**Edge AI for real time image and video analysis in IOT devices.**

**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.

**Keywords :**

IoT, Computer Vision, OpenCV, Machine Learning, AI, WebRTC, RTSP

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

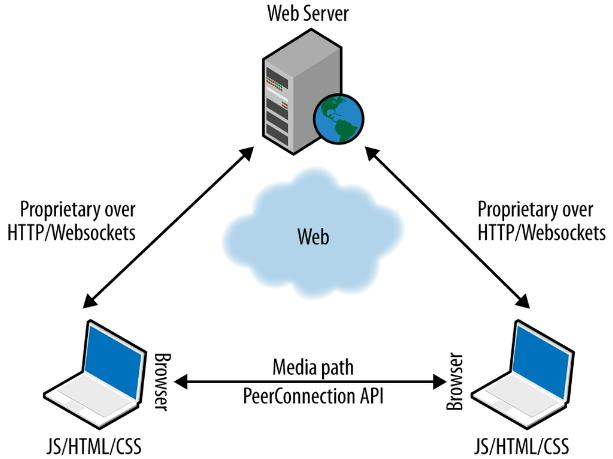
must be over 30 frames per second. Previously, IoT Camera was typically used with cloud computing because of the resource and energy-constrained edge devices, while video processing tasks require much power and resources. Our architecture may be the solution for small companies and individuals who need 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 Web RTC**

Since its first appearance in 2011, Web RTC has always claimed to be a revolutionizing way that helps users communicate, both in the consumer and enterprise world. WebRTC has attracted many researchers. A simple search with the keyword “WebRTC” in Google Scholar found nearly 39 thousand results. That means there are nearly 3 thousand studies related to WebRTC per year. WebRTC is written in JavaScript so that it can be used and supported by almost all currently used browsers. Web RTC, as it is called, can support video streaming in real-time context with a P2P connection, making it one of the fastest solutions with zero latency and having a high-quality image. Web RTC, with its capabilities, can also be applied in many applications, such as streaming from the camera or sharing the screen in a native web application. However, although various papers are using Web RTC as their solution to stream video, to our knowledge, no current research has combined AI with image processing directly on stream.



**Fig. 1. Web RTC Model**

Figure 1 shows operations of Web RTC. It is shown that Web RTC handles only the media stream between two browsers/clients. A protocol named “Signaling” establishes a P2P connection to transfer media streams directly from end-users to end-users without a server to stream as fast as possible. One of the main problems Signaling has to resolve is to overcome the Network Address Translator (NAT) to get the correct IP with the correct port. In this situation, the TURN and STUN servers will be implemented to access each user, which can act to establish the peer-to-peer connection. Web RTC provides three main APIs, including get User Media(), RTC Peer Connection(), and RTC Data Channel(). In this situation, the get User Media() API is to get access to the camera and microphone. After this step, IP and port are collected to create connections despite NATs or firewalls. When the P2P connection is constituted, the other two APIs are called to share the media stream. On the way to decide whether to choose Web RTC or other technologies for our architecture, following Bart Jansen and his partners, we found that Web RTC consumes a reasonable bandwidth as presented in Figure 2. This result is acceptable for most areas globally, with an average bandwidth of 31.95 Mbps with mobile devices and 74.32 Mbps with fixed broadband, as shown in Figure 3. Therefore, Web RTC is the possible solution for video transferring peer-to-peer, even within a small group of end-users. Performance is one of the main things that needs to be investigated. In this perspective, Web RTC shows impressive results. In that research, they create a new user every second to test the latency if 1–200 concurrent users join to view. The result confirms the real time capability of Web RTC (less than 500ms) with a size of 180 simultaneous users. Web RTC is an excellent solution for real-time media streaming.

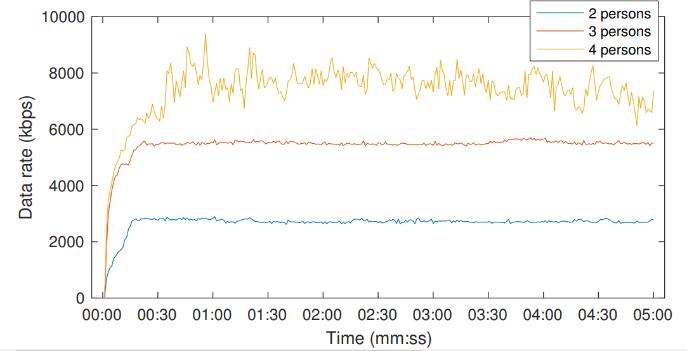


Figure 2 illustrates the average data rates for engaged calls involving two, three, and four participants.

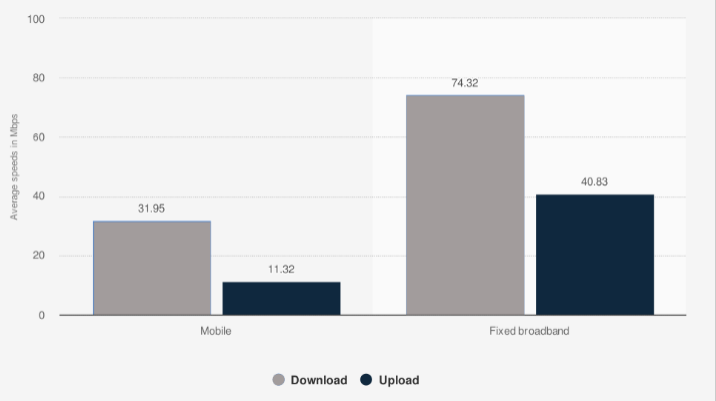
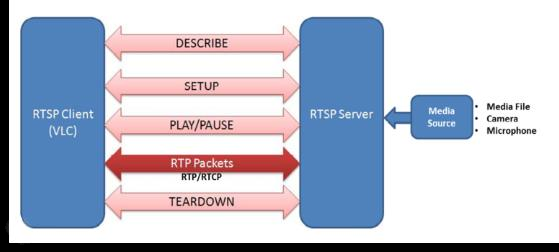


Fig. 3. illustrates the global average download and upload speeds for mobile and fixed broadband as of January 2020, measured in Mbps.

**1.3 RTSPs**

The media is encoded with the h264 standard to transfer files between users and then transmit on the line with Real-Time Streaming Protocol (RTSP) as shown in Figure 4. RTSP is a protocol that provides essential functions to control the video’s flow and is combined with the Real-Time Transport Protocol (RTCP) to distribute the media flow.



**Fig. 4. illustrates the RTSP Model.**

RTSP is commonly used in camera devices (IP cameras or security cameras) to transfer media to a server or another client. As it was born to be a real-time streaming protocol, RTSP is suitable for use cases in many fields. It is widely used in IP cameras, IoT, and mobile devices. Although RTSP is easy to use and implement, it is not natively supported by HTML5 browsers. Hence, in our architecture, we must use Web RTC to handle and display the RTSP stream on the browsers. With the help of Web RTC, we can display the RTSP stream in real-time with close to zero latency.

**1.4 Google Coral**

Developed by Google, Coral is a complete toolkit for developers to build products with local Artificial Intelligence (AI). It consists of:

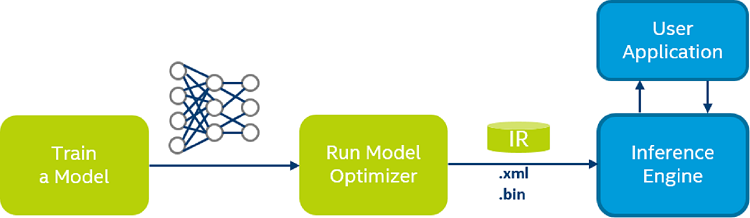
* Dev Board: a single-board computer with a removable systemon-module featuring the Edge Tensor Processing Unit (TPU).
* USB Accelerator: an Edge TPU USB compatible with Windows 10, macOS, and Debian Linux (including Raspberry Pi) host computer.
* Mini PCIe Accelerator, M.2 Accelerator A+E Key, M.2 Accelerator B+M Key: PCIe devices that enable easy integration of the Edge TPU into existing systems; support Debian Linux host computer.

The Edge TPU (Tensor Processing Unit) coprocessor is the whole Coral ecosystem’s heart. A TPU is an AI accelerator applicationspecific integrated circuit explicitly built for neural network operation and particularly compatible with the Google TensorFlow deep learning framework. The Edge TPU is designed for a high volume of low-precision computation (as low as 8-bit precision), which authorizes an inferencing speed of 4 trillion operations per second while only being powered by a supply of 2 watts. The coral USB accelerator is chosen because it is compatible with Debian Linux and other popular operating systems such as Windows 10 and macOS. Besides, being powered by a USB port makes it easier for Google Coral to attach to any edge devices or PC machine, while the other accelerators with an M.2 connector make it more difficult. A USB 3.0 port preferably powers the USB accelerator to get the best result, as USB 2.0 is still compatible, but the speed is much slower. The USB accelerator, like every other Coral brand device, supports only the Tensor Flow deep learning framework. Thus, it is easier for developers, especially those with Tensor Flow experience, to build and deploy deep learning models to their systems. Besides, with everything wrapped in one enormous hardware-software ecosystem, Coral products are well-optimized and perform excellently in inferencing speed. In their “comfort zone” (which means running deep learning models trained by Tensor Flow, quantized to 8-bit precision, using Tensor Flow Lite APIs), Coral Edge TPU devices can perform an image classification task at approximately 400 FPS using version 2 of Mobile Net architecture pre-trained on ImageNet dataset.

**1.5 Intel Compute Stick 2**

We can deploy many complex graphs on a small mobile device, such as Raspberry Pi, a single-board computer with high performance computing. To improve the performance of these devices, Intel provides the “Intel Neural Compute Stick” series. Neural Compute Stick 2 is a USB accelerator powered by the Intel Movidius X VPU to deliver industry-leading performance, wattage, and power. It supports Open VINO, a toolkit that accelerates solution development and streamlines deployment. The Neural Compute Stick (NCS) 2 offers plug-and-play simplicity, support for common frameworks, and out-of-the-box sample applications. Use any platform with a USB port to prototype and operate without cloud computing dependence. The Intel NCS 2 delivers 4 trillion operations per second with an 8X performance boost over the first generations. Today, the Intel Compute Stick with Open VINO is becoming more popular as it is convenient and has excellent computing capability. Intel Neural Compute Stick 2 is synchronized with the Open VINO toolkit. The toolkit helps quickly deploy deep neural networks and maximize device performance using Intel hardware (such as Intel Neural Compute stick) to extend Computer Vision workloads. With the support of Intel in both hardware and software, a deep learning model is easily embedded in many mobile devices. Therefore, researchers and developers can develop many products in AI and related fields. With the workflow of the toolkit, there are many ways to deploy deep learning models. For instance, Intel has developed an integrated module with OpenCV, called Open CV, with Open VINO Inference Engine. The module allows us to read models directly from Tensorflow, Caffe. Nevertheless, it is preferable to use Intermediate Representation (IR) to enhance performance, accuracy, and speed. IR format has an XML file to store model architecture and a bin file to store the model weights. Since we have a deep learning network, we can easily convert to the IR format by Model optimizer API.

IR format can be deployed efficiently in object detection and image. Fig 5 presents OpenVINO workflow.

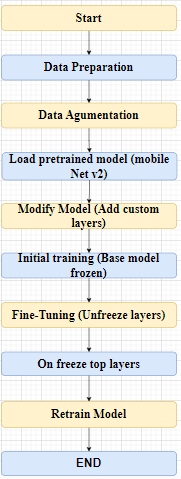


**Fig. 5. illustrates the workflow of OpenVINO.**

Intel document provides a full tutorial about setting up, running, and evaluating the deep learning model. Intel also has a large, free pre-trained model hub, from which we can use many deep learning models trained with massive datasets and can reach an accuracy above 98%. In this field, various researches about using a pertained model to solve problems such as recognition, detection, and segmentation with high accuracy. The other approach is to use Intel devices FPGAs to optimize deep learning models and deploy them in specific applications. In our problem, we provide a solution with deep learning embedded in the real-time interacting system. To optimize the framerate and the deep learning model’s accuracy, we must determine the communication method for real time responding.

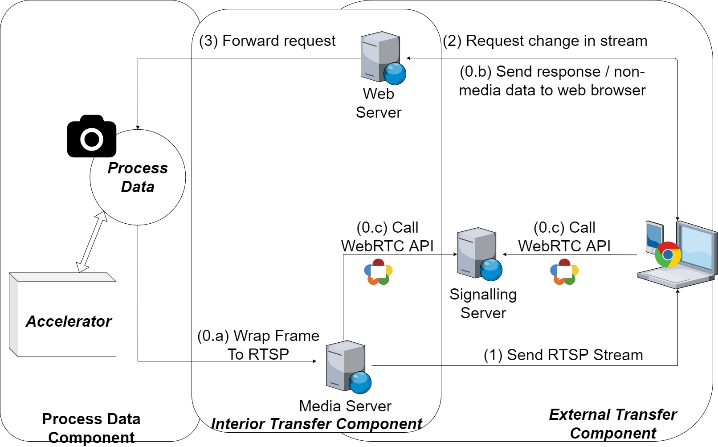
**2. Methodology**

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 6, the process commences with the Data Preparation phase, where the dataset is curated and prepared for processing, ensuring its compatibility for model ingestion.



**Fig. 6. 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 Pertained 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, directing the 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 moves to 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 soft max activation). The utilization of the Adam optimizer facilitates the adjustment of the network weights by the computed gradients.

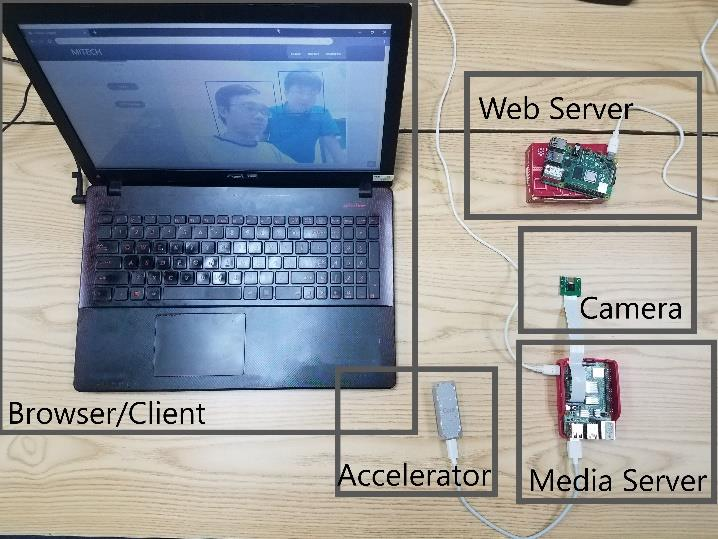


**Fig. 7. Proposed data Component Architecture**

**3. PROPOSED ARCHITECTURE**

**3.1 General Architecture**

Figure 7 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 7). The details of each component will be described in 3.2 to 3.4. As shown in Figure 7, 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. 8. This is Our system**

**3.2 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.

**3.3 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.

**3.4 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.

**4. 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.

**4.1 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.

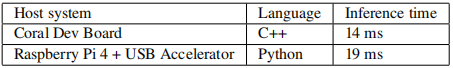


**Fig. 9. Result from Google Coral With Indoor Scenario**

We also tested with outdoor conditions. Figure 9 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. 10. 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.

**5. CONCLUSION AND FUTURE WORK**

Throughout the paper, we demonstrated our real-time architecture, which is fast, reliable, flexible, and secure. Our architecture was implemented within a local area network (LAN) and showed preliminary results. The architecture is tested with Google Coral, Intel Compute Stick 2 having acceptable results, especially in Google Coral with the impressive 55 FPS. Our system also worked in Ubuntu OS when we tried. However, we mainly focus on the embedded systems with flexibility and power-saving. Hence, the architecture can be developed to work cross-platform with high performance. Future research will improve the operation speed and make the architecture work on the Internet. We will test the system with the higher resolution camera, optimize the system to work more efficiently, and try with more accelerators to ensure the compatibility

**6. REFERENCES**

[1] J. Fredrik Dahlqvist, Mark Patel, Alexander Rajko, Growing opportunities in the Internet of Things, McKinsey & Company, 2019.

[2] S. Loreto, Salvatore Simon Pietro Romano, Real-Time Communication with Web RTC Peer-to-Peer in the Browser, O’Reilly Media, 2014.

[3] A. Johnston, D. Burnett, Web RTC: APIs and RTCWEB Protocols of the HTML5 Real-Time Web, Digital Codex LLC, 2012.

[4] B. Jansen, T. Goodwin, V. Gupta et al., Performance evaluation of Web RTC-based video conferencing,

[5] J. C Lement, Average global mobile and fixed broadband download & upload speed worldwide 2020, 2020.

[6] B. Garc´ıa, F. Gortazar, L. L ´ opez-Fern ´ andez et al, Analysis of ´ video quality and end-to-end latency in WebRTC, 2016 IEEE Globecom Workshops (GC Wkshps), Washington, DC, USA, 2016.

[7] A. Ahamed, “iWave Embedding Intelligence” [Online]. Available:https://www.iwavesystems.com/rtsp-mediastreaming-server-on-windows-embedded-compact-7-wec7. [Accessed 12 June 2020].

[8] Coral, Google, [Online]. Available: coral.ai. [Accessed 12 June 2023]