**Abstract**

The issue of object detection in remote surveillance using edge devices presents a complex scenario, largely as a result of the limitations inherent in edge computing settings and the requirements for instantaneous data processing. Current video surveillance systems exhibit proficient video capture functionalities; however, data analysis at the server level is impeded by constraints in transmission power and the availability of cloud computing resources. Consequently, Internet of Things (IoT) devices are primarily relegated to the role of data achievement. The Internet of Things (IoT) has been increasingly developed worldwide for the last five years. The need for businesses to use Computer Vision and Machine Learning in their IoT devices has grown significantly. These demands can come from using IP cameras, webcams, or anything requiring camera modules. People must transfer data from these devices to different personal computers (PCs) or mobile devices, especially in real-time with low latency, to get the information on time. In this article, we propose an architecture that helps businesses make IoT devices that can stream video and process the data in real-time to satisfy their demands using Edges AI Devices. The architecture is easy to implement and strong to serve. It is a flexible and secure architecture, which has already worked with some accelerators, having high speed and accuracy.

1. **Introduction :**

In recent years, we have witnessed the rise of trends using IoT devices in various fields. The number of businesses using IoT technologies has increased from 13 percent in 2014 to about 25 percent today, along with the number of IoT-connected devices that tend to be 43 billion 2023. IoT may become one of the most crucial things in the future because of its usefulness and capabilities. Many companies provide solutions to stream video from end to end in the market, making real-time streaming closer to users. However, the previous solutions are more suitable for medium and large-scale companies due to their expensive cost. Therefore, it is necessary to have an architecture for individuals and small companies to access. This paper proposes real-time processing with camera architecture to enable browsers to communicate “Peer-to-Peer” (P2P) without installing third-party plugins. The proposed system is based on Web Real-Time Communication (Web RTC), a web Application

Programming Interface (API) developed by the World Wide Web Consortium. Thanks to OpenCV, complicated tasks such as object detection, classification, and recognition can be performed. The main contribution of this paper is to propose an architecture that is easy to set up to solve real-world problems with high accuracy and speed. The operation speed of the proposed architecture must be over 30 frames per second. Previously, IoT Camera was typically used with cloud computing because of the resource and energy-constrained edge devices, while video processing tasks require much power and resources. Our architecture may be the solution for small companies and individuals who need a fast and reliable real-time image processing camera equipped with machine learning algorithms. Besides, the proposed system can be employed in portable, low-power wearable devices. The structure of this article is organized as follows. Section 2 provides the knowledge-related work concerning the Web RTC, Real Time Streaming Protocol (RTSP), and the accelerators used to test with our system–Google Coral and Intel Neural Compute Stick 2. Section 3 describes the components of our architecture in detail. Section 4 presents the experimental result tested with Google Coral and Intel Compute Stick. Finally, Section 5 concludes our paper.

**1.1. Need for Edge Computing**

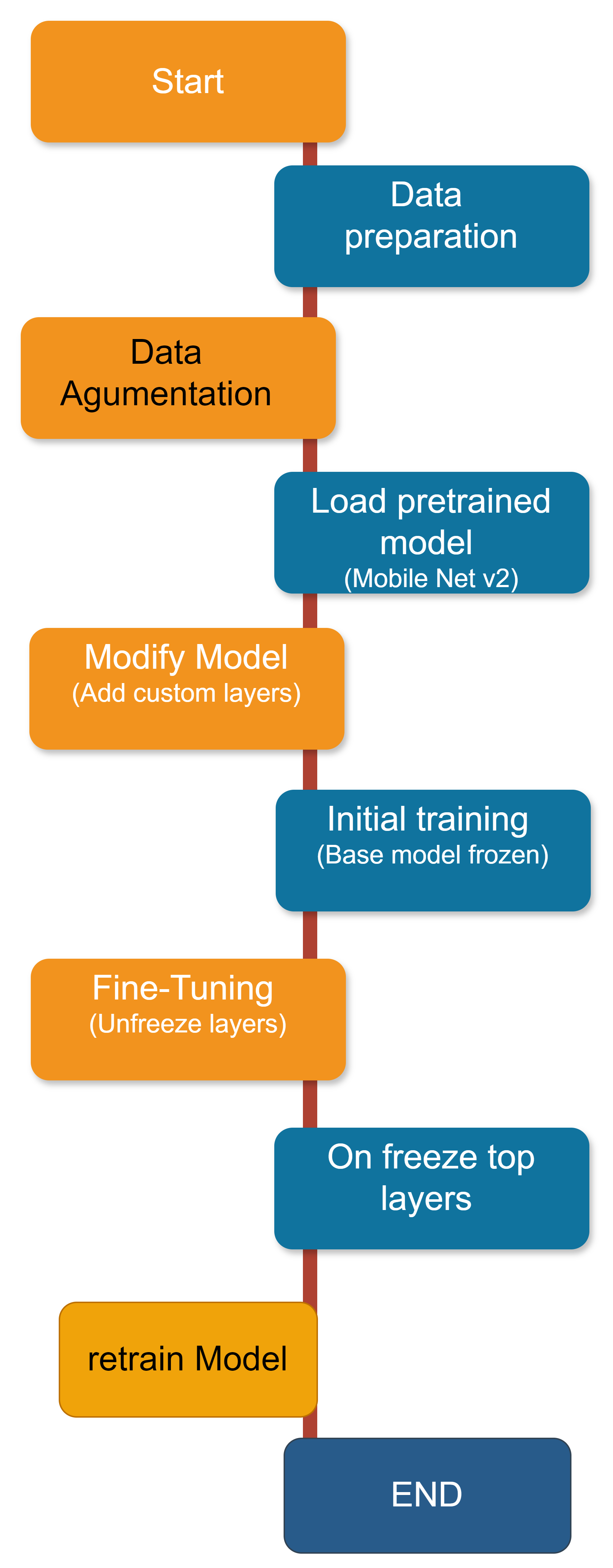
The computation could potentially be realized on the cloud by taking advantage of powerful and scalable computation resources that are available on-demand. However, bandwidth, latency, and privacy necessitate the use of the edge computing paradigm for streaming video analytics . To put this into context, a single H.265 1080p IP camera with a high video quality operating at 15 fps requires a 2.3 Mbps uplink and generates over 25 GB of data per day . Half a dozen cameras in a home or at a traffic intersection could easily saturate the local uplink capacity (typically 10 Mbps). Regarding latency, the camerato-cloud communication over the Internet is of the order of hundreds of milliseconds. For the pedestrian safety use case, a vehicle driving at a speed of 45 mph (72 kph) covers a distance of 60 ft (20 m) in a second. Detecting whether such a moving vehicle poses a threat to a pedestrian requires detecting events with tens of milliseconds of latency including computation and communication overheads. Regarding privacy, video streams are a rich source of information. Transmitting videos to a distant cloud data center could violate user expectations of privacy, and legal requirements such as GDPR and the guiding principles of the government use of surveillance technologies . Furthermore, if the video stream is accessed by unauthorized third parties, unintended information (for example, personal identities for the pedestrian safety use case) could be revealed. By performing video analytics at the edge close to the video cameras, communication is limited to local area networks, thus reducing the network latency. Furthermore, the aggregate bandwidth requirements are reduced due to the distributed processing at the edge enabling the scaling of video cameras. Moreover, sensitive video data can be confined to the privacy perimeter expected by the user (for example, a home or legal jurisdiction)

**1.2. Key Contributions**

In recent years, there has been a notable surge in research focused on SVA at the edge. Despite the widespread deployment of cameras in cities (for example, London, UK, has more than half a million IoT surveillance cameras) and private establishments, as well as the significant advancements in deep learning and AI-powered computer vision, a substantial gap still persists in effectively utilizing AI to analyze these camera streams in real-time to derive actionable insights. The research question that this paper seeks to answer is: what is the state-of-the-art in IoT edge streaming video analytics systems (IE-SVA)? The goal of this paper is to thoroughly analyze IE-SVA systems as reported in the research literature, so as to provide clarity to researchers and industry practitioners on the progress to date, the techniques employed, and the additional research and development that needs to be performed to further the field. To achieve these goals, we begin by outlining the characteristics of an ideal IE-SVA system. By using this ideal system as a framework, we analyzed existing research literature on video analytics along 17 unique dimensions. The analysis is presented in tabular format with 37 reported works listed in chronological order (2015–2023) to help the reader readily understand the research progress made by these systems along different dimensions. These systems are also classified according to their primary research focus, allowing readers to easily access works that delve into specific techniques aligned with their interests. Based on this analysis, we propose research directions for the short, medium, and long term.

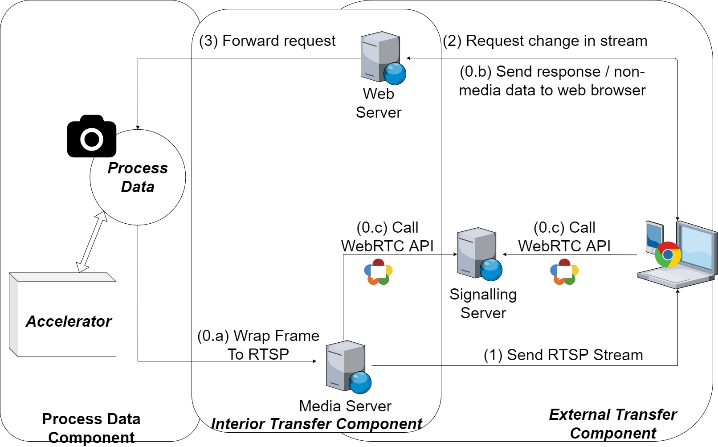
**2. Methodology**

An edge impulse platform is utilized for the training of the Raspberry Pi model with object detection. The flowchart illustrates a methodical approach for developing and optimizing a deep learning model, utilizing the MobileNetV2 architecture within the Tensor Flow framework. As shown in Figure 1, the process commences with the Data Preparation phase, where the dataset is curated and prepared for processing, ensuring its compatibility for model ingestion.



**Figure 2. Implementation workflow with Transfer Learning**

Following this, the Data Augmentation phase is implemented to improve the model’s capacity to generalize from the training data by introducing variations in the dataset. This is accomplished through methods like random flipping, cropping, and brightness adjustments, thereby replicating a wider array of data variances. The workflow advances to the Load Pretrained Model stage, where the MobileNetV2 architecture is instantiated with preloaded weights. This method leverages transfer learning, enabling the model to utilize knowledge obtained from a previously trained context, thereby improving its learning efficiency and efficacy. During the Modify Model stage, the architecture is adjusted by adding custom layers on the MobileNetV2 base, tailoring the model to the specific task at hand. This customization is crucial in refining the model’s output to meet the desired objectives. The Initial Training phase encompasses training the customized model while maintaining the base MobileNetV2 layers frozen, directingthe learning process towards the newly added layers. This step is essential for the initial adaptation to the task-specific features without disrupting the prelearned representations in the base model. The workflow then movesto the Fine-Tuning phase, where specific layers from the base model are unfrozen in the Unfreeze Top Layers step, enabling the fine-tuning of deeper representations within the model. This phase plays a vital role in enhancing the model’s performance, aligning it more closely with the task-specific intricacies. In the Retrain Model stage, the model undergoes additional training in this new configuration, allowing the unfrozen layers to adjust their weights in synchronization with the custom layers added earlier.The progression of data from the input (specifically the first Conv2D layer) throughout the network until it reaches the output (which is the Dense layer with softmax activation). The utilization of the Adam optimizer facilitates the adjustment of the network weights by the computed gradients.

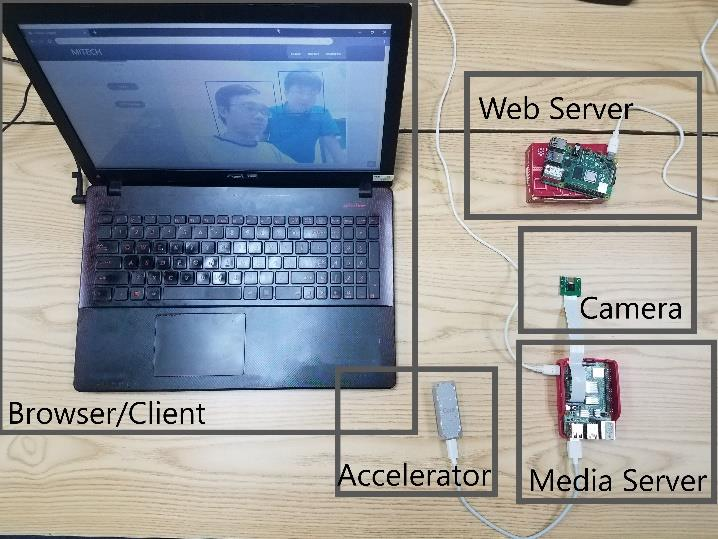


**Fig. 3. Proposed data Component Architecture**

**2. PROPOSED ARCHITECTURE**

**2.1 General Architecture**

Figure 3 shows our proposed architecture, which can be split into three main components: (1) Interior Transfer Component, (2) Exterior Transfer Component, and (3) Process Data Component (see Figure 3). The details of each component will be described in 3.2 to 3.4. As shown in Figure 3, our architecture first captures the video from the camera frame by frame. Later, with the help of the accelerator, our system processes frames and then wraps them to the RTSP stream (0.a) to send to the media server. After this step, the media server is ready. Simultaneously, in the Exterior Transfer Component, devices are connected to a web server and fetch the website content (0.b) (includes functions calling to Web RTC API). Next, browsers use these APIs (0.c) to establish a P2P connection to transfer the media stream directly. After the connection is established, browsers can receive the RTSP stream and handle the RTSP stream with Web RTC to watch on browsers. That is the endpoint of the traditional Web RTC application. However, our architecture provides a solution to respond to the requests of users to blur objects, object detection, and object recognition. Browsers sent these requests to the web server. Then, the web server forwarded it to Process Data Component, proposing to change the properties of the stream to complete requests. Consequently, the stream to the end-user will be processed and just be shown on the browser. That is the endpoint of our web application cycle. Our architecture uses Web RTC-based streamers to handle the video with Web RTC. However, the original itself has a JavaScript library for object detection. Therefore, this cannot go further with our custom model with the recognition problems. Our architecture solved this by creating the Process Data Component that enables users to customize their algorithms to solve their problems. Also, they can directly control the way data are processed as well as the performance of the system. Moreover, our architecture allows users to combine their projects in Raspberry Pi with different accelerators to save much time for developers to change or demo their projects to customers. Furthermore, to protect the stream from being attacked and prevent users with no permission from accessing the raw video, we decided to process directly on the server and send the processed frame on stream, which may reduce the possibility of leaking sensitive information on the line. This approach increases the CPU load on the server side but decreases the CPU users use to process and enhance the security. To visualize the model, we captured our current system, which we used to test within the paper in Figure 4.



**Fig. 4. This is Our system**

**2.2 Exterior Transfer Component**

This component is an excellent open-source project licensed under the “Un-license” license, and the code is public on GitHub. Hence, it provides the ability to use multiple purposes under no circumstances. Since using Web RTC, our architecture has its impressive property – low latency. The stream within the architecture can be processed and then sent to the end-user in real-time.

**2.3 Interior Transfer Component**

Our Media Server uses v4l2rtspserver, which does not natively support the format that results from processing images with Open CV. Therefore, to prepare a stream for the Media Server, after getting a processed frame from the accelerators, we must compress the resulting frame with the h264 standard to make it available with the v4l2rtspserver. After this, we can successfully send clients the frame as the RTSP Stream.

On the way to request sending from browsers to a server, we implemented an API server to get the parameters passed to the Process Data Component to change the stream properties.

**2.4 Process Data Component**

This component requires an external module (Google Coral and Intel Compute Stick 2 in our architecture). The camera will capture the frame and pass it as a parameter to the accelerator to ensure the system’s stability. Raspberry Pi does not have to process the image with such a complicated task. This approach may reduce the overload of Raspberry Pi and enhance the system’s performance due to the fast computing capability of Google Coral and Intel Compute Stick.

**3. EXPERIMENTS**

The following result was seen in a laptop with Intel 6300HQ, 8GB RAM, using Nvidia Geforce 940MX on Windows 10. The size of the image was 640 × 480 px during our experiments. We also investigated the latency of the proposed system.

**3.1 Google Coral experiment**

This subsection has tested with Google Coral attached to Raspberry Pi 4 directly to Raspberry Pi 4 via the USB 3.0 standard. The experiment was conducted in different environmental conditions, indoors and outdoors. This test intends to ensure that the system can work in different conditions, indoors and outdoors, with low light conditions. This experiment investigated the system’s operation and the camera module from Raspberry Pi in low light conditions. Google Coral, the Edge TPU device, is connected. The latency is under 1 second, from the Raspberry Pi camera input passes through our architecture to the end-user. We have tried to detect multiple faces in these conditions to test the system’s compatibility. Our manuscript calculated the Frame rate Per Second (FPS) based on the processing speed, as it can handle each frame in less than 20 ms. Figure 10 shows the experimental result with the indoor conditions and stable light. The system could detect all faces in the frames. The FPS in this test was over 55.

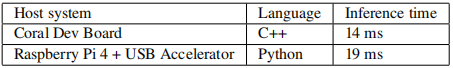


**Fig. 5. Result from Google Coral With Indoor Scenario**

We also tested with outdoor conditions. Figure 6 presents the result of testing at sunset (06:19 PM in Vietnam) when the light condition is considerably worse. In this case, the architecture with

Google Coral can still operate with an acceptable result. The faces are still detected, but the accuracy went down. However, the system works stable as the theoretical FPS can be maintained at high speed. Considering the processing speed as a crucial criterion, the performance of our system qualified for real-time demand. With Coral Edge TPU as the accelerator, our system maintains a stable processing speed of 50 FPS in face detection. Table I below compares the results from the Coral benchmark test and our results. Despite having a slightly slower speed than the benchmark tests of Coral Dev Board, our result is still considered coequal, as Coral

Table 1. Time per inference, using Mobile Net v2 SSD, quantized to 8-bit precision.



claims that benchmarking with Python results in slower speed due to overhead from Python. This result shows the effectiveness of AI accelerators in general and Edge TPU, particularly in our architecture.



**Fig. 6. Result from Google Coral With Outdoor Scenario**

**3.2 Intel Compute Stick 2 experiment**

Besides Google Coral, we employed another state-of-the-art edge device called the Intel Compute Stick 2 (ICS2) to investigate the compatibility of this device with our architecture. In this experiment, the ICS2 and the camera were put in Japan, and the server was set up in Vietnam. From the experimental results, the system may still work with Intel Compute Stick 2, providing a latency of less than 1 second, including line latency from Japan to Vietnam. Therefore, it can meet the real-time requirement.