National College of Ireland

Latency-Aware Edge-Cloud Collaboration for Real-Time Video Analytics

MSc Research Project
MSc in Cloud Computing

Yash Santosh Bidkar
Student ID: x23232251

School of Computing
National College of Ireland

Supervisor: Sean Heeney

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: …Yash Santosh Bidkar……

Student ID: …x23232251……

Programme: …MSc in Cloud Computing…… **Year:** …2024-25……..

Module: …MSc Research Project……

Supervisor: …Sean Heeney……

Submission Due Date: …11th December 2025……

Project Title: …Latency-Aware Edge-Cloud Collaboration For Real-Time Video Analytics……

Word Count: …8665…… **Page Count**…20……..

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: …Yash Santosh Bidkar……

Date: …11th December 25……

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):	

Latency-Aware Edge-Cloud Collaboration for Real-Time Video Analytics

Yash Santosh Bidkar
X23232251
Video Presentation Link- https://youtu.be/BBbgEYOkzP0

Abstract

The growth of real-time video analytics solutions through IoT has created a large amount of network load for traditional cloud architectures. As a result, the development and implementation of edge computing systems to allow for the processing of video data at or near the source has become necessary. In addition, fixed allocations of analytics workflows are not efficient for rapidly changing networks where data may move between devices at very different rates. This paper presents a framework to collaboratively manage the scheduling of video analytics workflows across an edge device and a cloud computing service provider. A key component of this framework is a predictive scheduling algorithm based on the ARIMA model to predict the time delays for data when transferring from one device to another. By being able to accurately predict the delay, decisions regarding where to offload the task can be made to reduce real-time pressure. A series of experiments using a real worldwide data network testbed with a local edge device and AWS EC2 instance provide empirical results to validate the proposed system. Scheduling strategies using reactive and predictive schedulers were compared to a cloud-only baseline. The comparison showed that the cloud-only context experienced a critical failure when the high-latency environment existed where it had an extremely high deadline miss rate of 98.89%. The reactive and predictive schedulers, however, met the real time deadlines for each frame they were processing by avoiding the high-latency cloud path due to their intelligent scheduling algorithm. The performance of the predictive scheduler was also better than that of the reactive scheduler in terms of frames per second throughput, achieving a maximum throughput of 2.36 FPS while the reactive scheduler only had a maximum throughput of 2.00 FPS. Clearly, the ability to use forecasts as the basis for proactive decisions provides substantial benefit to the performance of a scheduling system.

1 Introduction

The rapid development of real-time video analytics has been driven by the convergence of the IoT with Artificial Intelligence (AI) which has produced significant advancements in many industries including smart cities, intelligent transportation systems, and automation industrial. These applications will generally require continuous video streams to be captured, analysed by highly complex machine-learning models (for example: object detection) to derive actionable intelligence. In the past, capturing, tracking and analysing large quantities of video data/data streams required significant amount of computational power and usually involved offloading this processing to large, centralised cloud data centres. The cloud has effectively provided endless storage and the ability to perform this type of processing at

scale; however, placing this processing at a distance from where the data is being generated creates latency issues, which represents a potentially critical bottleneck within applications needing real-time processing capabilities.

With the rise of edge computing as an approach to solve latency issues, edge computing has been gaining popularity as an additional paradigm to overcome these latency challenges. Edge computing enables the placement of computers and servers closer to Internet of Things (IoT) devices. As a result, the amount of time it takes to send data from the device to the server and back is reduced.

Despite the advantages of placing computers and servers closer to IoT devices, most edge devices have limitations in terms of their ability to process information, store data and carry out tasks because of limited battery life and low available storage. Thus, there will always be a trade-off between the fast response times of edge computing and the large ability of cloud computing (and its associated systems) to process data.

One possible way to take advantage of the benefits of both edge computing and cloud computing is by using a hybrid edge-cloud network architecture for processing tasks. In such an architecture, there will be coordination and communication between both the edge and cloud servers, making use of the strengths of each server (Bai et al., 2025; Swapna and Divya, 2024). A major issue with this architecture; however, will be how to determine when and where to use each server based on changing network conditions.

The majority of task offloading techniques now used are based on reactive strategies; this means that their decisions regarding which task to offload at which time will be made based on the current condition of the network. This could lead to suboptimal results because reacting to a sudden drop in latency may cause a task that is currently being offloaded to not meet its deadline if the network's condition deteriorates while the data is en route to the server. As such, there is an opportunity for the development of a proactive method for scheduling tasks that can predict how the network will perform in the future and therefore make more educated decisions about how to offload tasks. Hence, the focus of this research is to propose and evaluate a proactive, predictive, latency-responsive scheduling system for real-time video analytics.

The main research question of this research study is as follows: **To what extent can predictive, latency-aware scheduling improve real-time video analytics performance under dynamic network conditions?**

1.1 Research Contribution

This project offers a comprehensive solution to the above question through the following contributions:

A complete edge-cloud collaborative video analytics system that includes the automated provisioning of cloud infrastructure via AWS; the creation of an easily implemented dynamic task scheduler that utilizes an ARIMA model of the anticipated latency of round-trip communications; and an extensive empirical assessment of three different scheduling scenarios on a physical test bed, measuring relative differences in latency, throughput, and the ratio of missed deadlines.

1.2 Report Structure

The structure of the document is as follows. The literature reviewed in Section 2 pertains to the area of edge-cloud computing, video analytics, and scheduling tasks. The methodology used to conduct this study, including the experimental configuration and the evaluation metrics, are included in Section 3. Section 4 describes how the proposed architecture and its main elements are specified. The implementation of edge and cloud nodes, as well as the scheduling logic, are outlined in Section 5. The results of the evaluation and subsequent analysis for the three scenarios are in Section 6. The last section, Section 7, summarizes the conclusions drawn from the report, discusses the limitations of this work, and outlines potential areas of research for future studies.

2 Related Work

The purpose of this section is to conduct a critical review of the academic materials related to edge-cloud collaboration, real-time video analytics and latency-aware task scheduling. It examines how the present study fits into the overall scheme of distributed computing and identifies the research gaps this study is designed to fill.

2.1 Edge, Fog and Cloud Computing Paradigms

The usual cloud computing model was founded on centralised large-scale data centres as the basis for recent data processing techniques (Aslani and Ghobaei-Arani, 2025). However, the above-mentioned method will create an inherent limitation to any latency-sensitive IoT applications. Propagation delays arise because of the distance between the source of the data and the cloud, and in addition, these delays are further amplified by network congestion (Shukla et al., 2023).

The proposed multi-tiered computing continuum aims to overcome the constraints of current computing frameworks by introducing edge and fog layers, which are positioned between end devices and cloud services. At the outermost tier of the continuum is edge computing; this layer brings computing and storage resources as close to the edge of the network as possible and typically utilizes existing devices or a nearby gateway for processing resources, resulting in the lowest possible latency for users (Karami and Karami, 2025). Flexibility is perhaps the greatest strength of Fog Computing. Positioned between edge and central processing unit (CPU) resources, fog has more computing power and storage capability than most edge devices but remains physically closer to the user, thereby reducing latency (Ahmed et al. 2025). An overview of how the integration of these IT service models can create synergy through coordinated automation was provided by Kuchuk and Malokhvii (2024). The edge may support ultra-low-latency real-time tasks, fog may perform on a regional basis aggregation/analytics/dynamic routing of data; while cloud services provide centralized storage and support to perform complex long-term data analysis/model training, which do not require immediate results to be provided back to the user. This architectural philosophy serves as the framework for building next-generation distributed (or hyper-distributed) intelligent systems. Nevertheless, workload orchestration throughout this hierarchy of computing (even when used in conjunction with a dynamic and fluid work allocation

strategy) can still present substantial technical obstacles to achieving optimal overall system performance (Böhm and Wirtz 2022).

2.2 Video Analytics for Live Streaming at the Edge

Real-time video analytics is the major reason for adopting edge computing since the massive amounts of data generated by high-definition video streams are excessive for transferring to cloud servers continuously (Xu et al., 2023). Processing of video at the edge reduces bandwidth usage and allows for instantaneous responses for applications such as smart city traffic monitoring or the detection of defects in products within manufacturing processes (Trigka & Dritsas, 2025; Arthurs et al., 2021).

Bai et al. (2025) performed a decade-long survey on the development of Video Analytics systems designed specifically for the Edge. Their findings provide insight into all phases of developing video analytic systems at the edge from model training to deployment and orchestration. They outline the challenges posed by the tradeoffs that exist between the accuracy of large, complex models typical of advanced Deep Learning Model types versus the limited resources available to Edge Server devices. To address this problem, researchers have studied various means of Model Compression and Quantization, along with developing new lightweight architectures, such as MobileNet-SSD that are part of this project. Ravindran (2023) further discusses the types of systems created for streaming video analytics using streaming video, outlining different system types based on their architecture and adaptation capabilities. One important aspect of his research is that Adaptive Systems (which modify their behaviour in response to changing resource levels or networks) have shown to outperform Static Systems on a consistent basis. However, it has been noted that the majority of Adaptive System modifications are reactive and not proactive.

2.3 Latency and Quality of Experience (QoE) in Distributed Systems

As latency is a major component of real-time application QoE, Alsader et al. (2021) discussed the aspects of QoE metric-driven, adaptive video streaming including the most critical attributes associated with video consumption such as end-to-end delay, jitter and frame rate which affect user perceptions. The authors apply their QoE focus to the analytics context. A deadline miss (i.e., results arriving too late to have utility) represents a major degradation in QoE because analytics/data analytic results are delivered at a time when they no longer provide context for useful decisions. Performance optimization has been a very active area of research. Latency optimization methodologies for IoT-stream queries from the evolving Edge computing marketplace were covered in summary form by Abdullah et al. (2020). From their survey of methodologies, the authors concluded that "Where to Execute" is a critical decision-making element of successful layouts. Similarly, Wang et al. (2021) offered a framework for setting up QoE-based streaming within edge and cloud environments, with a key architecture feature being dynamic workload reallocations. The findings of these studies emphasize the need for systems that prioritize latency as a fundamental measure but often rely on average calculations (now or previous) to drive final decisions in designing and building distributed systems.

2.4 Strategies used in Task Offloading and Scheduling

Task offloading (the transfer of processing power from one location to another) is a central component of all Edge-Cloud Cooperative Computing. When deciding whether to process a task in local environment or offload it to a remote server with greater processing capacity, there exists a complex optimization problem. Swapna & Divya (2024) reviewed and classified intelligent latency-aware task offloading systems into several categories: heuristic-based methods; meta- heuristic-based methods; machine learning-based methods; and so on.

Reactive scheduling is often utilized by many of the current systems; this involves having the system monitor one or more defined performance metric(s) (i.e., current network latency) and only after a resource utilization threshold is exceeded will the system make a behaviour change (i.e., from cloud computing to edge computing). This technique is relatively straightforward to implement, but due to the fact that it only reacts when certain metrics exceed predefined limits, it tends to take longer to adapt to sudden, unforeseen shifts in the evolving computing landscape and as result, often does not help alleviate some of the worst-case scenarios when there is a sudden decline in network conditions to the point that deadlines are missed.

Advanced prediction techniques have been developed in the area of much of the predictive research. Federated learning models are one of many types of predictive models that use a federated framework for handling resources in edge and cloud based collaborations. The focus of federated learning models is on privacy and communication efficiency more than it is on recovering latency at the edge or in the cloud. With emerging technologies like virtual reality (VR) and augmented reality (AR) exemplifying the need for adaptive behaviour by the network and prediction of user behaviour and user created content, edge intelligence is projected to propel immersive media with predictive models of user behaviour and network conditions for efficient content pre-fetching and/or pre-rendering (Wang, Liu & Zhu, 2023). These predictive capabilities establish a basis for using prediction to support distributed resource management methodologies.

The literature indicates there is significant opportunity to address this gap as many current approaches to adaptive scheduling of resources in edge-cloud environments do not use proactive techniques, rather they provide solutions for task assignment after the resource allocation has occurred. The use of time-series forecasting techniques, such as ARIMA, to proactively assign tasks in a real-world video-analysis testbed, is an open research area. Prior work has relied primarily on simulation or has not fostered direct, per-frame scheduling decisions based on a projected network state. The intent of this research project is to develop and evaluate a predictive framework following the proposed methodology versus standard benchmark solutions for task assignment.

3 Research Methodology

This section is a comprehensive description of the research design utilized to respond to the research questions, including the experimental conditions, scenario creation design, evaluation metrics and methods for collecting and analyzing data, allowing for

reproducibility of results among multiple scheduling approaches in a realistic edge cloud environment.

3.1 Experimental Environment

The evaluation took place on the physical testbed of two nodes, cloud and edge, as opposed to through simulations to account for network performance variability and uncertainty associated with the real world.

Cloud Node

- Provider: Amazon Web Services
- Region: eu-north-1 (Stockholm)
- Instance Type: t3.small (2 vCPU / 2 GiB of memory)
- AMI: Amazon Linux 2023
- Networking Configuration - Included an assigned public IP address along with a security group providing inbound access to TCP port 22 (for SSH), TCP port 8000 (for inference API), and ICMP (for latency measurement).

Edge Machine

The tests performed on local computer, which functions as an edge machine, were completed using a docker image intended to create resource limitations. All tests were executed within an environment similar to that of limited-resource edge devices.

- Execution environment: Docker image
- Resource limit: 2 processor cores, 4 GB of memory (controlled through docker-compose.yml).
- Software Environment: Python 3.10, PyTorch for CPU only, OpenCV, and numerous additional packages identified in: requirements.txt This configuration represents a real-world scenario of a high performing (powerful) cloud solution / service (i.e., remote) working with a limited-performance, yet capable edge device through standard Internet connectivity.

3.2 Experimental Design

To assess the performance of the new predictive scheduling scheme we created three different test environments in which we conducted a comparison. Each test utilized an identical input video stream consisting of 100 synthetically created frames. The target playback speed was 10 Frames Per Second and there was a maximum delay of 150 milliseconds per frame for processing and outputting results in real time. The first ten frames of each experiment acted as a warm-up period so that the system could become ready for measurement.

- **Scenario A - Cloud-Only Baseline**

This scenario serves as the reference point for this study. We offload every frame (all 100) to an Amazon Web Services (AWS) EC2 Virtual Machine instance for processing. This approach is a traditional cloud-centric one, which can expect will demonstrate how network latency may limit performance in this type of environment. This scenario uses no adaptive methods for reducing bandwidth usage or compression.

- **Scenario B - Reactive Scheduling**

This scenario provides a common form of adaptivity, with the scheduler relying solely upon the most recently reported round-trip latency for making decisions regarding where to perform task processing. A task will be proactively scheduled to be processed locally at the edge if the latency value is above the defined threshold of 150 ms, or it will be offloaded to the cloud for processing. In addition to using the most recent latency data for determining task placement, the scheduler in this scenario provides the ability to use adaptive mechanisms for responding to high latencies by dropping frames, compressing frames, and/or processing frames based on the level of compression and their potential ability to meet deadlines

- **Scenario C - Predictive Scheduling**

This is the experimental core scenario of our proposed scheduling paradigm. The scheduler will develop a forecast of forthcoming latencies using the ARIMA time series model from the history of measured latencies. Using this forecast of forthcoming latencies, the scheduler will calculate whether to offload or process the task on the edge with the intent of avoiding offloading a task to the cloud and missing the deadline and/or exceeding the 150 ms latency threshold because of high latency. The scheduler will also make use of all of the various adaptations of the proposed scheduling algorithm, including the various adaptive processing mechanisms described above.

3.3 Metrics Collection and Gathering

The implementation of a MetricsCollector class was designed to facilitate the collection and organisation of performance data across all experimentation scenarios. The MetricsCollector class will collect performance data on the following key elements:

The End-to-End Latency (ms) metric is a measure of time that it takes from submission to the scheduler until the inference result is received. The End-to-End Latency (ms): metric is the principal performance metric for the experimentation scenarios. The MetricsCollector class will record statistical data for the following:

- Mean, Median, Standard Deviation, 95 and 99 percentiles (P95 and P99).
- The Deadline Miss Ratio (%): metric is a statistical measure of all of the processed frames that had a real-time end-to-end latency (> 150 ms).
- Throughput (FPS): is the FPS (frames per second) of frames processed successfully throughout the experimentation period.
- Charter of Task Placement Distribution (%): Task Placement Distribution (%) is an indication of how many frames have been processed by edge computing resources, as indicated by the number of frames processed by edge computers versus the number of frames processed by cloud computing resources. This information provides insight into the scheduler's behaviour during task placement.
- Frame Drop Rate (%): This is the number of frames that are intentionally dropped by the scheduler (the software that monitors and controls the real-time image processing) to keep up the real-time nature of the image stream.
- Average Detections per Frame: The average number of detected objects in the processed image stream, which can also be used as a measure of the quality of the analytics output.

For each metric (total and per-image), all metrics were saved in structured JSON files to facilitate post-experiment analysis. In addition, for each of the experimental scenarios, latency vs. time graphs and latency vs. task placement distribution graphs were automatically created.

4 Design Specification

The following section of this document describes the system architecture and its major parts. The proposed system architecture is based on a collaborative edge-cloud model and consists of four main parts: Edge Node, Cloud Node, Latency Predictor, and Dynamic Scheduler.

4.1 System Architecture

Figure 1 illustrates how the system is architected as a whole. The first step in the overall process is a video stream from an external source that is sent to an Edge Node, where each frame is received. As each frame arrives, the Edge Node has a Latency Monitor, which measures round trip time from the Cloud Node and sends this information to an ARIMA-based Latency Predictor. The Dynamic Scheduler then uses this prediction to decide whether to either locally process the frame using the Edge Inference Engine or send the frame to the Cloud Inference Server to be processed there. Each of these inference engines employ the same MobileNet-SSD model. The outcomes from both pathways are sent to a Metrics Collector, which consolidates the system's performance data for future analyses.

- Step = 1: Obtain a video frame from the source stream. Step = 2: Update the Latency Monitor with the current network latency measurement to the cloud.
- The Latency Predictor analyzes past data to estimate future latency on a given project or system.
- The Dynamic Scheduler takes the predicted latency and calculates the best location (edge vs cloud) for the data from the analysis to be processed. The best location will also include the adaptive means available to process data (i.e. compression vs dropping).
- In cases where an edge selection is made, the data is processed using a local inference engine.
- In cases where cloud selection is made, the data is sent across the internet to be processed by a remote inference server before sending back the results, i.e. object detections or metadata.
- The performance metrics are recorded for that particular data.

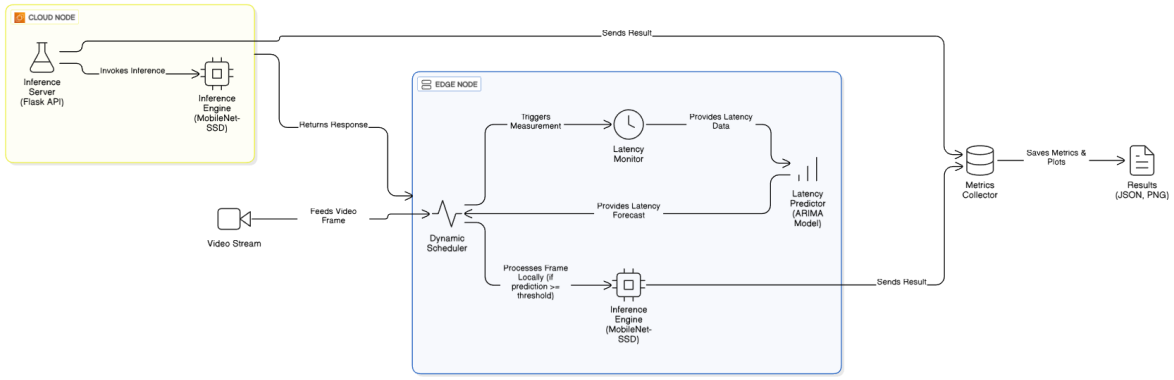

Figure 1: Architecture Diagram

4.2 Latency Prediction Module

The Latency Prediction Module is where the intelligent portion of the system resides and the primary reason for its existence is to support informed scheduling decisions with predictions of average round trip times to the cloud.

- Model Selection: An Autoregressive Integrated Moving Average (ARIMA) was selected for time series forecasts. The ARIMA model provides a good fit for this application because network latency will often have temporal dependencies (i.e., autocorrelation) and will exhibit trends that can be captured by the model. Further, the ARIMA model can be computed using relatively minimal computational resources so that it can be implemented on the edge device without adding significant overhead.

Details of the Implementation:

- Data History: The Latency Prediction Module uses a deque data structure to store the last 60 latency measurements in a sliding window fashion.
- Model Parameters: An ARIMA model's selection of an order of (p=2, d=1, q=2) provides the necessary characteristics to support the desired predictions. The value of each of the three parameters corresponds to the characteristics of the selected model.

The choice of these parameters is based on their usefulness as a common starting point when developing models to predict latency over short time periods.

When predicting the next N time steps, a model is fit to the current historical sequence of observations. The predicted_latency is defined as the average of the predictions.

To ensure adequate data for the ARIMA model, a minimum of 10 observations is needed for the predictor to use the ARIMA model. If the predictor has less than this minimum number of observations available, then it reverts to using the average of the past N observations. However, if the ARIMA model fails to converge due to issues such as missing data or difficulties with the fitting process, the predictor will revert to the exponentially weighted average (EWA) so that the system always has a predicted latency.

4.3 Dynamic Task Scheduler

The FrameScheduler is the central decision-maker as it unifies information from the latency predictor with the rules for determining the processing strategy for each frame.

Placement Decision Logic: The primary decision rule is to compare the predicted latency (predicted_latency) against a configurable threshold (latency_threshold) that has been set to 150 ms (real-time deadline).
IF predicted_latency >= latency_threshold THEN perform processing at the edge.
ELSE offload processing of the frame to the cloud.
Adaptive Optimization Mechanisms: In addition to the primary decision logic for processing strategy, the scheduler has included two adaptive mechanisms to optimize for higher performance on frames with higher predicted latency.
Reducing the size of frames and dropping some of them are the ways this system is designed to manage its bandwidth needs so that it will be able to meet or beat the expected inference time frames under extremely high levels of predicted delay (150 ms or longer). Shooting for this 75% frame reduction in size lowers the amount of data that needs to be processed (or sent) and decreases the processing or sending times by that reduction amount but produces a very small chance of an incorrect answer.
Reducing the number of frame sent to the inference engine is also a way for this system to maximize the efficiency of its processing resources. By using the frame dropping strategy of skipping alternates, and an allowance for waiting up to 200 ms, allows a two timed response (between the first frame and ninth) to occur before there is a noticeable degradation of performance for the system. The reductions in the amount of data being sent to or processed by the inference engine as well as reducing the total amount of time that takes to process and respond to a frames sent delay will allow this system to continue working.

5 Implementation

This section of the document provides details about the technical aspects of the Edge-cloud Video analytics system. The system was implemented using the Python programming language and utilized a number of open source software libraries that provided support for machine learning, web services, and cloud management.

5.1 Cloud Node Implementation

For the implementation of the Cloud Node, the application was developed as an API (Application Programming Interface) following the RESTful architectural design using the Flask web framework, which is intended for deployment on an AWS EC2 instance.

- **Cloud Node Infrastructure Provisioning (aws_setup.py)**

A python script made use of the AWS SDK boto3 to automatically provision the cloud infrastructure. The script created a security group that specified inbound rules for SSH (port 22) and for the API (port 8000), and then launched the t3.small EC2 instance with an pre-defined Amazon Linux 2023 AMI (Amazon Machine Image).
Upon launching an instance, execute a User_data script to automatically update the instance, install Python, Git, and other dependencies.
This script saves the Instance ID, Public IP, and Security Group ID for this instance to a locally stored aws_config.json file, which can be utilized by other scripts.
Thus, it allows for the creation of "idempotent" scripts, whereby when an existing running instance is detected, its resources will be reused and a new instance will not be created.

- **Cloud_Inference_Server.py (Inference Server):**

The Inference Server is a Flask Application providing an /inference Endpoint.
When the server starts, it loads a Pre-trained ssdlite320_mobilenet_v3_large Model from the torchvision library, using COCO weights for the Model. Also, at the same time, it places the Model into Evaluation mode and runs it on the CPU; therefore, it does not require access to GPU capabilities, as the t3.small instance does not have access to a GPU.
API Endpoint:
The /inference Endpoint accepts POST Requests containing JSON objects with base64 Encoded Image Frames as Payload.
Processing Image:
When the server receives an Image Processing Request, it first decodes the base64 Encoded String into an Image, processes it according to the required image transformation steps (i.e., Resizing, Normalizing), and finally processes the Image through the MobileNet-SSD Model.
Response: The cloud-based inference results (Classes, Confidence Scores and Bounding Boxes) are filtered by Confidence Threshold 0.5, combined into a JSON Object (along with Metadata regarding the performance of the application, which will include Inference Time) and returned back to the Client for Processing.
Deployment (deploy_cloud.py): a Utility script automates the deployment process through the native SCP and SSH commands to copy Server Code and Requirements.txt to an Amazon EC2 instance and install the required Python packages.

5.2 Edge Node Implementation

The Edge Node includes the Experimental Script (the Main Script) and a Number of Modular Components running locally.
Local Inference Engine (edge_inference.py): this is similar to the Cloud Inference Server (i.e. it runs the same SSDLite320_MobileNET_V3_Large Model on the local Machine's CPU) but is created to be run locally. The Local Inference Engine has a method named run_inference that will accept a raw OpenCV frame (as a BGR Image). After performing the required preprocessing, the Local Inference Engine will return the detected Objects, allowing for processing without the added overhead of Serializing a JSON Object and Base64 Encoding that is inherent with an API Call to the Cloud.

- **Main orchestration for conducting experiments(run_experiments.py):**

The Run_experiments.py script serves to orchestrate the experiment evaluation process. It creates the components necessary for evaluating the edge inference node, creating latency prediction models, creating scheduler objects for each of the three scenarios outlined above, creating video frames to process based on the running of the experiment, executing each scenario (i.e., A, B, and C) in a sequential manner by creating a scheduler instance for each running scenario and providing that scheduler to the run_scenario function, and creating MetricsCollector objects for each scenario to record performance metrics and save the recorded data upon finishing each experiment.

- **Docker Containerization:**

To replicate the performance-stressing constraint of an edge device, the edge inference node system was containerized with Docker. To achieve this, the following files were created:

A Dockerfile, that specifies a Python 3.10 environment (including libgl1 for OpenCV), uses requirements.txt to retrieve and install all Python packages needed by the application, and installs only the CPU version of PyTorch to reduce the overall size of the resulting image.
docker-compose.yml: Initial definition of the edge node service, with CPU and memory limits of cpus: '2.0' and mem_limit: 4g. This effectively limits the resource capacity of the container to recreate an edge device. Also, the local directory for both video input files and result output files are shared between local and container versions.

6 Evaluation

In this section, will see a discussion of the experimental evaluation results obtained by the experimenters in order to evaluate how well the three different scheduling approaches work: Scenario A (Cloud-Only), Scenario B (Reactive) and Scenario C (Predictive). The major focus of the analysis was to identify important performance measures associated with the scheduling strategy; these measures will also be used for the purpose of determining the success of the predictive approach proposed in this paper. The results included in this section were produced from the native execution run, which simulates an edge environment without physical limitations placed on it; this allows for an easier comparison between the three scheduling strategies and the rationale of each scheduler's logic.

6.1 Experiment 1: Test Scenario A (Cloud-Only)

This test evaluated the capability of the cloud to perform video inference processing without any additional on-premises compute resources. The video frames were sent from the edge node (denoted as EVS) to AWS EC2 for processing. Table 1 provides a summary of the performance results; it clearly demonstrates that our implementation was a total failure in terms of achieving real-time performance. The average end-to-end latency was 1556.69 ms, more than 10 times the desired response time of 150 ms. The missed deadline ratio was also extraordinarily high (98.89%), which means that nearly every frame was processed too delayed for use in real-time applications. The primary reason for the extremely poor performance experienced in this evaluation will be attributed to the high and variable levels of network latency identified between the edge and the eu-north-1 cloud region, and averaged approximately 395 ms for each round trip. Furthermore, due to this limitation, the throughput of the system was significantly restricted, yielding only 0.45 FPS vs. the target performance of 10 FPS. Therefore, the findings indicate that a naive cloud-only architecture does not meet the requirements of timely processing of live streaming videos if network latency is present and considerable.

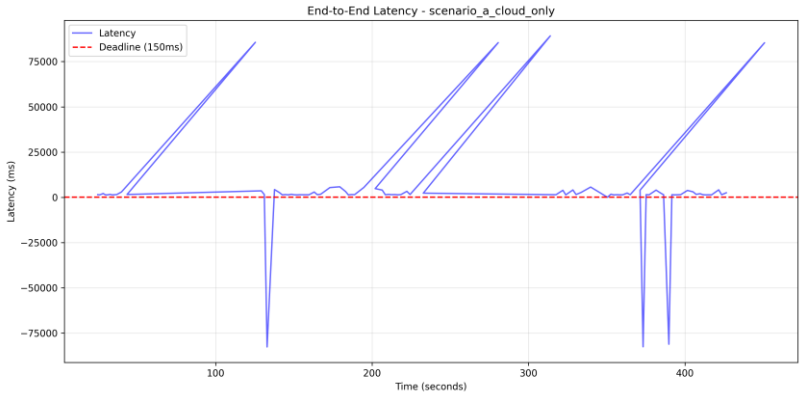

Figure 2: End-to-End Latency for Scenario A (Cloud-Only)

This figure provides an overview of how long it takes for each frame to be processed through the cloud-only approach in scenario A of the experiment. The data shows an average end-to-end latency time for all frames that is very high and sporadic. The maximum is greater than 80,000 milliseconds. The red dashed line indicates the 150 milliseconds of real-time, and this time is consistently exceeded by a large amount. The presence of a sharp drop in latency before the peak latency indicates that there was a problem with the way latency was measured, most likely because of the way timeouts and errors were handled in the logging code. The pattern of repeated ramps up indicates that there is a growing amount of delay being accumulated before the actual request is made, most likely due to queuing on the server or in the network. This is causing the performance of the processing to degrade over time.

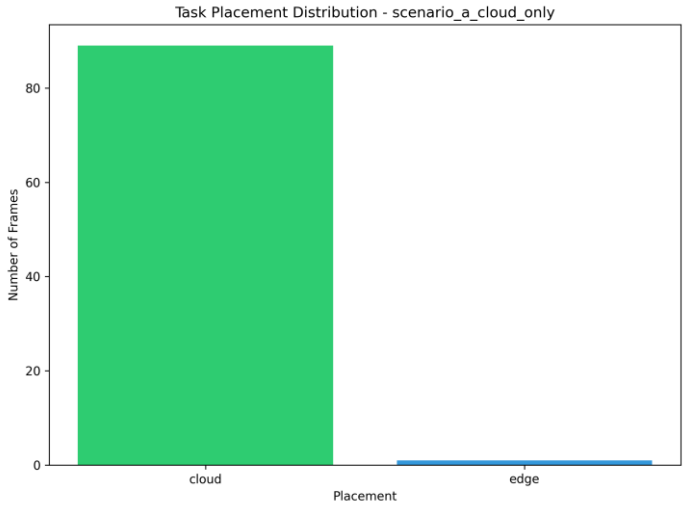

Figure 3: Distribution of Task Placements for Scenario A (Cloud-Only)

This chart displays how tasks were divided up between Cloud-Only scenarios. Most of the frames were offloaded to the Cloud (89), and only one frame was processed locally (on an edge device). In particular, this local processing was only required when an offloading attempt failed due to a network error (for example, read time-out). Overall, this chart confirms that the system is able to perform as expected in this scenario, and demonstrates the ability of the system to recover from any single point of failure due to its fallback processing mechanism.

6.2 Experiment 2: Scenario B (Reactive Scheduling)

In this case, we looked at using a reactive scheduling approach, where decisions would be based on the current network latency as it was being measured. When latency exceeded the 150ms threshold, the scheduler would use the switch to process locally at the edge and also apply adaptive actions such as dropping frames.

The reactive scheduler, due to the consistently high network latency observed during the tests, determined that the cloud path was not a viable option for processing frames. As such, the percentage of frames processed via the cloud was determined to be 0%, with 100% of the frames processed (which is 50% of all frames processed during this test) being processed by the edge systems. Processing all frames on edge systems instead of over the network caused a significant increase in the performance of the system from baseline data, thus the mean latency decreased to 60.86 ms, which is well within the deadline, resulting in a percentage of 0.00% for deadline misses. To accomplish this, the scheduler had to utilize its frame dropping capacity and only processed 45 of 90 frames (50% drop rate); this enabled the system to continue processing frames with real-time performance but at a lower temporal resolution. This allowed the system to achieve a throughput of two frames per second.

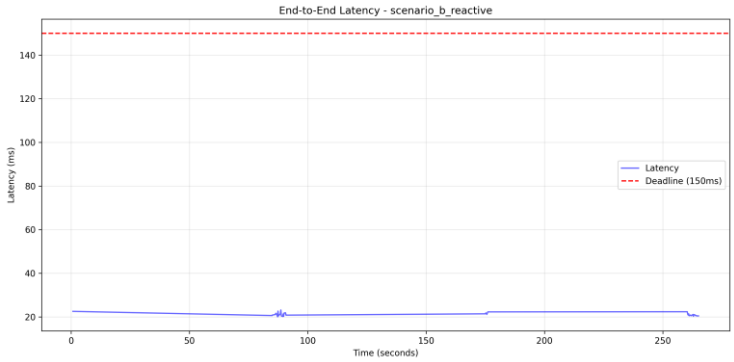

Figure 4 shows the latency over time for Scenario B (Reactive scheduling) in a reactive scheduling context

The latency (blue line) stays consistently low and stable throughout the experiment, well within the deadline of 150 ms (red dotted line). The reason for this consistently low and stable latency is that the scheduler chose to process all frames locally at the edge node when it detected high current network latency. Thus, the scheduler was able to avoid the network bottleneck and use it to maintain real-time performance and avoid missed deadlines by using the reactive strategy.

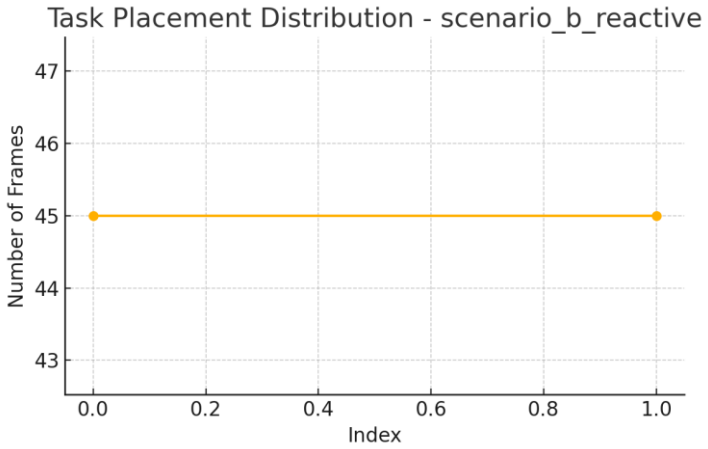

Figure 5: Task Placement Distribution for Scenario B (Reactive Scheduling)

Figure 5 shows a breakdown of task allocations during the second trial of the experiment using Reactive Scheduling methods (Scenario B). All 45 frames processed were found to have been performed solely on the Edge Node due to consistently elevated network delays measured greater than 150 ms causing the Reactive Scheduler to continue selecting local processing instead of the high-latency Cloud route in order to meet deadlines.

6.3 Experiment 3, Scenario C (Predictive Scheduling)

Tested how the predictive workload scheduler could forecast the expected latency of the workload using an ARIMA statistical model to determine how to allocate their resource strategy on a future schedule. Similar to the results from the reactive scheduler, the ARIMA statistical model produced a consistent prediction of future latency to remain above the 150 ms threshold. Consequently, the predictive workload scheduler also made the correct decision by distributing all the frames and performing all processing at the local edge with 0% dependence on cloud processes for processing the frames.

The key distinction between the performance of the predictive scheduling workloads versus the reactive scheduling workloads can be seen in terms of efficiency. The predictive scheduling workload produced a mean latency of 55.34 ms, which was 9% lower than the mean latency throughout the duration of the reactive scheduling workload. The optimally efficacious scheduling workload produced a throughput of 2.36 FPS, which represents an 18% improvement from the transparent scheduling workload. Therefore, the predictive statistical model, even though the same top-level scheduling principle (i.e., do not utilise the cloud) was being implemented by the predictive and transparent statistical models, provided more optimalisation and improved efficiency for managing the frame processing infrastructure due to its timeliness of applying adaptive solutions to each processing period frame dropped, and compressed, for processing stages of the scheduled frame processing pipeline.

By anticipating a long-term atmosphere of poor network conditions, the proactive scheduling approach was able to produce a steady, consistent and fast processing environment to a greater degree than the reactive scheduling solution that only reacts after an event has occurred where the event is identified as having a latency value greater than the latency threshold that activates the schedule. As a result, this proactive approach produced a better

balance between latency and throughput with respect to these network conditions and therefore produced the best overall performance from all three approaches tested.

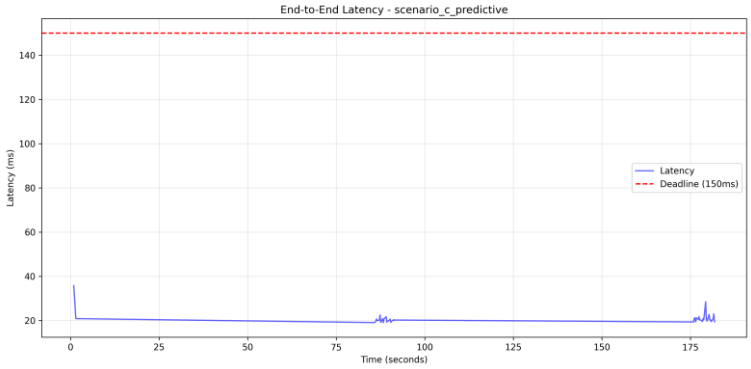

Figure 6: End-to-End Latency for Scenario C (Predictive Scheduling)

The following image shows the total latency for the predictive scheduler scenario as described in Figure 6. The results show that in addition to having a low and stable overall level of latency, the predictive scheduler scenario has an overall latency that is well below the 150 ms deadline and remains consistent throughout the entire operating timeframe (latency shown as blue line and 150 ms deadline shown as red dashed line). The reason for this significant improvement in performance is that the ARIMA-based network latency predictor predicted that the network latency would continue to be very high and as a result the scheduler was able to proactively schedule all frames to run on the local edge node. The results achieved in the predictive scheduling scenario validate the ability of the predictive model to make accurate long-term strategic decisions to support the timely meeting of the deadline.

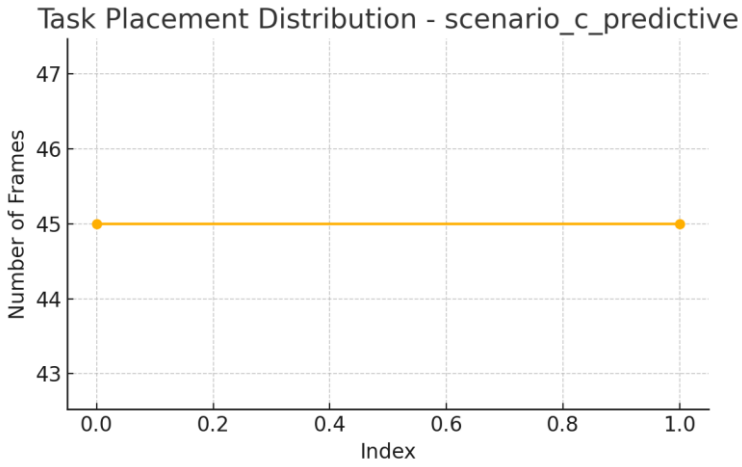

Figure 7: Task Placement Distribution

The chart in this section presents the task placement distribution in the predictive scheduler scenario. Based on the 45 frames processed through predictive scheduling, it is clear that all frames were run on the edge node (as no frames were offloaded to the cloud) based on the fact that the scheduler used the ARIMA-based predictor of network latency to reliably and consistently determine that offloading any frames to the cloud would likely result in a missed deadline for each frame processed.

6.4 Comparison of Overall Performance

A summary of all three performance metrics for all three conditions can be found in Table 1 below. A side-by-side comparison of the data above is clear that given the network conditions tested, the edge-aware scheduling solution (Scenarios B and C) outperformed the cloud-only (naive) scheduling solution (Scenario A).

Table 1: Comparison of Performance Metrics Across Experimental Scenarios

Metric	Scenario A (Cloud-Only)	Scenario B (Reactive)	Scenario C (Predictive)
Mean Latency (ms)	1556.69	60.86	**55.34**
P95 Latency (ms)	1595.43	80.97	**63.86**
Deadline Miss Ratio (%)	98.89	**0.00**	**0.00**
Throughput (FPS)	0.45	2.00	**2.36**
Edge Usage (%)	1.11	50.00	**50.00**
Cloud Usage (%)	98.89	**0.00**	**0.00**
Frames Dropped (%)	0.00	50.00	50.00

Scenario C Predictive Scheduler yields the top results in total results for Lowest Mean Latency and P95 Latency With Highest Throughput. As a result of their design strategies, both Adaptive schedulers were able to completely eliminate any Deadline Misses by never using the Cloud; however, given how the Predictive Scheduling Algorithm ow's design, it functioned optimally through utilizing both types of Scheduling Techniques prior to the onset of Actual Transmission.

The three scenarios are compared and show the differences between the Mean Latency for those levels; Deadline Misses for those levels and Throughput for those levels. These Data Comparisons are also listed in Table 1.

6.5 Discussion

Based on how we designed these three test scenarios, the data tells us that latency-adaptive Frontend Predictive Scheduling significantly improved real-time performance over both the Cloud Only Baseline and Reactive Scheduling Strategies.

Although Cloud Only Models were expected to fail, the results did allow us to quantify the rates of failure and provide an inequitable analysis of how to best optimize this very important Segment of Cloud Technologies as well establish why Edge Computing is the best option for Real Time Applications.

Additionally, the error message for Read Timed Out that occurred when using this Service reinforces not only the connectivity risks of relying exclusively on a server but also that it is also unreliable to provide real-time services on a Public Internet Connection.

The achievement of a 0% deadline miss ratio for both scenarios B and C supports what we consider the primary value of edge-cloud collaboration: having the ability to select a global optimal execution venue and switch between those venues seamlessly. In this trial, we consistently had poor network conditions and therefore the optimal execution venue was always the edge. The schedulers recognized this fact and configured their scheduling actions

accordingly. This is a clear indicator of the resiliency of the scheduler's performance under the worst-case scenarios of network conditions.

The most significant observation made during this trial involved the difference in performance level between the reactive and predictive schedulers. Both schedulers arrived at the same global scheduling decision (avoiding the cloud), however, the throughput performance of the predictive scheduler (higher throughput than the reactive scheduler) and latency performance (lower latency than the reactive scheduler) indicate that the predictive scheduler has much better operational efficiency than the reactive scheduler. Since the predictive scheduler identified that the high latency would persist, the system was able to reach a stable edge-only processing mode faster and manage its adaptive processes (for example, frame dropping) with more foresight as to future network constraints. The reactive scheduler, being reactive, was always one frame behind, responding to a measurement of latency that had already happened to a frame. When have many frames, this small lag time was likely the source of small but cumulative inefficiencies causing the lower overall throughput performance of the reactive scheduler when compared to the predictive scheduler.

This evaluation has a limitation in that there were no great variations in network conditions. Latency was high for all cases studied, meaning that there was no opportunity to observe how well the schedulers would implement dynamic switch back and forth between edge and cloud environments. In real-world use cases where network quality fluctuates (e.g., experience low latencies for periods), predictive scheduling could take advantage of lower latency windows by proactively offloading to cloud then quickly returning to edge shortly before congestion. This remains an area for further study and research.

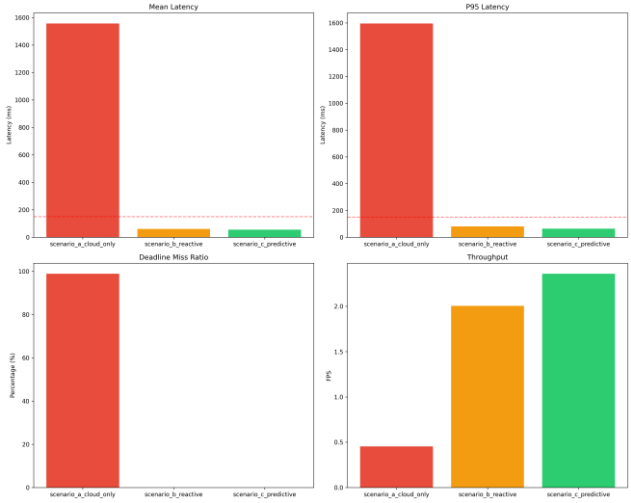

Figure 8: Comparative Analysis of Key Performance Metrics

Above is a visual representation of the key performance indicators compared across the three test cases. As can see in the upper left and upper right quadrants, the Mean and P95 Latency, respectively, show that Scenario A - Cloud Only has an extremely high latency compared to the more favorable results of Scenarios B (Reactive Scheduler) and C (Predictive Scheduler). The bottom left quadrant illustrates that Scenario A did not meet the 150 ms deadline on nearly all frames (approx. 99%), whereas both Scenarios B and C achieved 0% miss ratios. In addition, the bottom right quadrant compares each scenario by Throughput; in terms of Frames Per Second (FPS), the Predictive Scheduler has produced the highest average overall

results when compared to Scenario B (Reactive Scheduler) and also Scenario A (Cloud Only) significantly lagged behind. These charts clearly demonstrate that the Cloud Only (Scenario A) approach was a failure and that the results of the Predictive scheduling methodology proposed will yield more efficient results than currently achievable.

7 Conclusion and Future Work

Research has shown that using a predictive, latency-aware scheduling framework improves the performance of real-time video analytics in a collaborative edge-cloud environment. The design, development and rigorous evaluation of a predictive, latency-aware scheduling framework established that a proactive, forecast-based approach provides greater advantages than traditional cloud-only architectures and commonly deployed reactive scheduling strategies.

The primary finding of this study was the successful navigation of a high-latency network using the proposed predictive scheduler. The predictive scheduler combined the results of latency analysis obtained from the ARIMA model to accurately forecast the amount of latency in the network and was able to achieve a deadline miss ratio of 0% as opposed to the cloud-only baseline which had a deadline miss ratio of 98.89%. The predictive scheduler outperformed the reactive scheduler by providing a throughput that was 18% higher and a mean processing latency that was 9% less than the reactive scheduler. The predictive method of scheduling provides the potential to make more effective decisions regarding the processing pipeline by anticipating future network conditions. The results from this study have answered the primary research question by providing quantitative evidence for the superiority of the predictive model.

These findings have major implications for developing resilient IoT and real-time AI systems. As application workloads increase and networks become saturated with more devices, existing reactive systems that only react after performance degradation has occurred will become insufficient. In this project, it was demonstrated that the design of retroactive systems using proactive predictive intelligence is the best way to develop systems that are capable of adaptive behaviour by creating systems that are not only adaptive but are also robust and efficient.

Although the results of this study demonstrate many successful outcomes, there are some limitations associated with this study's experimental evaluation. First, the experimental evaluation was conducted in an environment that experienced continually poor network conditions. Although this environment effectively demonstrated the ability of the schedulers to avoid an unstable or failing network path, it did not test how quickly the schedulers would be able to transition to alternate paths in an environment with a high degree of latency variance. Lastly, while the ARIMA model used has been demonstrated to be a successful time series model, it is also a relatively simple model based on principal component analysis (PCA).

This research could be expanded into numerous additional areas of interest in the future. For example, future studies will benefit from a more thorough examination of how the system functions with various levels of strain on the network, as well as at varying levels of speed. This could be achieved through the use of emulation systems that provide a standardised way of measuring network congestion, packet loss, and other similar factors. These evaluations would give a better understanding of the predictive schedulers capability for efficient operation during this level of variability within a single network, or possibly between networks. Another area of further development would be to test the scheduling logic of predictive scheduling by implementing more advanced forecasting techniques, including

Long Short Term Memory networks. An LSTM network is capable of learning to identify non-linear patterns in network latency, which enables better predictions and ultimately improved scheduling logic. Finally, the scheduling logic could be developed to become content-aware, allowing the system to consider the complexity of the individual frames of video, as well as the importance of those frames, when scheduling events under heavy load.

References

Alsader, M., Barakabitze, A.A. and Mkwawa, I.H., 2025. QoE-Driven Adaptive Video Streaming: Architectures, Techniques, and Future Research Challenges Toward 6G Networks. IEEE Access.

Arthurs, P., Gillam, L., Krause, P., Wang, N., Halder, K. and Mouzakitis, A., 2021. A taxonomy and survey of edge cloud computing for intelligent transportation systems and connected vehicles. IEEE Transactions on Intelligent Transportation Systems, 23(7), pp.6206-6221.

Aslani, A. and Ghobaei-Arani, M., 2025. Machine learning inference serving models in serverless computing: a survey. Computing, 107(1), p.47.

Bai, T., Zhao, H., Huang, L., Wang, Z., Kim, D.I. and Nallanathan, A., 2025. A Decade of Video Analytics at Edge: Training, Deployment, Orchestration, and Platforms. IEEE Communications Surveys & Tutorials.

Khan, M.A., Baccour, E., Chkirbene, Z., Erbad, A., Hamila, R., Hamdi, M. and Gabbouj, M., 2022. A survey on mobile edge computing for video streaming: Opportunities and challenges. IEEE Access, 10, pp.120514-120550.

Karami, A. and Karami, M., 2025. Edge computing in big data: Challenges and benefits. International Journal of Data Science and Analytics, 20(7), pp.6183-6226.

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.

Ravindran, A.A., 2023. Internet-of-things edge computing systems for streaming video analytics: Trails behind and the paths ahead. IoT, 4(4), pp.486-513.

Ravindran, A.A., 2023. Edge Computing Systems for Streaming Video Analytics: Trail Behind and the Paths Ahead.

Shukla, S., Hassan, M.F., Tran, D.C., Akbar, R., Paputungan, I.V. and Khan, M.K., 2023. Improving latency in Internet-of-Things and cloud computing for real-time data transmission: a systematic literature review (SLR). Cluster Computing, 26(5), pp.2657-2680.

Swapna, B. and Divya, V., 2024. Latency aware intelligent task offloading scheme for Edge-Fog-Cloud computing–a review. Информатика и автоматизация, 23(1), pp.284-318.

Trigka, M. and Dritsas, E., 2025. Edge and cloud computing in smart cities. Future Internet, 17(3), p.118.

Wang, W., Wei, X., Tao, W., Zhou, M. and Ji, C., 2025. Quality of experience-oriented cloud-edge dynamic adaptive streaming: Recent advances, challenges, and opportunities. Symmetry, 17(2), p.194.

Wang, Z., Liu, J. and Zhu, W., 2023. Edge intelligence-empowered immersive media. IEEE MultiMedia, 30(2), pp.8-17.

Xu, R., Razavi, S. and Zheng, R., 2023. Edge video analytics: A survey on applications, systems and enabling techniques. IEEE Communications Surveys & Tutorials, 25(4), pp.2951-2982.

Zhan, S., Huang, L., Luo, G., Zheng, S., Gao, Z. and Chao, H.C., 2025. A Review on Federated Learning Architectures for Privacy-Preserving AI: Lightweight and Secure Cloud–Edge–End Collaboration. Electronics, 14(13), p.2512.