Real-Time Object Detection with Automatic Switching between Single-Board Computers and the Cloud

Ahmed Osman, Raouf Abozariba, A. Taufiq Asyhari, Adel Aneiba, Ambreen Hussain,
Bidushi Barua, and Moazam Azeem
School of Computing and Digital Technology, Birmingham City University, UK

E-mail: {ahmed.osman, raouf.abozariba, taufiq.asyhari, adel.aneiba, ambreen.hussain,
bidushi.barua, moazam.azeem}@bcu.ac.uk

***Abstract*—We present a wireless real-time object detection system utilizing single-board devices, cloud computing platforms and web-streaming. Currently, most inference applications statically perform tasks either on local machines or remote cloud servers. However, devices connected through cellular technologies face volatile network conditions, compromising detection performance. Furthermore, while the limited computing power of single-board computers degrade detection correctness, excessive power consumption of machine learning models used for inference reduces operation time. In this paper, we propose a dynamic system that monitors embedded device's wireless link quality and battery level to decide on detecting objects locally or remotely. The experimental results show that our dynamic offloading approach could reduce devices' energy usage while achieving high accuracy, real-time object detection.**

***Index Terms*—Machine learning, WebRTC, object detection.**

I. Introduction

In recent years, video surveillance systems have become common in many environments. Government agencies, businesses, and even fire safety agencies have recently turned to video monitoring to enhance public security and surveillance solutions [1]. The widespread availability of low-cost cameras and high-speed wireless networks has enabled economically and technically to deploy many vision techniques for security and safety purposes. Raspberry Pi [2] and other low-cost, small-sized Single-Board Computers (SBCs) are used to build applications that solve a wide range of real-world problems. Utilising a camera with an SBC can provide the functionality of a smart surveillance system.

Due to the lower cost and size, SBCs have CPU, memory, and power at a reduced capacity. As a result, running computer vision algorithms that normally require high-end GPUs on SBCs, would be challenging and would not generate the same performance. Many research studies have shown a variety of techniques to solve this challenge, including designing more efficient deep learning networks such as TensorFlow lite [3], creating computing add-on modules such as Google coral [4], and developing dedicated chips and processors like Neu Pro [5]. These solutions work well when SBCs are near power supply or the performance requirement is not so demanding. However, when powered by a battery, it is challenging for an SBC to run powerful computer vision algorithms.

Offloading tasks to the cloud is another solution for extending the battery life of mobile devices while maintaining performance of real-time object detection. The idea of using cloud infrastructure to assist low-resource devices was first introduced in [6], and it has since inspired many projects [7]–[9]. It is also becoming more accessible as more cloud computing services provide deep learning frameworks at affordable prices. Furthermore, the networks on these cloud machines are usually of high bandwidth, making the cloud suitable for real-time applications. However, most applications running on both the edge computers and cloud are unable to automatically change between these platforms without any input from users [10]. To overcome the challenge of limited battery resources of SBC and to deploy the cloud infrastructure to its full potential, this paper proposes a system to allow automatic switching between the platforms to perform uninterrupted real-time object detection while operating under various network conditions.

In addition, latency and high availability services are two important metrics for many critical industrial and healthcare applications. To understand the impact of auto switching between platforms on various networks, we provide detailed analysis of platforms communicating through common wireless networking technologies, such as Wi-Fi and 4G. The main objectives of the proposed system is to deliver real-time object detection with ultra-low latency streaming, while keeping energy consumption low, adding battery life of SBC by shifting heavy operations to the cloud when the battery of the SBC is running low.

Our main contributions are the use of WebRTC APIs to achieve ultra-low latency streaming for real-time object detection and the integration of WebRTC streaming and machine learning automatic switching between SBC and cloud computer based on the SBC's network link quality and battery level.

II. Background

A. Machine learning

Machine learning is the basic principle that drives object detection. For example, before objects within an image or a video can be identified, the data is processed through computer vision and fed into a model. Using machine learning algorithms or deep learning algorithms, conclusions are made. The results are laid out as a form of output image or video. The processed image or video includes drawing bounding boxes around the objects detected. This allows us to locate where the target objects are (or how they move through) in a given scene. There are two types of object detection algorithms available. One is two-stage, which divides object recognition and object location into two separate processes. Common algorithms include Faster-RCNN [11] and R-CNN [12]. They have a low recognition error rate and a low rate of false recognition. Their low speed makes them unsuitable for real-time detection applications. Another method, known as the one-stage algorithm, has emerged to tackle this challenge, and it can recognise and locate objects in a single step. Examples of this method are SSD [13], Yolo and YoloV4 [14]. Their recognition speed is faster than the two-stage method, and this allows them to be used for real-time object detection applications. Their accuracy rate is also comparable to that of a faster R-CNN. However, one downside of the one-stage method is that the CPU processing demand is too high, making it prohibitive for use in mobile SBCs. A suitable algorithm for embedded devices is MobileNets [15]. MobileNets is from the TensorFlow family of mobile-first computer vision models aimed to enhance accuracy while keeping in mind the limited resources available for SBC devices. We adopt MobileNets to perform object detection for the above stated rationale.

B. Streaming protocols

A streaming protocol is a set of rules that define how data such as video and audio can be transmitted across the Internet from one device or system to another [16]. An encoder generally starts the transmission, which is then ingested by a media server before transmission. Real-Time Streaming Protocol (RTSP) [17] and Real-Time Messaging Protocol (RTMP) [18] are frequently used on the encoder side. Dynamic Adaptive Streaming over HTTP (DASH) [19] and HTTP Live Streaming (HLS) [20] are used on the playback side. WebRTC is a new protocol that can be used on both sides, the encoder and playback sides. The proposed system in this paper uses WebRTC because it is a technology that allows direct communication between the devices without installing applications or software. WebRTC is also more effective than RTMP or RTSP when it comes to Quality of Service (QoS) and Quality of Experience (QoE) for browser-based applications [21]. For instance, RTMP is not designed for scalability and has intolerable latency. On the other hand, RTSP does not scale as well as WebRTC and does not support HTML5, the current standard for structuring and presenting web content. We deploy WebRTC due to its easy scalability and offered low latency.

C. Computation offloading to cloud servers

Most of the past computation offloading research has primarily focused on static offloading to the cloud [22], [23]. However, the idea of automatically offloading between the local device and the cloud is still open. This paper tries to fill the gap by proposing an object detection system that will automatically switch between SBC and the cloud, while minimizing the transition time. In this section, we look at how other researchers use technologies related to achieving energy-efficient applications for mobile computing systems as well as other limited resource computing systems such as the Raspberry Pi. Even though the research discussed in this section is mostly based on mobile device offloading, the frameworks used can also be applied to other limited resource computing systems.

The decision to offload a system or part of a system is usually made during the system design. For example, if the system includes a part that requires significantly high computation power and energy than what the local device can handle, then that part is typically offloaded. Phone2Cloud [22] is an example of an application that uses a static approach to determine whether an application should run locally or on the cloud. The increasing number of more advanced smartphone apps and IoT software, such as machine learning and gaming, has prompted the development of this framework. The project focuses on managing smartphone energy consumption in relation to the applications that use it. It achieves this by reducing the device's computing requirements. For example, to support smartphones and IoT devices, it uses a remote execution environment for applications that consume excessive energy. Phone2Cloud's system architecture comprises seven key components that all work together to decide whether to offload to the cloud or not. Bandwidth, resource monitor, remote and local execution managers are all used in the decision-making process. Since Phone2Cloud focuses on lowering smartphone energy usage, it is a more energy-efficient solution than the proposed offloading method in this paper. The offloading approach presented in this paper focuses on offloading the system to the cloud in the shortest amount of time possible. It accomplishes this by shortening the decision-making process, as opposed to Phone2Cloud, which must monitor many resources before deciding.

MAUI [23] provides a system for optimising mobile device energy consumption by dynamically offloading heavy computation tasks to a remote resource. It does this by only offloading computation to the cloud when it is effective in terms of boosting performance and lowering energy usage. This framework predicts the execution time of methods both locally and remotely using a history-based approach. If the prediction shows that offloading tasks are better in reducing energy consumption, tasks are offloaded to the cloud. When offloading data and computation, this technology, similar to Phone2Cloud, primarily considers energy savings only, other

measurements, such as application performance or latency, are ignored by this framework. As a result, the solution suggested in this paper is not as energy efficient as MAUI, but it is more responsive to the user and its distributed nature promotes high availability and shorter downtime (see Fig. 6).

III. Proposed System

We implemented a real-time object detection system which can automatically and dynamically select the best resources. The location of where the object detection will take place is determined by the current state of the local device. For example, if the internet connection is stable, the video stream is sent to the cloud, and object detection is performed on the cloud virtual machine (VM). However, if wireless connections are intermittent and unavailable, it will automatically switch back to the local device where object detection will be performed. Furthermore, if the device is running on a low battery ($\sim$ 20%), the object detection will run on the cloud Virtual machine (VM) in order to save the remaining battery. SBC device, such as a Raspberry Pi attached to a Unmanned Areal Vehicles with a rechargeable battery, is a simple example of the proposed system. The ideal scenario is to perform all the tasks and computations on the cloud VM.

A. Streaming

We use WebRTC for streaming, a free, open-source protocol that allows web browsers and mobile apps to communicate in real-time (RTC). We integrate WebRTC with machine learning such as TensorFlow 2 and Tensorflow Lite. We use the getUserMedia() [24] method to directly access the user's camera through the browser. We then capture an image from the camera using a canvas [25]. We also utilize the HTMLCanvasElement.toBlob [26] method which enables us to convert our canvas images into blob. After the conversion, we use HTTP POST [27] to send data to the Object Detection API [28]. The Object Detection API is comparable to the official demo notebook [29]. The main difference is that instead of displaying the output in GUI-based environments, we wrap our output into a JSON object [30] and send it through the web. Once we get JSON response from the Object Detection API, we create another canvas that will draw boxes using the Object API outputs that shows the detected objects and their locations in the frame.

IV. System Implementation

A. Requirements

Functional requirements:

- The Raspberry Pi should process high resolution streaming and continuously monitor its battery status.
- When the network is stable, the Raspberry Pi should send images to the cloud through 4G network for object detection.
- If the network fails to meet the required quality of service (e.g., low signal to noise ratio and signal strength), the SBC should perform object detection locally without sending frames to the cloud.

Non-Functional Requirements:

- Real-time video analytics: the aim is to minimize the sum time of capturing a video, detecting the object, and displaying the results to the end-user on a web browser.
- Offloading: The end-user should not notice when the system switches between the local and cloud processing (minimizing the transition time).
- Energy-Efficiency: The system should enable effective energy management to increase the battery life of embedded devices.

B. System design

This section describes the system design as illustrated in Fig. 1. There are four layers:

- Use cases layer: This layer is used for collecting images/video from the observed site and includes high-resolution camera, which is connected to the edge layer through a USB cable and a video capture card.
- Edge layer: The servers running on the edge computer include a machine learning server and a media server. When the cloud is used, the streaming server is the only server that runs on the edge, saving battery service lifetime. The Raspberry Pi connects to the cloud layer wirelessly through the built-in Wi-Fi card or using a Sixfab 4G HAT as well as the internet. Please note that we use Wi-Fi only for benchmarking.
- Cloud layer: The cloud hosts machine learning algorithms which requires considerable processing resources, a commodity the edge computer can not offer. Given the availability of resources, object detection is preferred to be performed on the cloud, except when the network conditions are highly unstable. The only server that runs in the cloud is the object detection server. The streaming server (or media server) can also be hosted on a cloud computer instead of the edge computer; however, because the edge is closer to the end-user, the streaming server is hosted on edge computer to reduce latency and enhance image quality.
- Application layer: The end-user interacts remotely with the edge computer through a Web page hosted on a Node.js server. The server also enables streaming through creating communication sessions. The Node.js server also hosts interactive pages which can display real-time video streams. When required, the user can also send appropriate commands to the machine learning server. For example, end-user can set the threshold and enables or disables object detection.

C. System components

Fig. 2 shows the core components of the proposed system, the edge subsystem and the cloud subsystem.

Media Server: The media server frontend is made up of JavaScript, HTML, and CSS, while the backend hosts Node.js with Express. The primary function of the media server is to enable WebRTC communication and create a user-friendly interface.

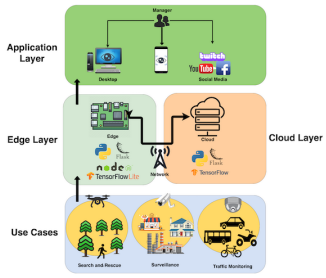

Fig. 1. System Layers

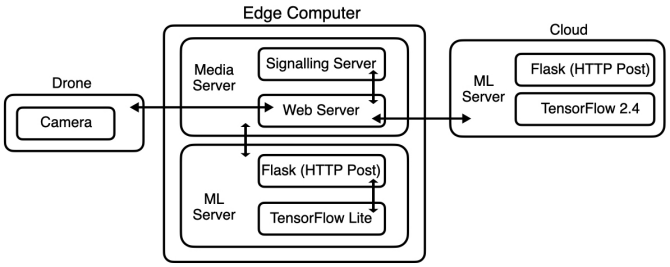

Fig. 2. Architecture of the system that enables streaming, detection and switching.

- Streaming multiple camera channels: After the user signs in as a broadcaster and establishes a session with the server, the default camera is broadcasted. In addition, we created a function that allows users to select from multiple source cameras and stream the feeds.
- Threshold: The default threshold for object detection is set at 50%. However, we have created a function that allows users to set a new threshold if they need to increase or decrease the detection limit. The new input is retrieved from the HTML and sent to the machine learning server through HTTP POST as soon as the user clicks the change button.

Machine learning server: This server uses a Flask server to get images from the media server, and it uses TensorFlow object detection API to detect objects in an image. The results are then sent back to the media server via HTTP Post. The ML server is hosted on an edge computer as well as in the cloud. It is the only server running on the cloud VM. The key difference between the edge and cloud machine learning servers is that the lite version of TensorFlow is used on the edge, but on the cloud, the full version is used (TensorFlow 2).

The following two conditions determine if the detection should happen locally (on an edge computer) or remotely (in the cloud):

- The first condition checks if the network is good or poor; if it is good (e.g., RSSI is more than -80 dBm when Wi-Fi is used), the system will do all the detection on the cloud VM. However, if the network becomes unreliable, and edge computer's battery level is more than 20%, it will immediately switch back to the edge computer.
- The second condition checks how much battery is left on the edge computer. For example, if the edge computer reaches 20% capacity (voltage drop), the object detection will run on the cloud VM to save the remaining battery for the streaming. However, if the battery is or more than 20% and the network connection is bad, all the detection will be performed locally.

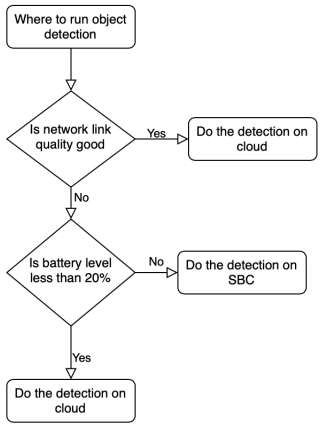

Fig. 3. Conditions to determine where object detection should occur.

After checking the conditions listed above, depending on which condition is met, one of two function will be called. For example, if the condition of cloud detection is met, the cloud function will be invoked, and this function will send the images to the cloud server through HTTP POST using the cloud server IP address. The output will also be retrieved from the cloud server using HTTP GET. If the edge computer detection condition is met, the other function which includes edge computer IP address will be called and images will be sent to edge computer.

V. Results and Discussion

A. Object detection accuracy

As pointed out above, we used Tensorflow Lite on the edge computer and TensorFlow2 API on the cloud for the object detection. The API is built on top of TensorFlow to construct, train and deploy object detection models. We selected an SSD model with Mobilenet (ssd_mobilenet_v1_coco_2017_11_17) for this work. The model is trained on MSCOCO [31] dataset, which contains 2.5 million labelled instances of 328K images of 91 object types such as a person, a laptop, cup, etc. We tested the pre-trained model without fine-tuning for both TensorFlow and TensorFlow Lite on a fraction of data and compared object detection accuracy. According to our results in Fig. 4, TensorFlow Lite is about 83% accurate as compared to TensorFlow 2, 99% accuracy on the same dataset.

B. Battery consumption

To justify our rationale behind for the proposed bi-directional switching between cloud and local processing, we measured the Raspberry Pi's battery consumption over time

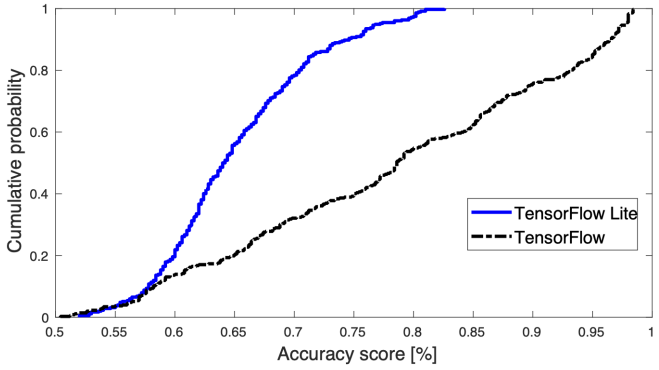

Fig. 4. Comparison of accuracy of TensorFlow and TensorFlow Lite.

for three different cases (a) when the SBC is in an idle state, (b) when streaming server and object detection run on the Pi and (c) when object detection tasks are executed on the cloud servers. Our results in Fig. 5 show that the battery lasts for about 60 minutes when the detection server runs on the cloud. However, when the streaming and detection servers are both running locally on the SBC, the battery lasts for around 40 minutes. If detection is performed on the cloud through a good wireless connection, the battery lasts for 53 minutes, providing 32.5% gain. SBC devices consume more energy when transmitting data under bad wireless connections. It would be interesting to quantify how long the battery will last under various wireless network conditions but this is beyond our objectives in this paper.

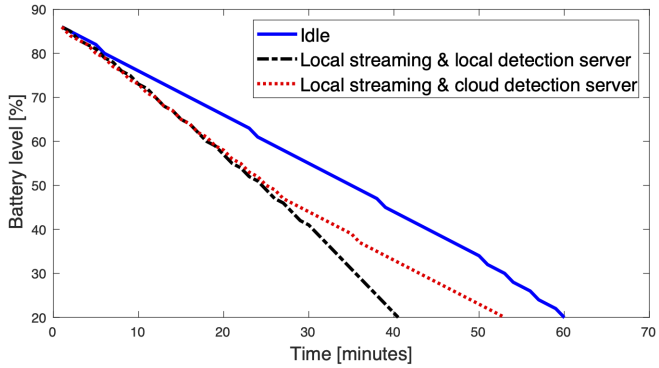

Fig. 5. Raspberry Pi's battery performance under different scenarios.

C. Detection time using 4G and Wi-Fi

To quantitatively evaluate our approach, we measure the detection time, i.e. the time between imagery data transferred to the cloud and back to the media server via both Wi-Fi (802.11ac) and 4G. Our results show (see Fig. 6) that around 99.99% of the processed frames took less than 0.5 seconds when streamed back to the client after detecting objects on the Wi-Fi network. On the other hand, 50% of the data took up to 0.5 seconds using 4G network, while the remaining data took up to 1 second. This shows that wireless network connections can play a pivotal role when transferring data to remote cloud servers for object detection. Technologies that offer higher speed and low latency, such as 5G, could provide faster detection times.

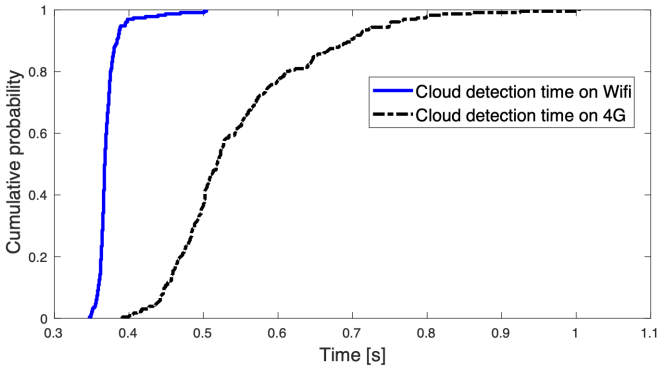

Fig. 6. Cloud detection time on 4G and Wi-Fi.

D. Switching time between edge and cloud computing

We recorded the time it took to switch object detection from the edge computer to the cloud and from the cloud to the edge computer based on the two conditions given in the previous section. Fig. 6 shows that about 70% of tests took 2 seconds to switch from the edge to the cloud computer and vice versa. However, when we look closely at the trial, we can see that switching from cloud to local is quicker. The latency in changing from cloud server processing to local is likely due to the severely limited computing resources available on the Raspberry Pi [32]. The performance can be enhanced using more powerful SBC such as Nvidia Jetson Nano.

VI. Conclusion

In this paper, we proposed real-time object detection with autonomous switching between local, on board machines and cloud computing platforms. The system is implemented and tested on state-of-the-art embedded devices such as Raspberry Pi 4. Our evaluation shows that the proposed system can increase the battery life of embedded devices by 32.5% when object detection tasks are offloaded to run on cloud instances. We also show that around 70% of our tests took only 2 seconds to switch from local to cloud and vice versa, which makes the process seamless and unnoticeable when switching occurs. Our findings also show that streaming a video and performing object detection on cloud instances does not add a significant latency to the detection time. We tested the time it takes to detect objects in a video on Wi-Fi network and 4G cellular network. Our results show that on a Wi-Fi network, with download speeds of 100 Mbps and upload speeds of 60 Mbps, more than 99% of the objects were detected in less than 0.5 seconds. However, because of the slower internet speed of 4G ($<$20 Mbps), detection took up to 1 second. The Wi-Fi results indicate that given faster cellular networking technologies such as 5G we can make the transition time significantly shorter.

This work can be extended in many directions. For example, we are planning to integrate other types of object detection

models such as YOLOv5 and other streaming protocols such as the new Secure Reliable Transport (SRT) protocol.

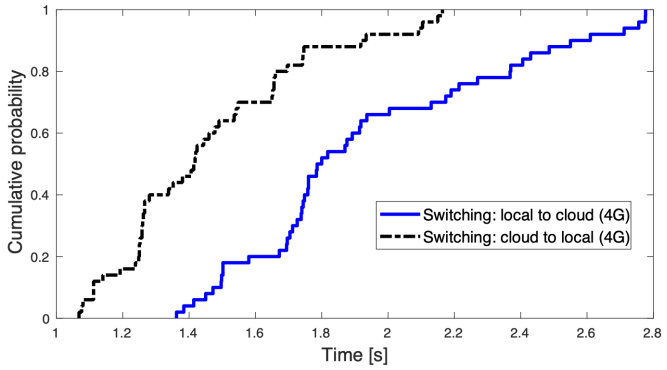

Fig. 7. Switching time between edge and cloud computing on 4G network.

ACKNOWLEDGMENT

This work was funded by the Department for Digital, Culture, Media & Sport (DCMS) as part of the 5G Connected Forest (5GCF) Project under its 5G Testbeds and Trials Program.

REFERENCES

[1] K. Lisowski and A. Czyżewski, "Pawlak's flow graph extensions for video surveillance systems," in *2015 Federated Conference on Computer Science and Information Systems (FedCSIS)*. IEEE, 2015, pp. 81–87.

[2] "Raspberry Pi 4," https://www.raspberrypi.org/products/raspberry-pi-4-model-b, accessed Apr. 20, 2021.

[3] "TensorFlow lite," https://www.tensorflow.org/lite, accessed Aug. 1, 2021.

[4] "Google Coral," https://coral.ai, accessed Aug. 14, 2021.

[5] "Neu Pro," https://www.ceva-dsp.com/product/ceva-neupro/, accessed Jul. 21, 2021.

[6] B. Kehoe, S. Patil, P. Abbeel, and K. Goldberg, "A survey of research on cloud robotics and automation," *IEEE Transactions on automation science and engineering*, vol. 12, no. 2, pp. 398–409, 2015.

[7] M. Sajjad, M. Nasir, K. Muhammad, S. Khan, Z. Jan, A. K. Sangaiah, M. Elhoseny, and S. W. Baik, "Raspberry Pi assisted face recognition framework for enhanced law-enforcement services in smart cities," *Future Generation Computer Systems*, vol. 108, pp. 995–1007, 2020.

[8] R. Mahmud and A. N. Toosi, "Con-pi: A distributed container-based edge and fog computing framework," *IEEE Internet of Things Journal*, 2021.

[9] M. Sajjad, K. Muhammad, S. W. Baik, S. Rho, Z. Jan, S.-S. Yeo, and I. Mehmood, "Mobile-cloud assisted framework for selective encryption of medical images with steganography for resource-constrained devices," *Multimedia Tools and Applications*, vol. 76, no. 3, pp. 3519–3536, 2017.

[10] K. Akherfi, M. Gerndt, and H. Harroud, "Mobile cloud computing for computation offloading: Issues and challenges," *Applied computing and informatics*, vol. 14, no. 1, pp. 1–16, 2018.

[11] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," *Advances in neural information processing systems*, vol. 28, pp. 91–99, 2015.

[12] R. Girshick, J. Donahue, T. Darrell, and J. 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.

[13] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "SSD: Single shot multibox detector," in *European conference on computer vision*. Springer, 2016, pp. 21–37.

[14] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," *arXiv preprint arXiv:2004.10934*, 2020.

[15] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[16] J. G. Apostolopoulos, W.-t. Tan, and S. J. Wee, "Video streaming: Concepts, algorithms, and systems," *HP Laboratories, report HPL-2002-260*, 2002.

[17] Y. Liu, B. Du, S. Wang, H. Yang, and X. Wang, "Design and implementation of performance testing utility for rtsp streaming media server," in *2010 First International Conference on Pervasive Computing, Signal Processing and Applications*. IEEE, 2010, pp. 193–196.

[18] X. Lei, X. Jiang, and C. Wang, "Design and implementation of streaming media processing software based on RTMP," in *2012 5th International Congress on Image and Signal Processing*. IEEE, 2012, pp. 192–196.

[19] T. C. Thang, Q.-D. Ho, J. W. Kang, and A. T. Pham, "Adaptive streaming of audiovisual content using MPEG DASH," *IEEE Transactions on Consumer Electronics*, vol. 58, no. 1, pp. 78–85, 2012.

[20] P. Chakraborty, S. Dev, and R. H. Naganur, "Dynamic http live streaming method for live feeds," in *2015 International Conference on Computational Intelligence and Communication Networks (CICN)*. IEEE, 2015, pp. 1394–1398.

[21] I. Santos-González, A. Rivero-García, J. Molina-Gil, and P. Caballero-Gil, "Implementation and analysis of real-time streaming protocols," *Sensors*, vol. 17, no. 4, p. 846, 2017.

[22] F. Xia, F. Ding, J. Li, X. Kong, L. T. Yang, and J. Ma, "Phone2Cloud: Exploiting computation offloading for energy saving on smartphones in mobile cloud computing," *Information Systems Frontiers*, vol. 16, no. 1, pp. 95–111, 2014.

[23] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "Maui: making smartphones last longer with code offload," in *Proceedings of the 8th international conference on Mobile systems, applications, and services*, 2010, pp. 49–62.

[24] "MediaDevices.getUserMedia()," https://developer.mozilla.org/en-US/docs/Web/API/MediaDevices/getUserMedia, accessed Jun. 21, 2021.

[25] "Canvas API," https://developer.mozilla.org/en-US/docs/Web/API, accessed Aug. 20, 2021.

[26] "HTMLCanvasElement.toBlob()," https://developer.mozilla.org/en-US/docs/Web/API/HTMLCanvasElement/toBlob, accessed Aug. 20, 2021.

[27] "HTTP POST," https://developer.mozilla.org/en-US/docs/Web/HTTP, accessed Aug. 17, 2021.

[28] "Is Google Tensorflow Object Detection API the easiest way to implement image recognition?" https://towardsdatascience.com/is-google-tensorflow- object-detection-api-the-easiest-way-to-implement- image-recognition-a8bd1f500ea0, accessed Aug. 28, 2021.

[29] "Detect Objects Using Your Webcam," https://tensorflow-object-detection-api-tutorial.readthedocs.io/en/latest/auto, accessed Aug. 28, 2021.

[30] "JSON," https://developer.mozilla.org/en-US/docs/Web/JavaScript, accessed Aug. 28, 2021.

[31] "COCO dataset," https://cocodataset.org/#home, accessed Aug. 19, 2021.

[32] "Network bandwidth," https://cloud.google.com/compute/docs/network-bandwidth, accessed Aug. 19, 2021.