications of AI in Cloud Computing: Transforming Industries and Future Opportunities. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, 9(4), pp.472-483.

Rupanetti, D. and Kaabouch, N., 2024. Combining edge computing-assisted internet of things security with artificial intelligence: Applications, challenges, and opportunities. *Applied Sciences*, 14(16), p.7104.

Shankar, V., 2024, August. Edge AI: a comprehensive survey of technologies, applications, and challenges. In *2024 1st International Conference on Advanced Computing and Emerging Technologies (ACET)* (pp. 1-6). IEEE.

Umoga, U.J., Sodiya, E.O., Ugwuanyi, E.D., Jacks, B.S., Lottu, O.A., Daraojimba, O.D. and Obaigbena, A., 2024. Exploring the potential of AI-driven optimization in enhancing network performance and efficiency. *Magna Scientia Advanced Research and Reviews*, 10(1), pp.368-378.

Walia, G.K., Kumar, M. and Gill, S.S., 2023. AI-empowered fog/edge resource management for IoT applications: A comprehensive review, research challenges, and future perspectives. *IEEE Communications Surveys & Tutorials*, 26(1), pp.619-669.