2024 IEEE International Conference on Cloud Engineering (IC2E) | 979-8-3315-2869-0/24/$31.00 ©2024 IEEE | DOI: 10.1109/IC2E61754.2024.00031

Scalable and Cost-Effective Edge-Cloud Deployment for Multi-Object Tracking System

Ratul Kishore Saha
TCS Research
Mumbai, India
ratulkishore.saha@tcs.com

Dheeraj Chahal
TCS Research
Mumbai, India
d.chahal@tcs.com

Rekha Singhal
TCS Research
New York, USA
rekha.singhal@tcs.com

Manoj Nambiar
TCS Research
Mumbai, India
m.nambiar@tcs.com

***Abstract*—The proliferation of edge devices has significantly advanced technologies in sectors such as autonomous driving and surveillance. However, deploying machine learning models on these resource-constrained devices presents challenges, including scalability and managing unpredictable workloads, which often hinder real-time performance (both latency and throughput) in edge-only environments. To address these issues, a potential approach is to deploy on an edge-cloud ecosystem, however, it may hinder latency due to communication delay between edge and cloud. We leverage the cloud service Message Queuing Telemetry Transport (MQTT) protocol powered by 5G internet, for communication to manage the latency. We propose a scalable edge-cloud ecosystem specifically designed for a Multi-Camera Sensor-based Multi-Object Tracking (MCS-MOT) pipeline within industrial deployment contexts, ensuring compliance with Service Level Agreements (SLAs). Our approach introduces a novel camera feed content-guided load balancing technique that dynamically manages workloads between edge and cloud. We extract features from incoming camera feed content to inform the load-balancing process efficiently. The load balancer determines the maximum number of concurrent feeds that can be processed at the edge, with the remaining feeds handled by the cloud based on the content of the camera feed. Additionally, we propose a cost model to estimate the expenses of deploying the edge-cloud ecosystem in real-world scenarios with dynamic workloads. Our experimental results show that the proposed SLA-compliant MCS-MOT system significantly outperforms edge-only architectures in terms of latency and throughput for the YOLO-DeepSORT algorithm. We also illustrate the estimated and actual deployment costs of the system, highlighting its cost-effective scalability and optimized real-time performance.**

***Index Terms*—MCS-MOT, Content guided load balancing, Edge-Cloud, MQTT, YOLO-DeepSORT**

I. Introduction

The growing demands of Multi-Object Tracking (MOT) in smart city applications such as video surveillance and traffic management necessitate robust technological solutions. These smart city systems utilize advanced surveillance technology to improve public safety and enhance the quality of life for residents by enabling functionalities like object tracking and suspicious activity detection through video-based systems. In various edge applications like surveillance, robotics, and autonomous vehicles, latency performance is critical. Typically, lightweight deep learning models perform adequately on edge devices for single inference tasks. However, these models struggle under the load of concurrent requests, often leading to Service Level Agreement violations due to the resource-constrained nature of edge devices and hence scalability remains a significant challenge in edge-only deployment settings. Also, the deployment of Deep Learning models at the edge entails high computational power and energy consumption, resulting in diminished system reliability and increased maintenance challenges. Traditional deployment on high-performance hardware is not viable for real-time processing due to high costs and inefficiencies at the edge.

On the other end of the spectrum, cloud service providers such as AWS [1], Azure [2], and GCP [3] offer flexible and scalable infrastructure capable of handling multiple concurrent requests. This shift not only enhances scalability but also ensures more reliable performance across various applications, making it a strategic move towards more robust and efficient edge computing frameworks. Moreover, the cost integration of the cloud services is flexible in nature while maintaining the workload to be balanced. This strategic integration of edge and cloud ensures that the systems are capable of adapting to varying workload demands effectively, thereby supporting more complex and demanding applications in smart city environments and beyond. This seamless fusion of edge and cloud computing frameworks marks a significant advancement in deploying real-time, efficient, and scalable multi-object tracking systems across diverse operational scenarios.

This paper explores the development of an SLA-compliant, multi-camera sensor-based MOT system, increasingly vital in the era of edge-cloud ecosystems. We propose a sophisticated edge-cloud architecture that combines the immediate processing capabilities of edge devices with the robust computational power of cloud services. This merger is crucial for managing the intensive data processing demands of real-time MOT systems without the limitations of edge-only deployments. Our integrated system ensures that deployments meet SLA requirements while adapting to dynamic processing demands, making it ideal for real-time deployment.

Succinctly, our contributions are as follows:

- An edge-cloud deployment ecosystem enhanced with capabilities orchestrating for Multi-Camera sensor-based multi-object tracking applications powered with 5G.
- We propose an efficient camera feed content-aware and lightweight SLA-based load balancing strategy designed to optimize workload distribution between edge and cloud

2694-0825/24/$31.00 ©2024 IEEE
DOI 10.1109/IC2E61754.2024.00031

environments, tailored to the dynamics of incoming camera feeds and requests.
- Furthermore, we introduce a complete scalable edge-cloud architecture alongside a mathematical cost estimation model to facilitate realistic deployments within industrial settings.

These major contributions are aimed at meeting the essential requirement for an edge-cloud combination in MOT tasks, focusing on scalability and reliability for real-time deployment with minimal cloud cost using our estimated model. The organization of the paper is structured to facilitate a comprehensive understanding of the subject and our work. Section II provides an overview of related work in the MOT domain, highlighting existing architectural approaches. Section III delineates the methodology and specifics of our proposed architecture. Feature extraction and proposed load balancing techniques used in our implementation are discussed in Section III-A and III-C followed by our cost model in Section IV. The experimental setup is discussed in Section V. Section VI presents the experimental analysis, showcasing the practical application and effectiveness of our approach. Finally, Section VII concludes the paper, summarizing key findings and proposing directions for future research in this area.

II. Related Work

The prior research in MOT primarily involves two components: object detection, which accurately localizes objects, and tracking algorithms, which predict object trajectories in subsequent frames. This approach is referred to as the "Tracking by Detection" (TBD) approach [4]. Significant progress in object detection has been made with deep neural networks like YOLO [5], RCNNs [6], and SSD [7]. Fast and Faster RCNNs are two-stage, computation-intensive detectors, while YOLO and SSD are more efficient one-shot detectors. However, their high computational demand challenges deployment on resource-limited devices. To mitigate this, lightweight models like Tiny-YOLO [8] and Slim-YOLO [9] have been developed for edge device deployment. The major drawbacks of lightweight models are their poor tracking accuracy and inability to effectively track small objects.

In terms of tracking objects across video frames, researchers have developed various methods. One notable approach is Simple Online Real-Time Tracking (SORT) [10], which combines three algorithms: the Kalman Filter [11], a matching cascade, and the Hungarian algorithm [12]. Building upon SORT, an improved version was introduced by the same authors in [12], where they introduced a deep association network to enhance object re-identification and achieve better tracking performance under challenging conditions, such as changes in illumination and occlusions. Another approach, known as Strong SORT, was proposed in [13]. Huang et al. [14] introduced a hierarchical deep high-resolution network (HDHNet) for an end-to-end online MOT system. Stadler et al. [15] proposed a PAS tracker that employs a novel similarity measure and Cascade RCNN to effectively utilize object representations. Yang et al. [14] designed a dense-optical-flow-trajectory voting method to measure object similarity across adjacent frames and integrated YOLOv3 for MOT. Jin et al. [16] proposed online MOT with a Siamese network and optical flow (Siamese-OF), while Dike et al. [17] introduced a quadruplet network to track objects in crowded environments. Yu et al. [18] proposed a self-balance method that integrates appearance similarity and MOT ion consistency. Youssef et al. [19] achieved MOT through cascade region-based convolutional neural networks and feature pyramid networks. Additionally, several TBD algorithms are presented in [12], [20].

Various approaches have been undertaken to implement vision systems within the edge-cloud paradigm. While some strategies focus solely on edge or cloud solutions, others explore a combination of both. In [21], authors present a novel approach to efficient and accurate visual MOT in smart cities using lightweight computing and edge devices. Noguchi et al. [22] introduce an open-based architecture for MOT, utilizing shared devices and distributed search to optimize efficiency and reduce network load. Xu et al. [23] investigated real-time human MOT on network edge devices, employing techniques like HOG, SVM, and KCF, and achieved notable results using only a Raspberry Pi 3 as the edge device. Additionally, Gu et al. [24] developed a collaborative edge-cloud framework aimed at enhancing the efficiency and accuracy of IoT-based MOT, optimizing both performance and energy consumption. Several related architectures have been presented in [25], [21], [26]. Besides the deployment at edge-cloud Phalak et al. [27] proposed cloud and fog computing to efficiently manage ML/DL inference, reducing costs and SLO violations in IoT based used cased applications. Manju et al. [28] proposed an empirical study demonstrating that deploying AI inference workflows using FaaS and cloud storage services across multiple CSPs can reduce costs by 83% with optimal resource mapping, alongside presenting analytical models for cost estimation. Chahal et al. [29] introduced a workload characterization approach for scheduling bursty workloads on a highly scalable serverless architecture, using AWS SageMaker and Lambda to effectively balance inference workloads and prevent SLA violations, demonstrated through a deep learning-based recommender system. In [30], authors optimized ML inference across heterogeneous infrastructures, reducing costs and improving efficiency in diverse computing environments. Gunasekaran et al. [31] introduced Spock which integrates VMs and serverless functions to enhance elasticity, reduce costs, and minimize SLA violations However, all the existing research does not address scalable SLA-based architectures within edge-cloud ecosystems and cost estimation models for edge-cloud ecosystems in MOT application based on the incoming content of camera feeds.

III. Proposed Methodology

Traditional MOT methods, especially those using resource-heavy deep learning models like YOLO-DeepSORT, often suffer from slow performance on standard computers due to

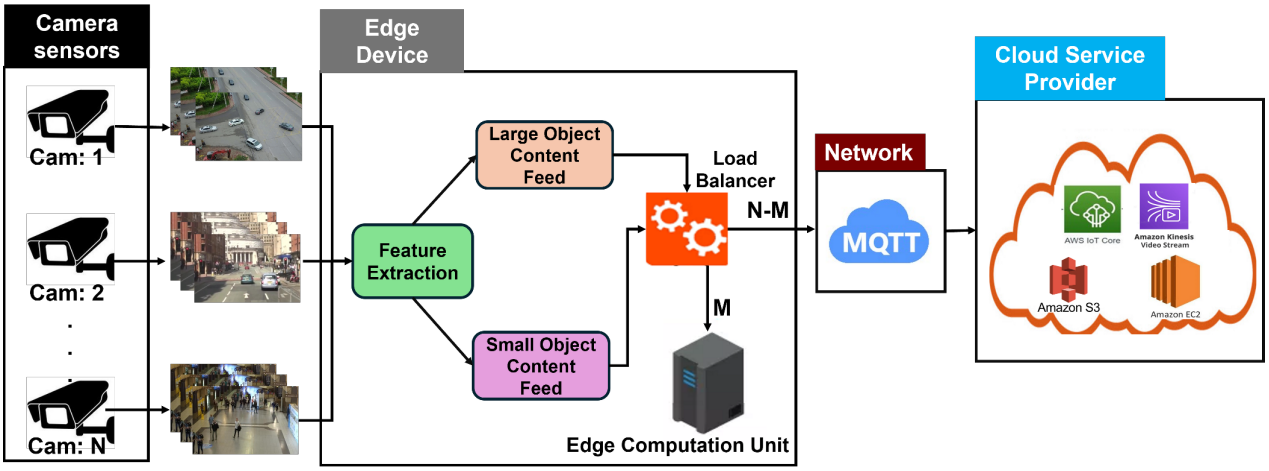

Fig. 1. Propsed Edge-Cloud framework for MCS-MOT system

sequential frame processing. This issue is even more pronounced on edge devices like Raspberry Pi and NVIDIA Jetson Nano, which lack the computational power to handle the high demands of these algorithms, particularly with high frame rate videos (e.g., 30 FPS). Although lightweight YOLO models such as YOLOv8n have been developed for edge devices, they struggle with dynamically analyzing videos containing small objects. Additionally, larger models require high-end computational power and accelerators, which are impractical for resource-limited edge devices. Furthermore, scalability on edge devices is not cost-effective and unsuitable for dynamic loads in real-world scenarios. Building on our previous work [32], where we enhanced the MOT pipeline on general-purpose machines, we recognize the need for effective deployment of MOT models on edge devices remains an open area of research.

To address these issues, this paper presents an advanced SLA-aware edge-cloud architecture designed to support real-time MCS-MOT applications based on incoming camera feed content for real-time use cases. Our proposed framework illustrated in Figure 1 enhances the processing of diverse video feeds captured by surveillance cameras across smart cities, which vary significantly in scene dynamics and objects.

The content of live camera feeds can vary significantly due to factors like camera angle and object presence, etc. Therefore, for dynamic feeds, it is essential to execute the MCS-MOT methodology with prior knowledge of the feed content. Our approach begins with feature extraction to identify objects, followed by classifying camera feeds into two categories based on object size: small objects and large objects. This classification helps determine the optimal processing location—either at the edge or in the cloud. For example, running a content-agnostic MCS-MOT algorithm with no objects present wastes edge and cloud resources. Feeds with large or simple objects are processed directly at the edge using lightweight models like YOLOv8n-DeepSORT to ensure rapid execution and minimal latency. Conversely, more complex feeds, especially those with small or intricate objects, are routed to the cloud where more robust models like YOLOv8x-DeepSORT are employed to ensure high accuracy in object detection and tracking within SLA.

To maintain SLA objectives and optimize resource utilization, we introduce an SLA-based load balancer that dynamically allocates video feeds between the edge and the cloud based on incoming camera feed content to the edge device. This ensures that the proposed edge-cloud ecosystem runs simultaneously to maintain a suitable balance between edge and cloud resources.

This integrated edge-cloud approach not only enhances the flexibility and efficiency of deploying real-time surveillance systems but also ensures that the network adheres to predefined SLA parameters, thereby optimizing the overall performance and cost-effectiveness of the MCS-MOT pipeline.

In our edge-cloud system, the cloud counterpart is constructed using the Amazon Web Services (AWS) framework,

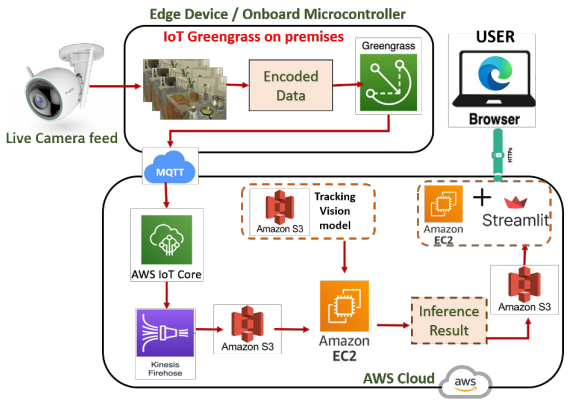

Fig. 2. Detailed Internal Edge-Cloud Architecture for MOT pipeline

facilitating robust and efficient data handling. AWS Kinesis Firehose [33] ensures real-time data streaming without bottlenecks, allowing for the simultaneous handling of large data streams. AWS Greengrass [34] extends cloud capabilities through edge devices, enabling local function execution and secure cloud interaction, even in offline scenarios. For data storage and accessibility, Amazon S3 [35] offers scalable and secure solutions, while scalable AWS EC2 [36] instances efficiently execute the MCS-MOT algorithm under varying loads in alignment with SLA objectives. Additionally, we introduce a cost estimation model designed to optimize financial efficiency when deploying our architecture in the cloud. This model focuses on scalable costs in real-time deployment scenarios.

This architecture optimizes resource allocation through effective load balancing and enhances scalability and flexibility, ensuring that the system meets stringent SLA requirements. This sophisticated use of edge and cloud resources offers a robust and scalable solution to the challenges of modern surveillance systems. Now, we present different blocks of the proposed methodology.

A. Feature Extraction

As mentioned above, for feature methodology, we employ the lightweight Canny edge detection [37] technique to analyze and classify image content based on structural characteristics. This method is adept at identifying sharp discontinuities within an image, which effectively delineates structural boundaries. The process begins by converting the image of an input camera feed to grayscale, which simplifies the complexity of the data and enhances the effectiveness of edge detection. Next, we apply the canny edge detection algorithm, for detecting a broad range of edges, to the grayscale images.

Once the edges are highlighted, we quantify the feature extraction by counting the number of edge pixels present in each image within the Region of Interest(RoI). These RoIs are figured out based on the detected edges of the observed frames of the images. Note that, in scenarios where no objects are present, no features are detected in the RoI. This metric is critical as it forms the basis for our classification system. We set a predefined threshold for the number of edge pixels, which allows us to determine whether an image predominantly contains larger or smaller objects. Images with a count of edge pixels below the threshold are classified as containing smaller objects, whereas those with counts above the threshold indicate the presence of larger objects, illustrated in Figure 3.

B. Proposed SLA-based Latency Model

In our architecture deployed on the edge-cloud platform, we recognize three critical factors dictating end-to-end latency: edge latency (L_{edge}), communication latency between edge and cloud (L_{comm}), and computation latency within the cloud (L_{cloud}). Equation 1 succinctly encapsulates the total latency for our MOT pipeline.

$$L_t = L_{edge} + L_{comm} + L_{cloud} \tag{1}$$

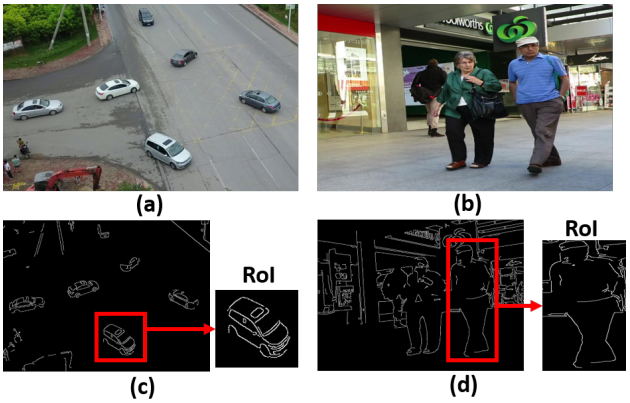

Fig. 3. (a) and (b) show image frames containing small and large objects of two incoming camera feeds, respectively, while (c) and (d) show the features extracted from these images using our proposed method.

While Equation 1 highlights the significance of these factors, it's crucial to note their impact when the entire algorithm operates solely on the edge. When the data is being processed at the edge device only the other latency (L_{comm}, L_{cloud}) will be zero. The whole latency equation will operate when the edge cloud running as a co-processing scenario.

C. Camera feed aware SLA based Load Balancer

To ensure scalability and compliance with our SLA, we have developed a dynamic SLA-based load-balancing technique that intelligently allocates processing tasks between edge and cloud environments based on the camera feed content. This approach adapts to fluctuating workloads and varying resource availability.

Consider a scenario where multiple camera feeds are streamed to an edge device. Not all feeds contain objects that need to be tracked by the MOT algorithm, so the computational load highly depends on the content of the camera feed. If a feed does not contain any objects, the load is considered zero. This dynamic load concept motivates the design of an intelligent load-balancing system that adjusts based on the content of the camera feeds. Therefore, after the feature extraction, our load balancer assesses each feed's processing needs and allocates resources appropriately. When the edge device can efficiently process a subset of these feeds containing large object features within the latency constraints set by the SLA, it does so locally. Excess feeds, which exceed the edge device's capacity, are smartly offloaded to the cloud where more extensive computational resources are available. Figure 4 shows the minimum latency supported at a given concurrency for our proposed methodology on an edge device illustrated in Section VI-A To manage this balancing act, we have developed a regression model to predict the number of concurrent operations that can be handled by the edge device without breaching SLA thresholds. We describe the relationship between concurrency and latency at the edge using a three-degree polynomial regression model:

$$Y_c = \delta \times L_{edge}^3 + \alpha \times L_{edge}^2 + \beta \times L_{edge} + \gamma \quad (2)$$

Where Y_c represents the maximum concurrency supported on at the edge device with a latency less than L_{edge}. The optimum value of the unknowns(δ, α, β and γ) can be found by solving the regression least square method based on the data point shown in section VI-A. The optimum value of the known parameters is shown in Table I

Device	γ	δ	α	β	RMSE
Edge	-2.093	11.333	-28.548	31.591	0.4

TABLE I
ESTIMATED PARAMETERS OF LOAD-BALANCER

From Table I, the load balancer's ability to forecast the maximum number of concurrent video feeds (Y_{max}), particularly those with large aspect ratios that the edge device can process within SLA latency (L_{SLA}), becomes evident. This predictive capability is crucial for effectively managing the processing workload between the edge and the cloud. When the edge device reaches its maximum concurrency limit Y_{max} that can be executed within SLA latency L_{SLA} as forecasted by the load balancer, additional feeds are seamlessly offloaded to the cloud. Note that these parameters may vary with the hardware configuration of the edge device. Here, a robust infrastructure running both nano and large MOT models concurrently takes over, ensuring that the SLA's stringent performance objectives are continuously met.

For video feeds characterized by smaller object aspect ratios, which typically require more detailed analysis, the system bypasses local processing at the edge device entirely to avoid SLA violations due to high processing time. Note that this aspect ratio is a quantity defined as the ratio of width to height of the objects in a video frame. Instead, these feeds are directly sent to the cloud, where advanced processing capabilities can better handle their complexity, thus enhancing the accuracy and the execution speed of the multi-object tracking.

This strategic distribution of tasks not only optimizes the utilization of computing resources across the edge and cloud but also minimizes response times and the cost of the deployment. Because we intelligently deploy only selected feeds to the cloud for processing, not all feeds are sent directly to the cloud. As a result, the system adheres more strictly to SLA requirements, providing a reliable and efficient multi-object tracking service. By intelligently balancing processing loads based on the content characteristics and computational intensity of each feed, this approach significantly enhances both the scalability and performance of our MCS-MOT pipeline, making it highly effective for deployment in diverse real-world scenarios. The detailed algorithm is illustrated in 1.

Algorithm 1 Camera feed aware SLA-based load balancing for MCS-MOT pipeline

Input: Total number of camera feeds N, SLA latency L_{SLA}, Load balancer (LB) parameters
Output: Processed camera feeds using MOT algorithm within SLA
$i \leftarrow 0$
Perform feature extraction on all N camera feeds
Segregate camera feeds into large and small object content feeds
$large\_feeds \leftarrow$ feeds with large objects
$small\_feeds \leftarrow$ feeds with small objects
$M \leftarrow$ LB decides the maximum concurrency of feeds containing large objects at the edge within L_{SLA}
while $i < N$ **do**
  $SLA\_met \leftarrow$ LB checks if processing M camera feeds at edge meets SLA
  if SLA_met **then**
    for each feed f in $large\_feeds$ up to M feeds **do**
      Process camera feed f locally at the edge using lightweight MOT model
      $i \leftarrow i + 1$
    end for
    for each feed f in $small\_feeds$ or remaining $large\_feeds$ if $i < N$ **do**
      Transfer camera feed f to the cloud via MQTT via 5G internet
      Process in the cloud using robust MOT algorithm to handle complexity of the and maintain L_{SLA}
      $i \leftarrow i + 1$
    end for
  end if
end while

IV. PROPOSED COST ESTIMATION MODEL

The migration of edge workload to the cloud for processing incurs an additional cost. A robust cost model can be useful in estimating the cost of cloud deployment using various services. Furthermore, cost models provide a systematic and quantitative method for comparing various architectural design schemes. A cost model can be used to find a trade-off between performance and cost by selecting the appropriate network bandwidth, storage service configuration, EC2 machine types, service configuration, etc.

In this segment, we outline the expense framework for implementing an edge-cloud architecture in a real-time setup employing AWS cloud platforms. This involves the utilization of two cloud services and a comparative examination based on the deployment structure. For AWS deployment, we employ Kinesis Video Stream, Simple Storage Service (S3), EC2 instances, etc. In the proposed edge-cloud model the total cost can be defined as:

$$C_{total} = C_{edge} + \sum_{i=1}^{S} C_i \quad (3)$$

Where C_{edge} represents the fixed cost of the edge device, C_i signifies each cloud service cost of the deployment architecture, and S indicates the number of services utilized for deployment on a specific cloud provider. Our primary

focus will be on estimating costs related to cloud deployment scenarios as it vary with the incoming loads. However, the cost of each cloud service C_i is contingent upon various sub-factors such as functions, storage, region, network, etc. Note that, in this study, we aim to focus on the dynamic cloud cost only. The cost due to various cloud services used in the deployment architecture is calculated as follows:

A. AWS IoT Core

The cost of utilizing a core on AWS is determined by various factors. Firstly, there is a charge for messaging, which depends on the number of messages transmitted from edge devices to the AWS Core. Each message incurs a cost, denoted by R_{message}. Additionally, there are costs associated with the utilization of rules and actions within the Core environment. Rules are used to process incoming messages and trigger actions accordingly, with each rule evaluation incurring a cost represented by R_{rule}. Similarly, actions triggered by rules, such as storing data or invoking functions, have associated costs denoted by R_{action}. Furthermore, there are connection costs involved in maintaining the connection between edge devices and the Core service, represented by $R_{\text{connection}}$. The following is the cost of AWS core:

$$\begin{aligned} C_{\text{IoTcore}} &= m \times K \times L \times R_{\text{message}} + m \times K \times L \times R_{\text{rule}} \\ &\quad + m \times K \times L \times R_{\text{action}} + T \times R_{\text{connection}} \end{aligned} \tag{4}$$

Where, m, K, L are the number of the edge devices, total messages, and average billable messages. Usually, L can be computed as $D_a/billed\_meter$, D_a is the average data size of each message. The billed_meter is the minimal size of the data taken as a message, it depends on the cloud service provider. Additionally, the duration of device operation (T) plays a role in determining the total cost.

B. Kinesis FireHose

For Kinesis Firehose service, the cost can be expressed as:

$$C_{kinesis} = m \times K \times D_a \times R_{kinesis} \tag{5}$$

Here, $R_{kinesis}$ denotes the cost of the Kinesis Firehose service, typically charged on a per gigabyte (GB) basis.

C. AWS Storage S3

Upon the arrival of data at the AWS live video storage service, it is then stored across the storage services of various cloud providers. The cost of storage, denoted as C_{st}, primarily depends on the cloud vendor and their specific storage operations, including data reading and writing activities. The storage cost can be effectively estimated using the formula:

$$\begin{aligned} C_{S3} &= D_{storage} \times R_{storage} + (n_p + n_l + n_c) \times R_{request} \\ &\quad + D_{out} \times R_{out} \end{aligned} \tag{6}$$

Here, $D_{storage}$ represents the total amount of data stored in AWS S3, expressed in GB and charged $R_{storage}$ per GB basis while n_p, n_l, and n_c are the number of PUT, LIST, and COPY requests respectively. The variable $R_{request}$ denotes the cost incurred per storage operation request. Additionally, D_{out} signifies the volume of data transferred out from one region to another, and R_{out} is the associated cost per GB for outbound data transfer. Notably, the cost for data transfer within the same region (intra-location) is zero, thereby avoiding additional charges for local data movements.

D. AWS EC2 Computation Cost

Apart from the aforementioned costs involved in deploying applications, there is a cost associated with computing the algorithm. Typically, this is billed on an hourly basis to execute the computation algorithm. The cost for EC2 instances in AWS can be expressed as:

$$C_{compute\_i} = T \times R_{instance\_i} \tag{7}$$

Here, T represents the total time consumed to execute the function/application in that instance as a unit of hour, and $R_{instance\_i}$ is the cost of the i^{th} instance on an hourly basis. During the implementation of the workflow, I computational instances have been utilized. Therefore, the total computational cost incurred can be calculated as:

$$C_{compute\_t} = \sum_{i=1}^{I} C_{compute\_i} \tag{8}$$

Here, $C_{compute\_t}$ represents the total computational cost, and $C_{compute\_i}$ represents the computational cost of the i^{th} instance.

V. Experimental Setup

In this section, we discuss the infrastructure and cloud services we used in the experimental analysis of the proposed framework.

A. Hardware Setup at Edge

In realizing this architecture, we employed a standard CPU-based laptop as the edge device. This laptop is equipped with four physical Intel i5 processors, each running at a clock frequency of 2.70GHz. The available memory of the edge device was 8 GB. To design a prototype of the architecture within an in-house lab environment, multiple camera sensors were utilized. This setup provides adequate computational power to capture a live feed in real-time and efficiently transfer it to the cloud. During the experiments, no additional load was deployed on the edge device. Additionally, the edge system was equipped with a Windows platform, and Python-based codes were deployed there. Note that, at the edge, the lightweight Yolov8n-DeepSORT algorithm is deployed, with a size of approximately 6 megabytes (MB).

B. Network Setup

For the real-time communication between the edge device and the cloud, we utilized the technology of highspeed 5G internet. Specifically, the transfer of captured image data to the cloud was facilitated using the MQTT protocol, operating over a high-speed 5G network. The connection between the edge device and the cloud was secured with protected certification, ensuring safety and confidentiality as provided by the cloud service provider. Once this secure communication link was established, the edge device could seamlessly communicate and transfer data to the cloud without interruption.

C. Cloud Setup in AWS

For the execution of the MCS-MOT algorithm, we leveraged the AWS cloud infrastructure. The edge device was integrated with the cloud using AWS Greengrass services, ensuring a high-speed and secure connection. The live video frames captured by the edge device were first encoded using the base64 technique and then transmitted to AWS IoT Core services. The base64 encoding method efficiently converts frame integer arrays into a compact string, significantly reducing the data size (to just a few KBs) and enabling rapid transfer with minimal latency, even over low-bandwidth network connections via MQTT. This base64 encoding seamlessly binds with to make .JSON format which is to be published to the AWS IoT core. This encoding is efficient in reducing the heavy data transfer from edge to cloud.

Upon arrival in the AWS cloud, the data was streamed to an Amazon S3 bucket using the AWS Kinesis Firehose service. Here, Kinesis Firehose is used as a streamlined flow of data from core to S3. The Yolo-DeepSORT algorithm was deployed on an AWS Elastic Compute (EC2) instance, specifically a g4dn.xlarge instance for our experimental purposes. The instance is T4 GPU enabled along with 4 CPU cores. Note that, the cloud has both large and small models of YOLO-DeepSORT to be applied based on the camera feed feature. Note that, the Size of the YOlOv8x large model is 131 MB. This setup allowed for the real-time decoding of image data from encoded strings and the tracking of objects within these frames. For the tracking of the objects, the YOLO-DeepSORT algorithm is running continuously on the EC2 instance. The results of this tracking process were then stored back in the S3 bucket. For scenarios involving multiple cameras, we will establish to spawn multiple EC2 instances based on the SLA objective to be maintained while running. Note that we use an elastic compute strategy for horizontal scaling of EC2 instances to maintain the SLA.

Additionally, to facilitate live visualization of the tracking results, we developed a user interface tool using the Python Streamlit framework. This tool was hosted on a free-tier EC2 instance (t2.micro), functioning as a web server to present the tracking outcomes on a proper HTTP website. The biggest advantage of AWS EC2 instances is that they allow hosting a web user interface globally using the machine's outbound port. This setup enables users to access and visualize results from any location worldwide.

VI. Experimental Analysis

In this section, we discuss experiments to evaluate the performance of edge-only and single-sensor-based architecture. Also, we evaluate the accuracy of the proposed cost model followed by scalability and performance analysis of the edge-cloud framework.

A. Performance Analysis of Edge Only Architecture

In our experiment, we deployed the MCS-MOT algorithm using the YOLOv8n-DeepSORT algorithm on an edge device, processing multiple concurrent camera feeds. The results of this edge-only execution study are depicted in Figure 4, where the red line indicates an SLA threshold of 0.75.

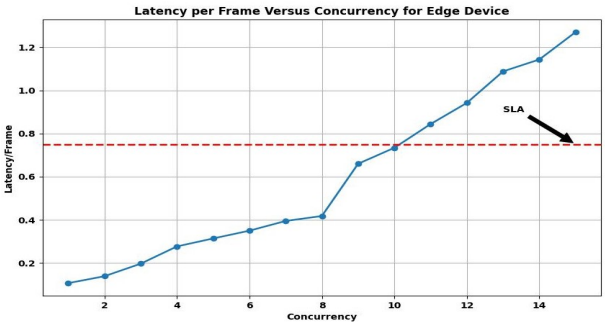

Fig. 4. Latency/frame versus concurrency for edge-device.

The Figure illustrates that 4 the edge device maximum cater 10 concurrency of 15 camera feeds within its resource constraints limits. For more than that the edge device fails to handle the concurrent request to maintain the SLA. The SLA is set to 0.75 here. However, one can choose the required SLA based on the need to run the MCS-MOT system. However, the data clearly shows that the system maintains compliance with the SLA up to a concurrency level of 10. Beyond this point, the SLA is violated, indicating a need to offload additional camera feed to the cloud for processing to maintain the SLA. Hence, this proves the need for an edge-cloud ecosystem to be useful in maintaining SLA. However, the large MOT model YOLOv8xDeepSORT model is not fitted to this edge device, so it was discarded for this edge concurrent study.

B. Scalability and Performance Analysis of Proposed Edge-Cloud Ecosystem

To demonstrate the efficiency of the proposed SLA-aware MCS-MOT methodology, we initiate our experiment with 15 concurrent camera feeds directed to the edge device, comprising a mix of small and large object content. The SLA is set at 0.75 seconds per frame. Initially, the edge device classifies the video feeds using our feature extraction method to determine whether they contain large or small objects.

To balance the concurrent load between the edge and cloud, we use the SLA objective in equation 2 with the optimal values from Table I. We observe that the edge device could support a maximum concurrency of approximately 10 feeds for large object content. The remaining 5 camera feeds are transferred to

the cloud, where the proposed architecture processes them on EC2 instances using the larger YOLOv8x-DeepSORT model. This setup demonstrates the maximum efficiency of the algorithm in the cloud environment. Note that the YOLOv8n-DeepSORT model can also be used for large object content if the edge cannot handle the concurrency within the SLA, and the larger model can be used for small object content due to the cloud's flexibility and larger deployment options to maintain the SLA.

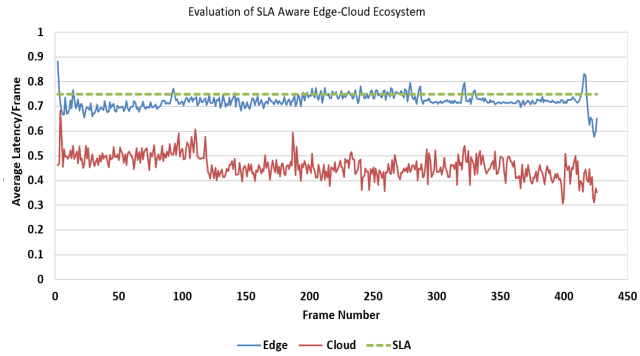

Fig. 5. SLA aware execution of Edge-Cloud MCS-MOT application for 15 concurrent workloads

Figure 5 illustrates the SLA-aware execution results of the edge-cloud ecosystem. We calculate the average latency per frame for both the edge and the cloud. For the edge, the average timing per frame is calculated for 10 feeds, and for the cloud, it is calculated for 5 feeds. The results show that both the edge and cloud systems respond within the SLA for each frame, based on the respective algorithms determined by feature extraction. The cloud average latency per frame includes the communication time to send data from the edge to the cloud and an additional 0.1 seconds to account for any uncertainties. Despite this, the latency per frame remains within the specified SLA. Moreover, Figure 6 demonstrates how the load balancer manages the dynamic camera feed workload between the edge and cloud while maintaining the SLA of 0.75 seconds per frame. It shows that the edge device can handle up to 10 concurrent feeds. When the number of feeds exceeds 10, the additional feeds are processed in the cloud. However, as the workload decreases, the system dynamically adjusts and reduces the reliance on cloud processing. This demonstrates the robustness and efficiency of our method in handling mixed video feeds while adhering to stringent SLA requirements, making it suitable for industrial deployments.

C. Cost Analysis of proposed single camera sensor-based architecture set up in AWS

In this study, we conduct a detailed cost analysis for our proposed edge-cloud architecture for a single camera sensor feed (here m=1) in the AWS ap-south-1 (Mumbai) region. The execution cost is comprehensively determined based on various AWS services and their associated usage for an hour

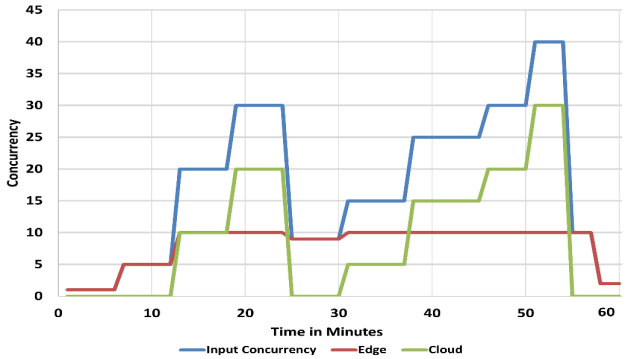

Fig. 6. Load Balancing between edge and cloud-based on incoming dynamic camera feed load.

of the execution. The calculation begins by examining the data transfer from the edge device to AWS Core (Table II).

For AWS Core, we conduct a detailed cost calculation to estimate the expenses associated with device connectivity, message handling, and rules engine operations through the proposed equation as follows.

(Device connective cost) We consider one device connected continuously throughout the month, accruing a total of 43, 800 minutes. The cost for connectivity is significantly low at 0.092 USD per million (M) minute ($R_{connection}$) and hence this cost is ignored.

(Messaging cost) Since AWS IoT core bills messages in 5 KB increments, each 86 KB message is treated as average (86/5) = $\sim$ 18 billable messages. The cost of AWS IoT core for publishing messages is 1.05 USD per million(M) messages ($R_{message}$).

(Rule engine cost) The rules engine which is integral to processing and responding to incoming data, is triggered 23598 times in one hour. Each rule triggers one action leading to an equivalent number of billable actions. Each rule and action is charged at the rate of 0.158 USD per million rule/action triggered (R_{rule}/R_{action}).

Therefore, the estimated cost for AWS IoT core based on the equation 4is as follows:

$$\begin{aligned} C_{\text{IoTcore}} &= 1 \times 23598 \times 18 \times 1.05 \times 10^{-6} \\ &\quad + 1 \times 23598 \times 18 \times 0.158 \times 10^{-6} \\ &\quad + 1 \times 23598 \times 18 \times 0.158 \times 10^{-6} \\ &= 0.59 \end{aligned}$$

Also, from the actual experiment, we observed the same cost for the AWS IoT core service.

Video frame Size (KB)	Time to upload (s)	Data transfer per hour (GB)	No of image frame per hour
86	0.15	1.92	23598

TABLE II
DATA TRANSFER DETAILS FROM EDGE TO AWS CORE

To estimate the cost of data streaming via AWS Firehose, charges are applied based on the volume of data injected. The Kinesis Firehose charges .034 USD per GB ($R_{kinesis}$) data injection. As detailed in our proposed formula (referenced in 5), the estimated cost was calculated as follows:

$$C_{kinesis} = (1 \times 23598 \times 86/1024/1024) \times 0.034 = 0.0658$$

Note, this division by 1024 is done to make the data in GB. During practical operations, the actual cost incurred was slightly higher at 0.0678 USD per hour. This variance between the estimated and actual costs is minimal, demonstrating the reliability of our cost estimation model.

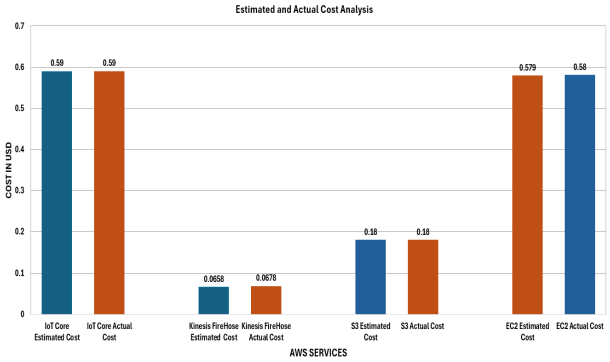

Fig. 7. Estimated and actual cost analysis of Single Camera Feed MCS-MOT algorithm

In our evaluation of AWS S3 Standard storage for an optimal balance of accessibility and cost, we initially stored 1.92 GB of data, with the storage cost calculated at $0.023 per GB. Alongside storage, we utilized an EC2 instance to monitor and manage data interactions. This setup involved periodic checks to determine if new data had arrived. Upon detection of new data, read operations were executed to fetch data from S3 and download it to the local storage of the EC2 instance for processing. Post-processing, the results were written back to S3, necessitating write operations. During one hour of system operation, we recorded a total of 858 list operations, 23,598 disk writes, and 1,902 read operations. The AWS S3 charges 0.000005 USD per request ($R_{request}$). Here, the data is moved within the region so the outbound data(D_{out}) is assumed to be zero. Hence, with references to equation 6 the cost involved for S3 leading to an estimated operational cost is estimated to be:

$$C_{s3} = 1.92 \times 0.023 + (858 + 23,598 + 1,902) \times 5106 = 0.18$$

Also, we have used only one EC2 instance i.e $g4dn.xlarge$ which costs around 0.579 USD per hr. This estimate was precisely corroborated by the actual costs incurred, affirming the accuracy of our cost estimation model and the efficiency of our cloud-based data handling strategy illustrated in Figure 7. Therefore, this analysis demonstrates that the same equations can be seamlessly applied to estimate the cost of scalable operations for multi-camera feed sensors by using appropriate data measurements.

D. Performance vs Cost vs Concurrency Trade-off

In this study, we evaluate the performance and cost trade-offs of cloud services within the dynamic camera feed workload of our edge-cloud ecosystem. Our architecture utilizes scalable AWS services such as AWS IoT Core and Kinesis Firehose, with our algorithm deployed on EC2 instances for computation. We use three different types of GPU instances named g4dn.xlarge (I), g4dn.2xlarge (II) and g5.xlarge(III) (Table III). Total cost per inference is calculated using the cost model discussed in Section IV

Instance name	GPU	vCPU	GPU Memory (GB)	Price (USD/hr)
g4dn.xlarge	1	4	16	0.579
g4dn.2xlarge	1	8	16	0.828
g5.xlarge	1	4	24	1.208

TABLE III
GPU INSTANCES AND THEIR CONFIGURATION

Figure 8 shows the per-frame latency and on-demand cost per hour for varying concurrency observed with three different types of EC2 instances I, II, and III as discussed above. As shown in the figure, for a given $L_{SLA} = 0.75$ we observe that I, II, and III can serve a maximum of 15, 18, and 23 concurrences at the rate of 0.579, 0.828, and 1.208 USD per hour respectively. Also, we see that for a given concurrency, maximum latency is observed with instance type I while type III instance delivers better latency at a higher cost.

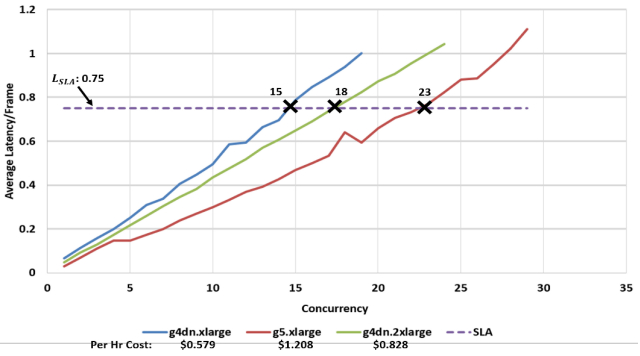

Fig. 8. Concurrent execution of feeds at different cloud hardware.

Based on this study, we can conclude that the proposed cost model in conjunction with the load-balancer in our architecture can assist in maintaining cost and performance trade-offs.

VII. CONCLUSION AND FUTURE WORK

In this paper, we demonstrated an innovative and scalable edge-cloud ecosystem for MCS-MOT applications. We designed an SLA-aware MCS-MOT system that leverages

content-aware camera feed features to balance the processing of dynamic incoming feeds between edge and cloud ecosystems. Efficient communication was achieved through the high-speed MQTT protocol, empowered by a 5G internet connection. We also implemented cost estimation for this edge-cloud ecosystem to handle unforeseen concurrent load injections. Our evaluation showed that the edge-cloud ecosystem outperforms edge-only deployments in terms of latency and throughput within the specified SLA. Additionally, the system provides cost benefits based on the incoming camera feed, ensuring cost-effectiveness.

Looking forward, this edge-cloud co-design can be leveraged in various deployment applications such as robotics, embedded systems, and mobile robotics. Furthermore, we can explore multi-cloud deployment with different vendors to optimize cost benefits and performance for real-time industrial applications.

REFERENCES

[1] "Amazon web services:," https://aws.amazon.com/.

[2] "Azure:," https://azure.microsoft.com/en-in/.

[3] "Gcp:," https://cloud.google.com/.

[4] Xin Wu, Wei Li, Danfeng Hong, Ran Tao, and Qian Du, "Deep learning for unmanned aerial vehicle-based object detection and tracking: A survey," *IEEE Geoscience and Remote Sensing Magazine*, vol. 10, no. 1, pp. 91–124, 2021.

[5] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao, "Yolov4: Optimal speed and accuracy of object detection," *arXiv preprint arXiv:2004.10934*, 2020.

[6] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 580–587.

[7] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg, "Ssd: Single shot multibox detector," in *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14*. Springer, 2016, pp. 21–37.

[8] Pranav Adarsh, Pratibha Rathi, and Manoj Kumar, "Yolo v3-tiny: Object detection and recognition using one stage improved model," in *2020 6th international conference on advanced computing and communication systems (ICACCS)*. IEEE, 2020, pp. 687–694.

[9] D Nathasha U Naranpanawa, Yanyang Gu, Shekhar S Chandra, Brigid Betz-Stablein, Richard A Sturm, H Peter Soyer, and Anders P Eriksson, "Slim-yolo: A simplified object detection model for the detection of pigmented iris freckles as a potential biomarker for cutaneous melanoma," in *2021 Digital Image Computing: Techniques and Applications (DICTA)*. IEEE, 2021, pp. 1–8.

[10] Alex Bewley, Zongyuan Ge, Lionel Ott, Fabio Ramos, and Ben Upcroft, "Simple online and realtime tracking," in *2016 IEEE international conference on image processing (ICIP)*. IEEE, 2016, pp. 3464–3468.

[11] Rudolph Emil Kalman, "A new approach to linear filtering and prediction problems," 1960.

[12] Nicolai Wojke, Alex Bewley, and Dietrich Paulus, "Simple online and realtime tracking with a deep association metric," in *2017 IEEE international conference on image processing (ICIP)*. IEEE, 2017, pp. 3645–3649.

[13] Yunhao Du, Zhicheng Zhao, Yang Song, Yanyun Zhao, Fei Su, Tao Gong, and Hongying Meng, "Strongsort: Make deepsort great again," *IEEE Transactions on Multimedia*, 2023.

[14] Wei Huang, Xiaoshu Zhou, Mingchao Dong, and Huaiyu Xu, "Multiple objects tracking in the uav system based on hierarchical deep high-resolution network," *Multimedia Tools and Applications*, vol. 80, pp. 13911–13929, 2021.

[15] Daniel Stadler, Lars Wilko Sommer, and Jürgen Beyerer, "Pas tracker: Position-, appearance-and size-aware multi-object tracking in drone videos," in *Computer Vision–ECCV 2020 Workshops: Glasgow, UK, August 23–28, 2020, Proceedings, Part IV 16*. Springer, 2020, pp. 604–620.

[16] Jiating Jin, Xingwei Li, Xinlong Li, and Shaojie Guan, "Online multi-object tracking with siamese network and optical flow," in *2020 IEEE 5th International Conference on Image, Vision and Computing (ICIVC)*. IEEE, 2020, pp. 193–198.

[17] Happiness Ugochi Dike and Yimin Zhou, "A robust quadruplet and faster region-based cnn for uav video-based multiple object tracking in crowded environment," *Electronics*, vol. 10, no. 7, pp. 795, 2021.

[18] Hongyang Yu, Guorong Li, Weigang Zhang, Hongxun Yao, and Qingming Huang, "Self-balance motion and appearance model for multi-object tracking in uav," in *Proceedings of the ACM Multimedia Asia*, pp. 1–6. 2019.

[19] Yomna Youssef and Mohamed Elshenawy, "Automatic vehicle counting and tracking in aerial video feeds using cascade region-based convolutional neural networks and feature pyramid networks," *Transportation Research Record*, vol. 2675, no. 8, pp. 304–317, 2021.

[20] Caglayan Dicle, Octavia I Camps, and Mario Sznaier, "The way they move: Tracking multiple targets with similar appearance," in *Proceedings of the IEEE international conference on computer vision*, 2013, pp. 2304–2311.

[21] Hong Zhang, Zeyu Zhang, Lei Zhang, Yifan Yang, Qiaochu Kang, and Daniel Sun, "Object tracking for a smart city using iot and edge computing," *Sensors*, vol. 19, no. 9, pp. 1987, 2019.

[22] Hirofumi Noguchi, Tatsuya Demizu, Misao Kataoka, and Yoji Yamato, "Distributed search architecture for object tracking in the internet of things," *IEEE Access*, vol. 6, pp. 60152–60159, 2018.

[23] Ronghua Xu, Seyed Yahya Nikouei, Yu Chen, Aleksey Polunchenko, Sejun Song, Chengbin Deng, and Timothy R Faughnan, "Real-time human objects tracking for smart surveillance at the edge," in *2018 IEEE International conference on communications (ICC)*. IEEE, 2018, pp. 1–6.

[24] Haifeng Gu, Zishuai Ge, E Cao, Mingsong Chen, Tongquan Wei, Xin Fu, and Shiyan Hu, "A collaborative and sustainable edge-cloud architecture for object tracking with convolutional siamese networks," *IEEE Transactions on Sustainable Computing*, vol. 6, no. 1, pp. 144–154, 2019.

[25] Siyan Guo, Cong Zhao, Guiqin Wang, Jiaqing Yang, and Shusen Yang, "Ec^{2}detect: Real-time online video object detection in edge-cloud collaborative iot," *IEEE Internet of Things Journal*, vol. 9, no. 20, pp. 20382–20392, 2022.

[26] Ching-Hu Lu and Kuan-Ting Lai, "Dynamic offloading on a hybrid edge–cloud architecture for multiobject tracking," *IEEE Systems Journal*, vol. 16, no. 4, pp. 6490–6500, 2022.

[27] Chetan Phalak, Dheeraj Chahal, and Rekha Singhal, "Sirm: Cost efficient and slo aware ml prediction on fog-cloud network," in *2023 15th International Conference on COMmunication Systems & NETworkS (COMSNETS)*. IEEE, 2023, pp. 825–829.

[28] Manju Ramesh, Dheeraj Chahal, and Rekha Singhal, "Multicloud deployment of ai workflows using faas and storage services," in *2023 15th International Conference on COMmunication Systems & NETworkS (COMSNETS)*. IEEE, 2023, pp. 269–277.

[29] Dheeraj Chahal, Surya Palepu, Mayank Mishra, and Rekha Singhal, "Sla-aware workload scheduling using hybrid cloud services," in *Proceedings of the 1st Workshop on High Performance Serverless Computing*, 2020, pp. 1–4.

[30] Yongji Wu, Matthew Lentz, Danyang Zhuo, and Yao Lu, "Serving and optimizing machine learning workflows on heterogeneous infrastructures," *arXiv preprint arXiv:2205.04713*, 2022.

[31] Jashwant Raj Gunasekaran, Prashanth Thinakaran, Mahmut Taylan Kandemir, Bhuvan Urgaonkar, George Kesidis, and Chita Das, "Spock: Exploiting serverless functions for slo and cost aware resource procurement in public cloud," in *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*. IEEE, 2019, pp. 199–208.

[32] Ratul Kishore Saha, Rekha Singhal, and Manoj Nambiar, "Camot: Content aware multi object tracking," in *Proceedings of the Third International Conference on AI-ML Systems*, 2023, pp. 1–9.

[33] Amazon, "AWS Kinesis.," Accessed May 16, 2024.

[34] Amazon, " AWS IoT Greengrass.," Accessed May 16, 2024.

[35] Amazon, "AWS S3.," Accessed May 16, 2024.

[36] Amazon, "AWS EC2.," Accessed May 16, 2024.

[37] John Canny, "A computational approach to edge detection," *IEEE Transactions on pattern analysis and machine intelligence*, , no. 6, pp. 679–698, 1986.