electronics

MDPI

Article

Offload Shaping for Wearable Cognitive Assistance †

Roger Iyengar [1,*], Qifei Dong [1], Chanh Nguyen [1], Padmanabhan Pillai [2] and Mahadev Satyanarayanan [1,*]

[1] Computer Science Department, Carnegie Mellon University, Pittsburgh, PA 15213, USA; qifeid@cs.cmu.edu (Q.D.); chanh@cs.umu.se (C.N.)
[2] Intel Labs, Santa Clara, CA 95054, USA; padmanabhan.s.pillai@intel.com
* Correspondence: raiyenga@cs.cmu.edu (R.I.); satya@cs.cmu.edu (M.S.)
† This paper is an extended version of our paper published in IEEE International Conference on Edge Computing and Communications (EDGE), Chicago, IL, USA, 2–8 July 2023.

Abstract: Edge computing has much lower elasticity than cloud computing because cloudlets have much smaller physical and electrical footprints than a data center. This hurts the scalability of applications that involve low-latency edge offload. We show how this problem can be addressed by leveraging the growing sophistication and compute capability of recent wearable devices. We investigate four Wearable Cognitive Assistance applications on three wearable devices, and show that the technique of offload shaping can significantly reduce network utilization and cloudlet load without compromising accuracy or performance. Our investigation considers the offload shaping strategies of mapping processes to different computing tiers, gating, and decluttering. We find that all three strategies offer a significant bandwidth savings compared to transmitting full camera images to a cloudlet. Two out of the three devices we test are capable of running all offload shaping strategies within a reasonable latency bound.

Keywords: computer vision; machine learning; offloading; cyber foraging; wearable computing; mobile computing; edge computing; IoT; cloudlet; augmented reality

Citation: Iyengar, R.; Dong, Q.; Nguyen, C.; Pillai, P.; Satyanarayana, M. Offload Shaping for Wearable Cognitive Assistance. *Electronics* **2024**, *13*, 4083. https://doi.org/10.3390/electronics13204083

Academic Editor: Fernando De la Prieta Pintado

Received: 18 August 2024
Revised: 30 September 2024
Accepted: 1 October 2024
Published: 17 October 2024

1. Introduction

Offloading compute-intensive operations at low latency from underpowered wearable devices over a wireless network to a nearby cloudlet was one of the original motivations for edge computing [1]. Today, it continues to be an important driver of edge computing, but faces the challenge of limited elasticity. A cloudlet is designed for a much smaller physical and electrical footprint than a cloud data center. Hence, modest load spikes can overwhelm a cloudlet and its wireless network. Since low end-to-end latency is non-negotiable for many edge-native applications [2], shifting load to the cloud is not feasible. Techniques that reduce the average utilization of shared resources for edge offload are therefore valuable.

These techniques, known as offload shaping, were proposed by Hu et al. [3] in 2015. That work presented empirical evidence that many instances of offloading are wasted work because of imperfect real-time sensing (e.g., blurry image capture or near-duplicate frames). Offload shaping eliminates resource consumption by useless work via early discard in the processing pipeline that starts at the wearable device. Hu et al. showed that offload shaping was possible, even with the limited capability of wearable devices, and it significantly reduced the utilization of shared resources.

We extend the concept of offload shaping by leveraging the growing compute capability of recent wearable devices. We observe that on-device hardware accelerators for tasks such as deep learning inference [4], super-resolution [5], and scene analysis [6] have emerged. The Magic Leap 2 is advertised as having a "14 core computer vision processing engine". Google states that its Glass® Enterprise Edition 2 uses a Qualcomm® Snapdragon™ XR1, which includes an artificial intelligence engine (Qualcomm is based

in Santa Clara, CA, USA). At the same time, we observe that the complexity and resource demand of the end-to-end processing pipelines have also grown (e.g., because of larger deep neural networks (DNNs)).

We explore offload shaping in the context of *Wearable Cognitive Assistance* (*WCA*) applications for assembly tasks. Originally described in 2014 [7], this genre of applications has emerged as a "killer app" for edge computing because (a) they transmit large volumes of video data from device to cloudlet; (b) they have stringent end-to-end latency requirements; and (c) they make substantial compute demands of the cloudlet, often requiring a GPU. A WCA application runs on a wearable device such as Google Glass® or Microsoft® Hololens®, leaving the user's hands free for task performance (Google is based in Mountain View, CA, USA. Microsoft is based in Redmond, WA, USA). It provides visual and verbal guidance and error detection for a user who is performing an unfamiliar task. We investigate four WCA applications on three wearable devices: Google Glass® Enterprise Edition 2, Vuzix Blade® 2, and Magic Leap 2 (Vuzix is based in Rochester, NY, USA. Magic Leap is based in Plantation, FL, USA).

The main contribution of this work is a study of the different aspects and metrics of offload shaping. First, we show that pipelines of DNNs, rather than standalone DNNs, are required for accurate computer vision for WCA. Next, we examine three techniques for offload shaping; split computing, gating, and decluttering. For each, we measure the change in inference time, accuracy, network utilization, and power consumption across a diverse set of wearable devices. Some of these metrics show clear wins for offload shaping, while others do not. The savings that are achievable varies across devices, but the concept of offload shaping is robust. We argue that offload shaping is thus a valuable technique for reconciling the conflicting demands of scalability and end-to-end performance for WCA tasks.

The rest of the paper is organized as follows. Section 2 provides the Background and Related Work. Section 3 describes the four WCA applications studied in this work, and the computer vision pipelines associated with them. Sections 4–6 describe the three strategies for offload shaping we examined, and they present the main experimental results of this paper. Section 7 concludes the paper.

An earlier version of this work was published in a conference paper [8] that lacked much of the data analysis presented here. The conference paper also did not contain any of the work on offload shaping or decluttering.

2. Background and Related Work

2.1. Computing Tiers

The modern computing landscape has been described as a tiered model [9]. Tier-1 is the cloud, which consolidates large numbers of servers into single data centers. The cloud benefits from economies of scale. Electrical infrastructure, cooling facilities, and maintenance staff are leveraged efficiently because there are many servers in a data center. In addition, individual applications can scale to run on different numbers of servers as demand for these applications changes. Data centers also offer large amounts of storage.

Cloud data centers take up a lot of space, and they are often located far away from sensors, mobile devices, and end users. Tier-2 servers, also called cloudlets, are distributed so that there is likely to be a tier-2 device in close network proximity to end users and sensors that are producing large volumes of data. The tier-2 devices are less powerful than the devices in tier-1. The lack of consolidation makes it more expensive to maintain, cool, and power tier-2 devices. However, the close network proximity enables low latency and high bandwidth network connections between tier-2 devices, sensors, and end users.

Tier-3 devices include mobile phones, smart glasses, and internet-connected cameras. These devices are typically capable of modest computations, but they are often limited by size, weight, and power constraints. Tier-3 devices often have to transmit data to tier-1 or tier-2, and computationally intensive operations must be run there.

2.2. Partitioning Offloaded Computation

How to partition a processing pipeline so that its lightweight head executes on a mobile device, and its heavyweight tail executes in the cloud or on an edge computing node (cloudlet), is a hot topic today [10–16]. The partitioning is called split computing (SC) in the current literature.

The essence of SC, partitioning a processing pipeline, has deep roots that reach back to the very first use of offloading to amplify the capabilities of a mobile device [17]. This work described how a speech recognition application was modified to operate in one of three modes. In one of the modes, a prelimary phase of speech processing was done locally (i.e., the "head"), and the extracted information was shipped to a remote server for the completion of the recognition process (i.e., the "tail"). For certain combinations of network bandwidth and device/server capabilities, this split offered lower end-to-end latency than fully local or fully remote execution.

In 1999, Flinn et al. [18] showed how SC could extend battery life. Abstracting and generalizing from these efforts, the concept of "cyber foraging" was introduced in 2001 [19]. Today, we use the term "offloading" instead of "cyber foraging". Building on this foundational work, the period from 2001–2008 saw vigorous research activity [20–28]. Flinn's survey [29] provides a detailed account of these efforts. Between 2009 and 2015, MAUI [30], Odessa [31], CloneCloud [32], and COMET [33] explored programming language and virtual machine support for offloading.

In 2015, Hu et al. [3] introduced offload shaping:

> "...we show that it is sometimes valuable to perform additional cheap computation, not part of the original pipeline, on the mobile device in order to modify the offloading workload. We call this offload shaping. We show that offload shaping can be applied at many different levels of abstraction using a variety of techniques, and that it can produce significant reduction in resource demand with little loss of application-level fidelity or responsiveness".

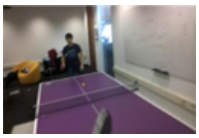
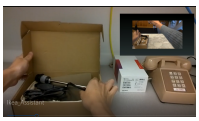
The "cheap computation" mentioned above can take many forms, including blur detection using vision algorithms [34,35], IMU-based blur detection, and perceptual hashing [36–38] to detect nearly similar frames for deduplication. Although the head algorithms explored by Hu et al. did not include DNNs, the 2018 extensions of this concept by Wang et al. [39] used DNNs at the head to implement the early discard of video frames. The head DNN is cheap but not very accurate; the DNN at the cloudlet (tail) is far more accurate. Together, the cascaded pair of DNNs achieve good precision and recall.

The DNN-centric focus of SC for mobile devices began in 2017, with Kang et al. [40]. Couper, a 2019 system by Hsu et al [12], showed how production DNNs could be optimally sliced for SC in different hardware settings. MARVEL [41] is a latency-sensitive mobile augmented reality (MAR) system that computes optical flow on a mobile device and offloads more computationally intensive tasks to the cloud. Liu et al. [42] developed a system that runs object detection on a server, but runs tracking and rendering on the mobile device to compensate for the latency added by network communications. A 2022 survey [16] highlights recent work in this space since 2019.

2.3. Wearable Cognitive Assistance

Wearable Cognitive Assistance (WCA) refers to a class of cyber-human systems that have the "look and feel" of AR, but use compute-intensive AI algorithms such as computer vision based on machine learning (ML) in their time-critical execution paths. An application of this genre combines a wearable device such as Microsoft Hololens or Google Glass (tier-3) with edge computing (tier-2) to offer real-time task-specific guidance. The tasks can span a wide range, from assistive guidance for a visually-impaired user to the assembly or repair of a complex mechanical artifact. The role of the AI algorithms is to track progress through the task, verify the correct execution of steps, and offer the timely detection and correction of errors as they occur [7,43,44]. Table 1 shows three example WCA applications.

Table 1. Examples of WCA applications. The Lamp is closest to the applications considered in this paper.

App	Example Input	Description
Face		Whispers name of a familiar person whose name you cannot recall. Detects and extracts a tightly-cropped image of face, and then applies a face recognizer.
Pingpong		Tells novice to hit ball to the left or right, depending on which is more likely to beat opponent.
Lamp		Guides a user in assembling a table lamp from an IKEA kit. Gives video and verbal guidance for next step.

2.4. WCA Latency Bounds

Chen et al. [43] conducted user studies to determine acceptable latency bounds for WCA applications. They found tight and loose latency bounds, which they describe as follows:

> "The tight bound represents an ideal target, below which the user is insensitive to improvements, as measured, for example, by impact on performance or ratings of satisfaction. Above the loose bound, the user becomes aware of slowness, and user experience and performance is significantly impacted".

The tight latency bound for an assembly application was 600 ms, and the loose bound was 2700 ms.

3. Image Processing in WCA

We developed four WCA applications, that each assist users with a different assembly task. Table 2 lists these application and describes the tasks they assist with. Each application determines progress on the task using DNN computer vision models, fine-tuned for the specific task. We collect separate training images for each application, that depict every step of the assembly task. We label each image to indicate the step of the task that is displayed, and draw a bounding box around the section of the image that contains the object being assembled. We also collect and label separate sets of test data to evaluate the accuracy of the DNN models.

Table 2. The four WCA applications we developed.

Name	Description
Stirling	Assemble a heat engine from metal parts
Meccano	Build a model bike from metal parts
Toyplane	Build a model helicopter from 3D printed plastic parts
Sanitizer	Assemble a sanitizer for a smartphone from metal and plastic parts

For each task, we had one class for each step of the task. A class in our dataset represents a step of the assembly process, rather than a different type of object.

DNNs with low accuracy will result in a poor user experience. When a frame is misclassified, an application either fails to recognize that the user has successfully completed a step, or it detects that a step has been completed when it has not and gives the user a new instruction prematurely.

3.1. Standalone DNN

We train and test standalone DNNs on the data that we collected for each application. These include both image classifiers and object detectors. Image classifiers are given an image, and we assign a label indicating the type of object that is shown in the whole image. When training our image classifiers, we ignore bounding box labels and just train the models using class labels. The image classifiers we tried were Resnet 50 [45] and Fast MPN-COV [46]. Object detectors can find multiple objects present in an image, as opposed to just one. They return bounding box coordinates and class labels for each object present in an image. We evaluate the Faster R-CNN [47] and EfficientDet [48] object detectors. Our evaluation looks at three different versions of EfficientDet. EfficientDet-Lite0 has the smallest number of learned parameters, and EfficientDet-Lite2 has the largest number of learned parameters. Table 3 lists the two image classifiers and the two object detectors that we tested.

Table 3. The DNN models used and the types of devices they can be run on.

Name	Type	Supported Hardware Type
Resnet 50	Image Classifier	Server and Mobile
Fast MPN-COV	Image Classifier	Server
EfficientDet	Object Detector	Server and Mobile
Faster R-CNN	Object Detector	Server

We train and test DNNs on data specific to each application. Each image contains exactly one instance of the object being assembled. We compute Top-1 accuracy for the image classifiers by comparing the highest confidence label from the model with the ground truth label assigned for the image. Top-1 accuracy is computed for the object detector by comparing the class label of the object that the model detected with the highest confidence score and the ground truth class label. The bounding box coordinates returned by the object detector are ignored, because the applications do not need to know the location of the object being assembled. They just need to know the step of the assembly task that is shown in an image. Figure 1 shows the accuracy of standalone DNN models for all four applications. For each application, a single accuracy score is computed across all test data. The test data for a given application will include images of every step of the task. All models achieve under 80% for three out of four WCA applications. This suggests that a standalone DNN is insufficient.

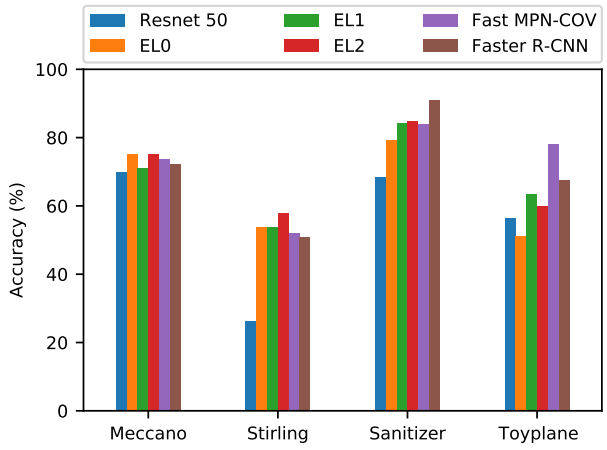


Figure 1. Top-1 classification accuracy for standalone DNN models. This is the percentage of images that the model classified correctly. EL means EfficientDet-Lite.

3.2. Pipeline

In an attempt to increase accuracy, we use a two stage process inspired by Gebru et al [49]. An object detector first finds the region of an image that contains the section of the object that the user is currently assembling. The application then crops the image around this region, and then determines the step of the task that is shown, using an image classifier. As with the standalone DNN implementations of our applications, the image classifier has one class for each step of the task. This process is depicted in Figure 2.

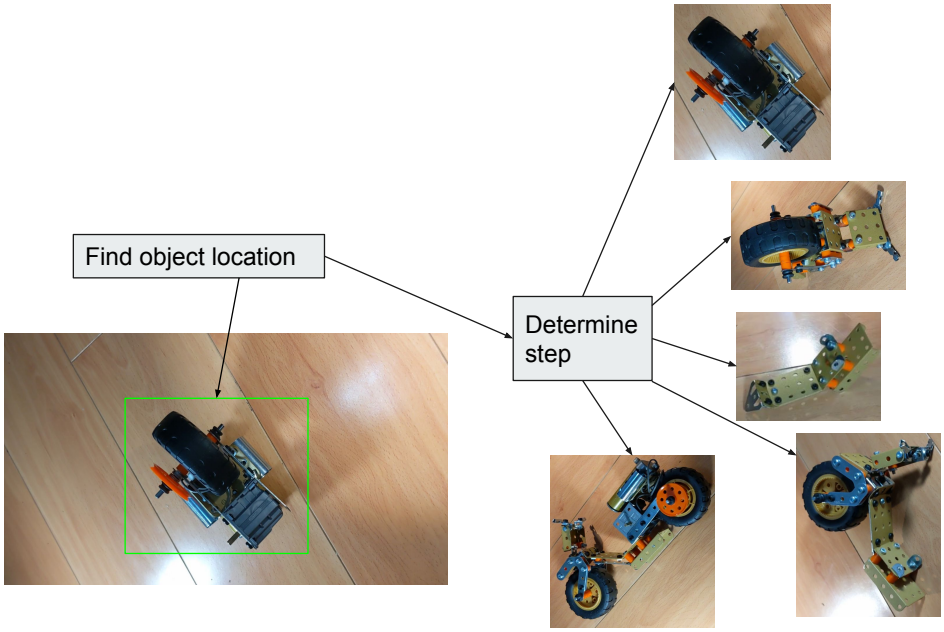


Figure 2. The two stage process our applications use to determine the step of an assembly task that is shown in an image.

In order to train the object detectors to find the region of an image containing the assembly object, without identifying the step of the task that is being displayed, we modified our labels such that every bounding box gets assigned to the same class. Rather than determining bounding box coordinates, and the step of the task, the object detectors in our pipeline found bounding box coordinates and assigned every bounding box to the same class label.

The image classifiers in our pipeline were trained on cropped images with class labels. We generated training data for our image classifiers by cropping the original images in our dataset using the labeled bounding boxes. Only the part of the image inside the bounding box remained after this process. However, the original class labels for each step of the task were left alone. They were not merged into a single class, as they were in the training set for the object detectors. This allows the image classifiers to classify cropped images, rather than the original full images that also contained part of the empty table around the object that was being assembled.

In this way, we were able to train object detectors that found the region of an image containing an assembly object, and image classifiers that determined the step of an assembly task that is shown in a crop. We trained the object detectors Faster R-CNN [47] and EfficientDet [48] and the image classifiers Resnet 50 [45] and Fast MPN-COV [46].

We test the pipelines of models trained for each application on the test set for that application; the results are shown in Figure 3. The pipeline consisting of Faster R-CNN and Fast MPN-COV achieves the highest accuracy for all applications except Stirling. The best pipeline outperforms the best standalone DNN for all four applications.

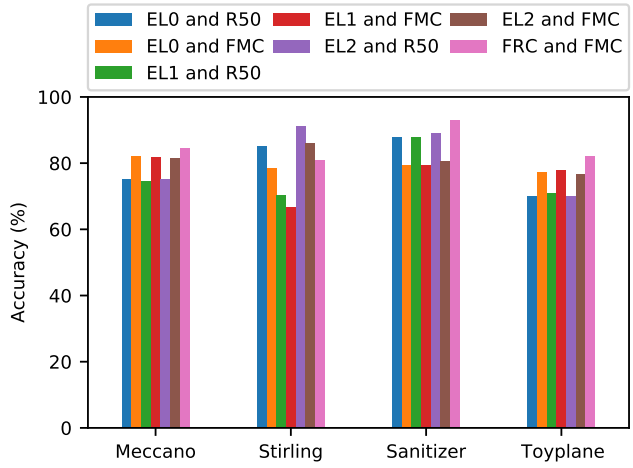


Figure 3. Top-1 classification accuracy for pipelines. EL means EfficientDet-Lite. R means Resnet. FMC means Fast MPN-COV. FRC means Faster R-CNN.

4. Mapping Processing to Computing Tiers

All of the models and pipelines described in Section 3 can be run on a cloudlet, with the applications implemented as thin clients. However, cloudlets are a limited resource [50]. Heavy cloudlet load limits the number of other users that can share the cloudlet. We thus examined how wearable devices can reduce the amount of processing on cloudlets. Network bandwidth is also a shared limited resource. Our experiments examine how running some computations locally, instead of offloading them to a cloudlet, can reduce the amount of bandwidth used by WCA applications. Figure 4 shows the three partitioning strategies we explore.

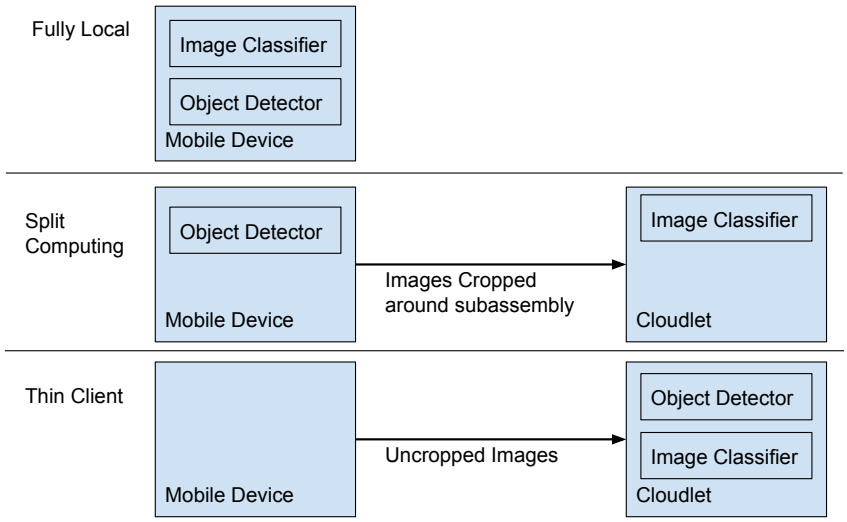


Figure 4. Implementation options for WCA.

4.1. Fully Local Computation

Carrying out all computations locally avoids consuming any cloudlet resources or network bandwidth. Unfortunately, fully local computation limits an application's image processing capabilities to what the wearable devices are capable of running. In addition, the battery in the devices must power all of these computations. We run experiments to find the accuracy, inference time, and power consumption for models running directly on wearable devices. This indicates whether or not it is practical to run all computations for WCA applications locally on the wearable device.

EfficientDet and Resnet 50 can be run on Android devices using PyTorch Mobile and TensorFlow Lite. However, Fast MPN-COV and Faster R-CNN cannot currently run on mobile devices [51,52]. All of our pipelines that run locally on the wearable devices were thus limited to using EfficientDet and Resnet 50.

4.2. Thin Client

Offloading all computations to a cloudlet allows us to process images using Faster R-CNN and Fast MPN-COV, which enables us to use the most accurate pipelines graphed in Figure 3. We measured the inference time and power consumption of a thin client that offloads all computations to a cloudlet. We implemented a flow control mechanism in the thin client that only allows the client to send one image to the cloudlet at a time. After the client sends an image to the cloudlet, the client waits to receive the result of processing that image before the client sends the next image. This prevents a buffer of stale frames from building up on the cloudlet while the cloudlet is busy processing a frame. Figure 5 shows this topology.

In a real application capturing live data, frames captured by the camera, while the server is busy processing data, will be dropped. This approach is used by Ha et al. [7]. However, our applications process existing sets of images, so they do not drop any frames. They simply wait until the server finishes processing a frame and then transmit the next one.

The wearable devices were Internet-connected via Wi-Fi, while the cloudlet was Internet-connected via 1 Gbps Ethernet. Figure 5 shows this topology. The ping time between the wearable device and the cloudlet was under 5 ms. The cloudlet had an Intel® Xeon® Processor E5-2699 CPU and an Nvidia® GeForce® GTX 1080 Ti GPU (Both Intel and Nvidia are based in Santa Clara, CA, USA).

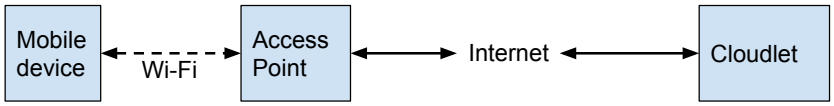


Figure 5. The network connecting wearable devices to the cloudlet. Solid lines represent a wired connection. The wearable device and cloudlet are in close network proximity to each other.

The thin client experiments represent the best case scenario: a user with a high bandwidth and low latency connection to a server with a GPU. The wearable device can offload all expensive computations to this server. This saves power on the wearable device, and supports compute-intensive DNNs. Our thin client experiments achieve the highest accuracy and the lowest possible execution time and device power consumption.

4.3. Split Computing

Split computing is a form of offload shaping to reduce network bandwidth and/or cloudlet resource usage. Preprocessing on the wearable device can reduce the amount of data that must be sent to the cloudlet, and/or replace some work that would have been done on the cloudlet.

As we showed in Section 3, a standalone object detector or image classifier is not sufficient for our applications. In addition, implementing the pipeline described in Section 3.2 using an object detector that is split across a client and a cloudlet is impractical. A split object detector has a head DNN which outputs an embedding that is sent to the cloudlet. The cloudlet feeds this embedding to the split object detector's tail, and the tail's output just contains the bounding box coordinates and class labels for the detected objects. There is no way for the cloudlet to obtain a cropped image from the original embedding that was sent to the cloudlet. Our application would either have to send the entire image to the cloudlet along with the embedding, or it would have to send the bounding box coordinates back to the wearable device and have the wearable device send the cropped image in some form to run the classifier on the cloudlet. The former approach eliminates all of the bandwidth

savings that split computing offers, while the latter approach requires a second round trip to the wearable device, which increases latency.

One possibility is running an object detector on the wearable device, cropping the image there, and then feeding the cropped image to a split image classifier. However, this requires running both the object detector and the head of the split image classifier on the wearable device. We instead opt to run just the object detector on the wearable device, transmit the cropped image to the cloudlet, and run the full image classifier there. This allows us to use the Fast MPN-COV image classifier, which cannot be run on a wearable device. In addition, it saves the wearable device from having to run the head of a split image classifier. This relatively simple implementation of split computing does not require ML expertise in DNN splitting. Can such a simple implementation still offer a significant bandwidth savings without unreasonably harming battery life or classification accuracy?

4.4. Results

Classification accuracy, inference time, and power consumption all impact a user's experience with a WCA application. A model with low accuracy might result in the application failing to recognize a completed step, or prematurely giving the user a new instruction. High inference time results in a large delay between a user completing a step and the application providing the next instruction. High power consumption will drain the wearable device's battery quickly. The experiments in this section measure these quantities for fully local execution, thin clients, and split computing.

4.4.1. Classification Accuracy

Figure 6 presents the highest accuracy possible for each implementation type, based on the data from Sections 3 and 4.4.3. The best performance for three out of our four applications is achieved by the thin client, which requires offloading all computations to a cloudlet.

4.4.2. Bandwidth Savings

The bandwidth savings percentages we achieved by transmitting cropped images instead of uncropped images are presented in Figure 7. These values were calculated using the formula in Figure 8.

The bandwidth savings achieved is content-dependent. The distance between the camera and the object being assembled will change how large an object appears in the image, and this will directly impact the number of bytes required to transmit the cropped image. The bandwidth savings of techniques such as image compression or DNN-based split computing vary less based on the specific content in an image.

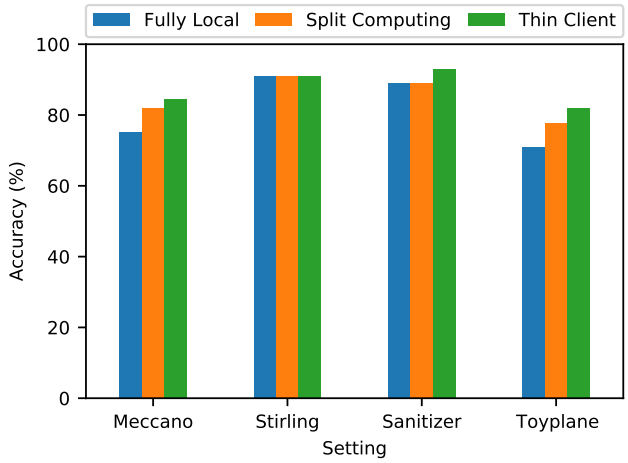


Figure 6. Classification accuracy for the best performing pipeline in each setting.

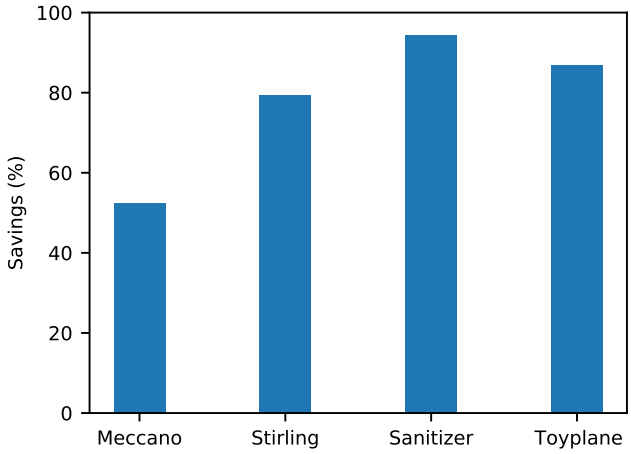


Figure 7. The bandwidth saved by transmitting cropped images compared to sending uncropped images. Images were cropped around the bounding boxes returned by EfficientDet-Lite0. Higher values represent more bandwidth savings.

$$\frac{\text{Bytes for full images} - \text{Bytes for cropped images}}{\text{Bytes for full images}}$$

Figure 8. The formula for bandwidth savings. Higher values represent more bandwidth savings.

Transmitting cropped images requires less than 50% of the bandwidth that the uncropped images require, for all of our datasets. The savings are over 90% for the Sanitizer dataset. In many cases, these significant bandwidth savings are worth the reduction in accuracy that comes along with using split computing instead of offloading all computations.

4.4.3. Inference Time

We measured image processing times for pipelines running directly on the wearable devices. We carried this out by storing our test set on the devices and running code that looped through each image. Inside the loop, our code ran the pipeline of models that was being timed. The code logged the elapsed time every 20 frames, based on Android's uptime counter. The elapsed times were divided by 20, to obtain the per-frame inference time. Each pipeline was run for five minutes. Figure 9 shows our results.

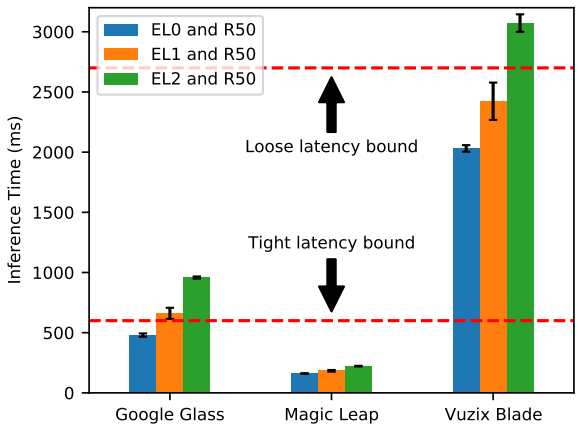


Figure 9. Single-frame inference time for fully local computation. EL means EfficientDet-lite. R50 means Resnet 50. The dashed lines represent the latency bounds from Section 2.4.

Table 4 lists the largest pipeline that meets the latency bounds from Chen et al. [43] on each device. The accuracies of each pipeline, for all applications, are graphed in

Figure 3. There is a large gap between the accuracy of the pipeline that can meet the latency bounds and the accuracy of the most accurate pipeline for most device and application combinations. This indicates that fully local computation is not an acceptable strategy in most of our cases.

Table 4. The largest pipeline that meets tight and loose latency bounds. "Largest" refers to the number of parameters used for the pipeline's version of EfficientDet.

	Tight Bound	**Loose Bound**
Google Glass®	EfficientDet-Lite0 and Resnet 50	EfficientDet-Lite2 and Resnet 50
Magic Leap	EfficientDet-Lite2 and Resnet 50	EfficientDet-Lite2 and Resnet 50
Vuzix Blade®	None	EfficientDet-Lite1 and Resnet 50

Inference time measurements for thin clients are shown in Figure 10. These measurements include the time to transmit images to the cloudlet, process them there, and then transmit the results back to the wearable device. As with our other time measurements, the applications were run for five minutes, and elapsed time was recorded every 20 frames. These values were well below the tight latency bounds on all three devices. The thin client offers the highest possible accuracy and the lowest inference time. However, it consumes the largest amount of bandwidth and cloudlet resources.

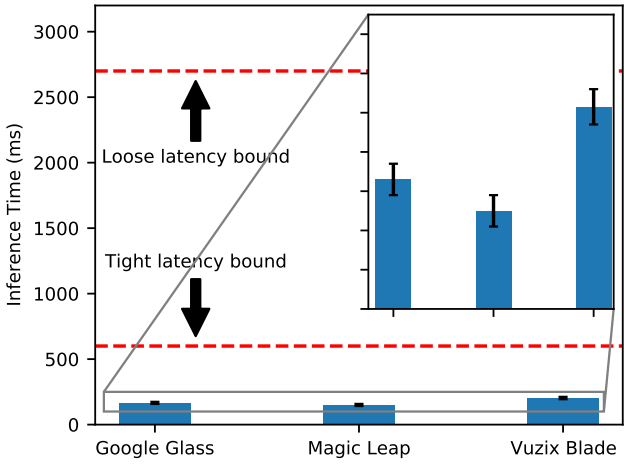


Figure 10. Single-frame inference time for the thin client on each device.

Figure 11 graphs the per-frame inference times of our split computing pipelines, across all three devices. As with accuracy, the inference time for split computing is in between fully local computations and the thin clients. The pipeline that uses EfficientDet-Lite0 runs on the Google Glass within the tight latency bound from Chen et al. [43]. The pipeline that uses EfficientDet-Lite1 is almost under the tight latency bound when run on Google Glass. All three pipelines run within the tight latency bound on Magic Leap. However, none of the pipelines run within the latency bound on Vuzix Blade.

4.4.4. Power Consumption

We measure how much power each device uses while running the pipelines fully locally in a loop. As with our previous experiments, we run each pipeline for five minutes. None of these devices have user serviceable batteries, so we cannot measure power consumption based on the current and voltage supplied to the device by its charger. Instead, we run our code with the devices unplugged, and query for current and voltage readings from Android, using the BatteryManager class. We multiply the voltage and current to compute power. Our code contains a background thread which logs the current and voltage every 100 ms.

Figure 12 shows the power values for each pipeline, running on all three devices. The baseline measurements were recorded for an application that shows empty Android activity, but do not carry anything out aside from recording current and voltage values in a background thread.

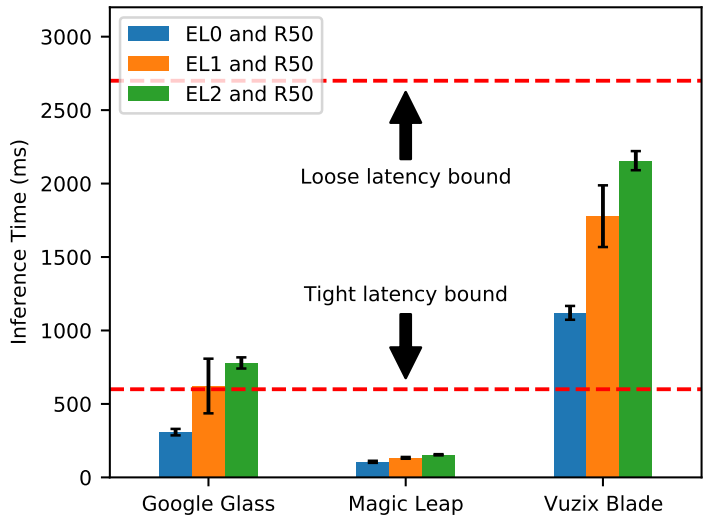


Figure 11. Single-frame inference time for split computing pipelines.

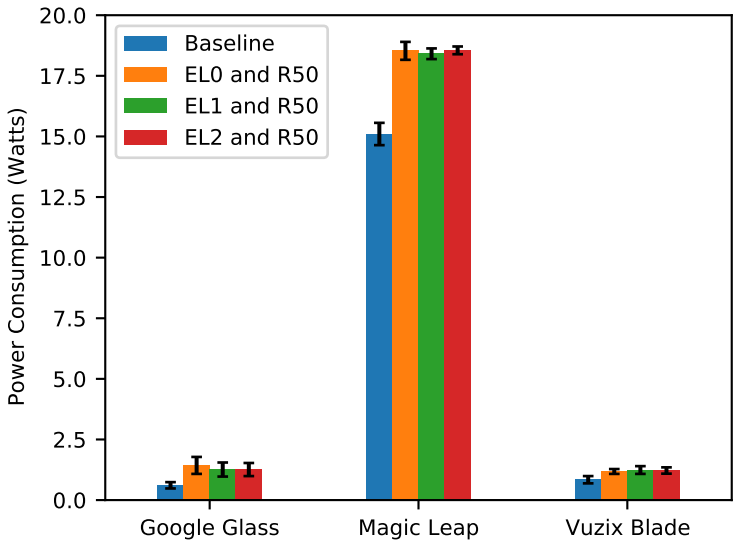


Figure 12. Average power consumption for fully local execution. These measurements are recorded while the wearable device is running the full pipeline. The baseline application does not carry out any computation.

The Magic Leap 2 consumes over 15 watts running the baseline application. The device's depth sensors or spatial mapping code might have been responsible for a lot of this. None of the pipelines increase the Magic Leap 2's power usage by more than 25% of the power consumed by the baseline. The most dramatic increase over baseline power usage is for EfficientDet-Lite0 and Resnet 50 on Google Glass, with an average power usage of 1.43 Watts. However, this still implies a reasonable battery life. A 3.2 Wh battery can supply 1.43 Watts for over two hours.

The power consumption experiments for thin clients measure the power consumed on the wearable device, but they do not measure the power consumed on the cloudlet. These results are presented in Figure 13. Running the thin client on the Vuzix Blade 2 consumes

more power than running the baseline application, but less power than running any of the on-device pipelines. The Google Glass consumes slightly more power running the thin client as it does when running the on-device pipelines. However, all three of these clients consume significantly more power than the baseline. The thin client on the Magic Leap consumes slightly less power than the baseline application. There is not a clear explanation for this, but the difference is fairly small.

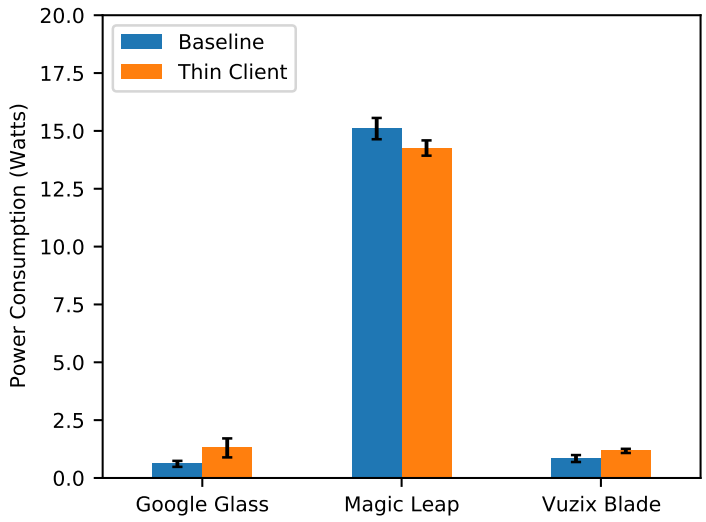


Figure 13. Average power consumption for wearable devices running thin clients. The thin clients were only run with a server that processed images using Faster R-CNN and Fast MPN-COV. Thus, this figure does not contain multiple measurements for different DNNs.

The split computing power consumption measurements are shown in Figure 14. As with our other power measurements, these measurements were made on the wearable devices and do not include the power consumed by the cloudlet. These power measurements are similar to our measurements for fully local execution from Figure 12. The increase in power consumption, above the baseline, is reasonable for all of the three devices running all of the pipelines we tested.

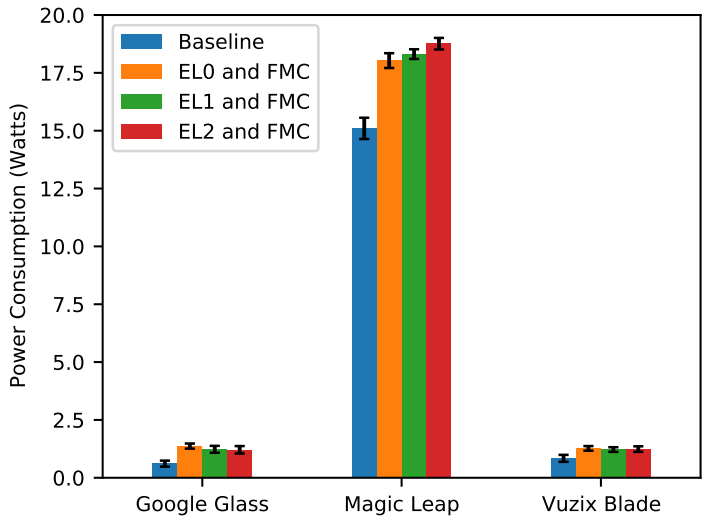


Figure 14. Average power consumption of wearable devices running split computing pipelines. An EfficientDet-lite object detector is run on the wearable device. Then, a cropped image is sent to a cloudlet, which runs a Fast MPN-COV fine-grained images classifier.

5. Gating

While running a WCA application, most of a user's time is spent completing assembly steps. Applications only need to check if a step has been completed after a user thinks the step is done. A step cannot possibly be complete while a user is in the middle of working on it. This section considers modifying applications so that users have a way to indicate when they believe they have completed a step. This prevents the applications from having to process any images in between when an instruction is given, and when the user indicates that they believe that a step is complete. We will henceforth refer to this strategy as gating. During the periods that applications do not have to process frames, they are not consuming cloudlet resources or network bandwidth. Gating is thus a form of offload shaping.

When a user indicates they believe a step is complete, the application begins processing camera images to verify if this is true. If the application determines that the step is in fact complete, it will give the user the next instruction and then stop processing frames until the user indicates that the next step has also been completed. If the application determines that a user was mistaken, and a step has not actually been completed, it will alert the user of this discrepancy.

The simplest form of gating requires the user to press a button on the wearable device to indicate that they think a step has been completed. This is trivial to implement. However, it requires the user to move one hand all the way from the object they are assembling to the side of their wearable device. An alternative form of gating we implement uses MediaPipe [53] to determine when a user shows a thumbs up gesture to the camera. The thumbs up gesture is the user's way of indicating that they think a step has been completed. Our last form of gating uses automated speech recognition. The user speaks the words "ready for detection", when they believe that a step is complete. This does not require the user to move their hands away from the object that they are assembling. But it is unlikely to work well in a noisy environment.

5.1. Experiments

A practical gating method will not significantly increase the amount of time it takes for a user to complete a task or the amount of power that a wearable device consumes. In addition, a good gating method will reduce network bandwidth substantially. We therefore measure the power consumption, task completion time, and bandwidth usage.

We implement four versions of our Toyplane application. The baseline version does not use gating. The button version has the user press a button to suggest step completion. The thumbs up version has the user make a thumbs up gesture. The speech version has the user say "ready for detection".

We implement speech gating using the PocketSphinx continuous speech recognition engine [54]. As the Sphinx developers note, their engine does not use state of the art methods for speech recognition. However, Android does not natively include a continuous speech recognition engine that third party developers can access. The Azure Cognitive Services Speech container has to be run on a server [55]. We note that there may exist a better continuous speech recognition engine that can be run entirely on an Android-based wearable device.

Three users assemble the toy plane using all four implementations of the Toyplane application. All implementations are run using a Vuzix Blade 2 headset. The headset records traces of the user completing the task with each of the applications. We then play back these traces on all three headsets to measure power consumption, task completion time, and bandwidth usage. Our playback application processes frames at the rate they were recorded at. For example, if another wearable device can process a frame with MediaPipe faster than the Vuzix Blade did when the trace was recorded, the playback app will pause until it reaches the timestamp when the frame had been processed in the original trace. This allows us to play back traces in a reproducible way on the Vuzix Blade itself, as well as other headsets that have faster hardware. Computations for gating (such as

detecting a thumbs up gesture or recognizing speech) were run on the wearable device. Determining the task step shown in an image was done on the cloudlet.

5.2. Results

Figure 15 presents average power consumption. None of the gating strategies result in significantly more power being consumed. The button gating consumed less power than the baseline on all devices. This was likely a result of the large amount of data that was sent between the cloudlet and the wearable device while the baseline application was running. The button-based gating application does not require any expensive computations to be run on the wearable device.

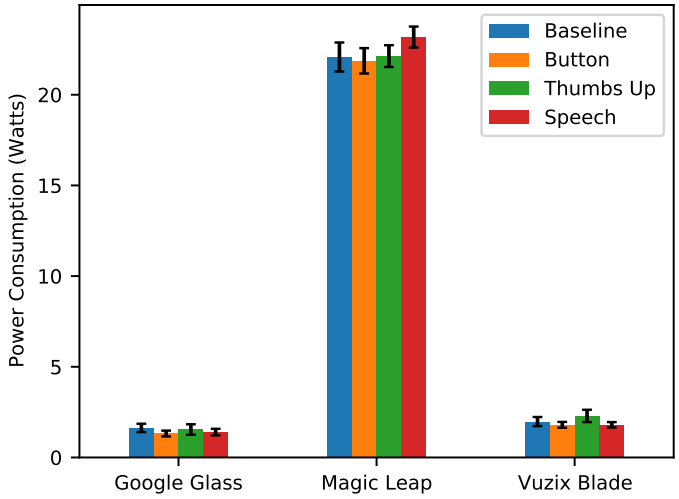


Figure 15. Average power consumption of wearable devices using different gating options.

Figure 16 shows the average task completion time for each gating strategy. The button gating adds almost no time to the task. The thumbs up and speech gating lead to a noticeable increases in average completion time. Thumbs up gating increased average completion time by over 40%.

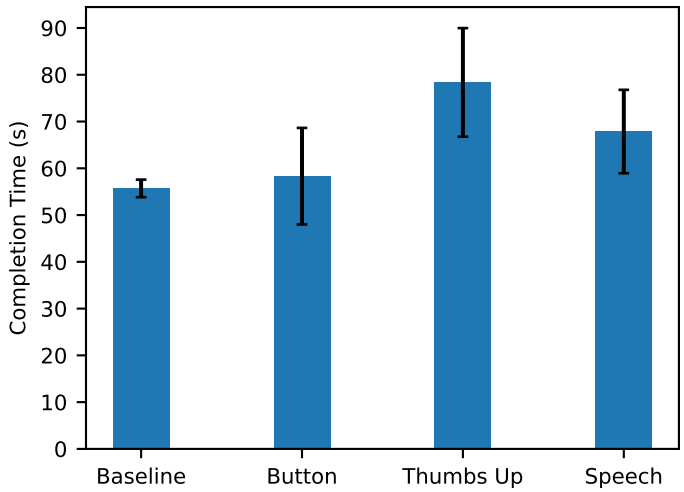


Figure 16. Average task completion time for each gating strategy.

Figure 17 graphs average bandwidth savings across our three traces for each gating strategy. All three gating strategies significantly reduce the bandwidth usage.

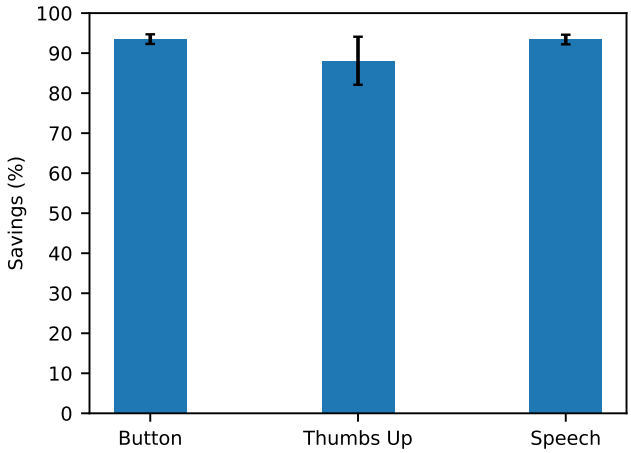


Figure 17. Average bandwidth savings over the baseline for each gating strategy. We computed the average bandwidth consumed for each gating strategy across all three traces. We next computed the average bandwidth consumed in the baseline traces, in which no gating was used. We then computed the savings percentages by plugging these values into the formula in Figure 8. Higher values represent more bandwidth savings.

5.3. Discussion

The number of users in our study is extremely limited. Although our results are encouraging, a larger user study must be run to evaluate gating as a strategy for offload shaping. In particular, the latency bounds from Chen et al. [43] might not apply. Having to press a button, make a thumbs up gesture, or speak a phrase might change a user's tolerance for processing delays. A user study examining how cumbersome users find different forms of gating would also be of interest. Some users might find it annoying to have to push a button or say a phrase.

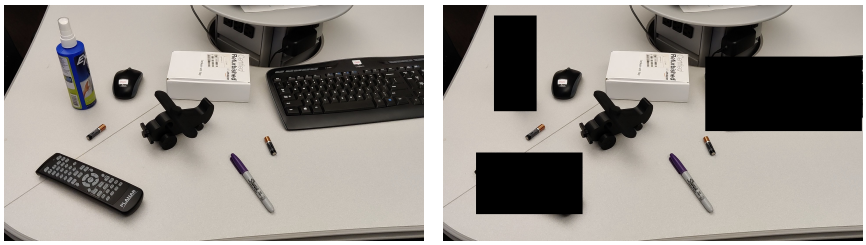
Two of our three users had difficulty getting the application to recognize the phrase "ready for detection", unless they spoke in a particularly deep voice. A more advanced speech recognition engine might not have this problem.

6. Decluttering

Decluttering is an offload shaping technique that finds and removes objects from the camera image that are not part of the item being assembled. A black box is drawn over these objects, and then the image is compressed by a JPEG encoder. For a given resolution, a JPEG encoding of an image with parts blacked out will consume fewer bytes than an encoding of the original image without parts removed.

The applications detect known objects to filter by running an EfficientDet-Lite0 object detector [48] that has been trained on the COCO dataset [56]. Our assembly tasks do not involve any objects that are classes in COCO, so we are able to simply fill in the bounding boxes with black pixels for all objects that were detected with high confidence. Figure 18b shows an example of this. After this step, images are compressed using a JPEG encoder, and then sent to the cloudlet. The cloudlet then runs the Faster R-CNN and Fast MPN-COV pipeline introduced in Section 3.2.

Our original test sets were uncluttered, so we collect new test sets that contain the object being assembled along with other objects from the COCO dataset in the background of the scene. The bandwidth saved using decluttering for this test data is presented in Figure 19. Decluttering saves over 10% of bandwidth for all four applications, and the savings on the Sanitizer data was almost 20%. As with split computing, the bandwidth saved by decluttering is content dependent. The number of objects that are filtered, and the size they appear in the image, both affect the number of pixels that become colored black.



(**a**) Without decluttering (**b**) With decluttering

Figure 18. The Toyplane being assembled.

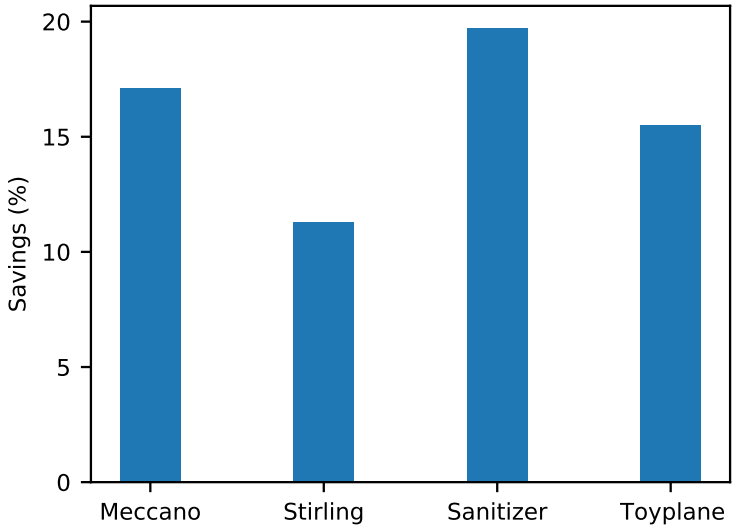


Figure 19. Bandwidth saved by decluttering. Higher values represent more bandwidth savings.

We compare the accuracy for our applications on the new test data, with and without decluttering. These results are shown in Figure 20. The accuracy is higher when using decluttering, for three out of four applications. We suspect that this is due to the fact that our training data did not contain any cluttered images. Decluttering results in lower accuracy for Stirling, but the difference is only around 1%.

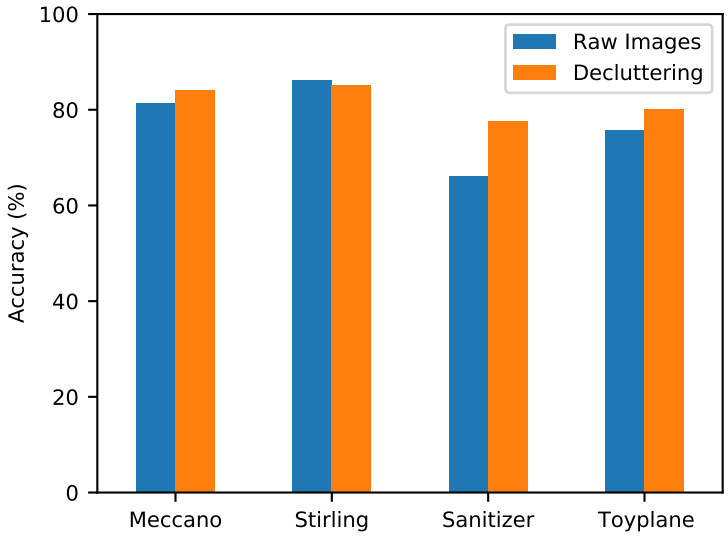


Figure 20. Top-1 classification accuracy for pipelines with and without decluttering.

We compare the average latency and power consumption with and without decluttering, for all three devices. Average per-frame latency is shown in Figure 21 and average power consumption is shown in Figure 22. Latency measurements include time for decluttering (if applicable), JPEG encoding, network transmission, and processing on the cloudlet.

The per-frame latency when using decluttering on Google Glass was almost under the tight bound from [43]. It was well under the tight bound on Magic Leap. Decluttering is thus reasonable on these devices. However, the latency was well above this bound on Vuzix Blade. Decluttering did not cause a significant change in power consumption on any device. Power consumption should thus not be a concern when a developer is deciding whether or not to use decluttering in a WCA application.

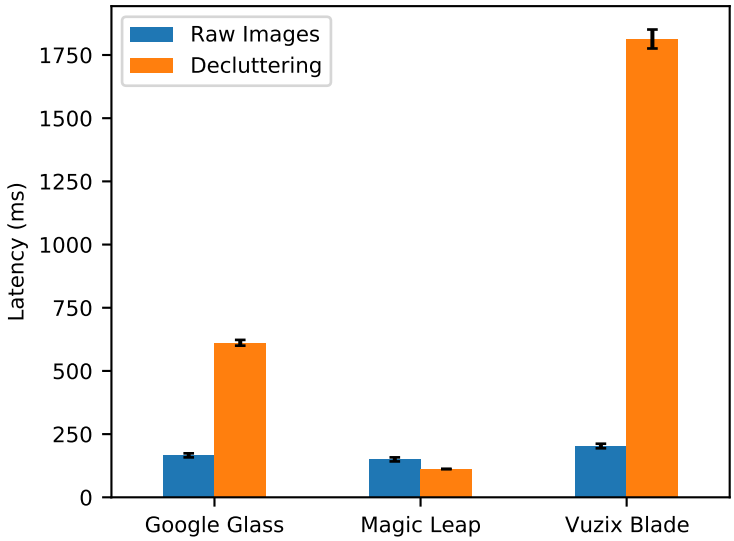


Figure 21. Average per-frame latency with and without decluttering.

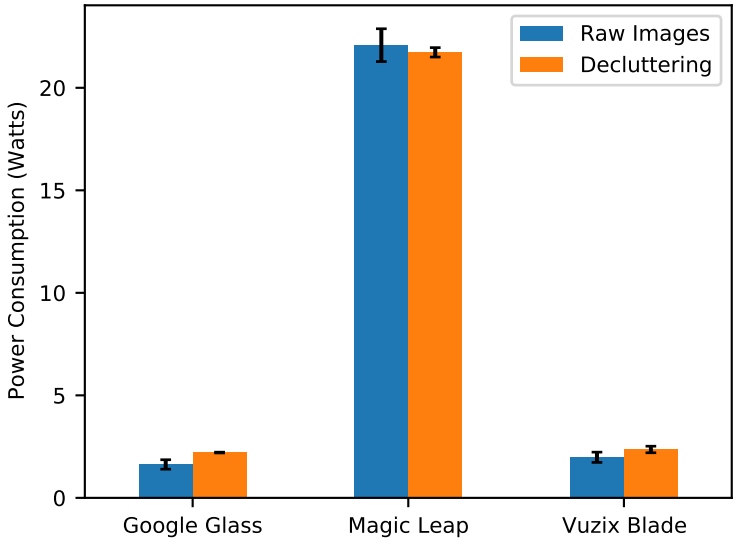


Figure 22. Average power consumption with and without decluttering.

7. Conclusions

We explore the concept of offload shaping for WCA. Our results show that a two stage pipeline consisting of an object detector and an image classifier is effective in four WCA applications. A thin client strategy offers the highest accuracy and lowest latency. However, split computing offers significant bandwidth savings compared to full offloading. Gating also offers a sizeable bandwidth savings if a developer is willing to have users indicate

when they believe that a step is complete. Lastly, image decluttering offers bandwidth savings when there is clutter in the space around the object that a user is assembling.

None of the mobile devices we tested were specifically designed for WCA. All of them ran the thin clients well. However, once local compute was needed in the critical path of execution, the result varied dramatically. The Google Glass Enterprise Edition 2 and the Magic Leap 2 are both capable of running some version of each offload shaping technique with reasonable latency. Offload shaping thus enables a significant reduction in bandwidth usage for WCA while the increases in device power consumption and decreases in accuracy and end-to-end latency are minimal.

Looking to the future, mobile device hardware is likely to improve. More compute-intensive DNNs for object detection and image classification may also emerge. The accuracy of DNNs that can be run on future mobile devices within the tight latency bound may also improve. While it is hard to predict the net effect of all these changes, it is clear that offload shaping will continue to be valuable. Quantitative comparisons of fully local, split, and thin client approaches should therefore inform the optimal partitioning strategy at each point in time.

Author Contributions: Conceptualization, R.I., Q.D., C.N., and M.S.; software, R.I., Q.D., and C.N.; validation, P.P. and M.S.; writing—original draft preparation, R.I.; writing—review and editing, R.I., Q.D., C.N., P.P., and M.S.; visualization, R.I.; supervision, M.S.; funding acquisition, M.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research was sponsored by the National Science Foundation under award number CNS-2106862. This work was done in the CMU Living Edge Lab, which is supported by Intel, ARM, Vodafone, Deutsche Telekom, CableLabs, Crown Castle, InterDigital, Seagate, Microsoft, the VMware University Research Fund, and the Conklin Kistler family fund. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring entity or the U.S. government.

Institutional Review Board Statement: All subjects gave their informed consent for inclusion before they participated in the study. Ethics approval is not required for this type of study. The study was conducted following the local legislation: https://www.ecfr.gov/current/title-45/subtitle-B/chapter-VI/part-690/section-690.104 (accessed on 30 September 2024).

Informed Consent Statement: All the subjects involved in the study were authors of the paper.

Data Availability Statement: The source code and model weights for the Stirling application are shared freely under an MIT license at https://github.com/cmusatyalab/gabriel-stirling-engine (accessed on 30 September 2024). Please contact the authors by email to request access to the other applications and datasets evaluated in this paper.

Acknowledgments: Anthony Rowe and Edward Lu provided us with access to a Magic Leap headset.

Conflicts of Interest: Author Padmanabhan Pillai was employed by the company Intel. The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest. The authors declare that this study received funding from the National Science Foundation, Intel, ARM, Vodafone, Deutsche Telekom, CableLabs, Crown Castle, InterDigital, Seagate, Microsoft, the VMware University Research Fund, and the Conklin Kistler family fund. Funders were not involved in the study design, collection, analysis, interpretation of data, the writing of this article, or the decision to submit it for publication.

References

1. Satyanarayanan, M.; Bahl, P.; Caceres, R.; Davies, N. The Case for VM-Based Cloudlets in Mobile Computing. *IEEE Pervasive Comput.* **2009**, *8*, 14–23. [CrossRef]
2. Satyanarayanan, M.; Klas, G.; Silva, M.; Mangiante, S. The Seminal Role of Edge-Native Applications. In Proceedings of the 2019 IEEE International Conference on Edge Computing (EDGE), Milan, Italy, 1 July 2019; pp. 33–40.
3. Hu, W.; Amos, B.; Chen, Z.; Ha, K.; Richter, W.; Pillai, P.; Gilbert, B.; Harkes, J.; Satyanarayanan, M. The Case for Offload Shaping. In Proceedings of the International Workshop on Mobile Computing Systems and Applications (HotMobile), Santa Fe, NM, USA, 12–13 February 2015; pp. 51–56.

4. Lane, N.D.; Bhattacharya, S.; Georgiev, P.; Forlivesi, C.; Jiao, L.; Qendro, L.; Kawsar, F. DeepX: A Software Accelerator for Low-Power Deep Learning Inference on Mobile Devices. In Proceedings of the 2016 15th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN), Vienna, Austria, 11–14 April 2016; pp. 1–12.
5. Lee, R.; Venieris, S.I.; Dudziak, L.; Bhattacharya, S.; Lane, N.D. MobiSR: Efficient On-Device Super-Resolution through Heterogeneous Mobile Processors. In Proceedings of the 25th Annual International Conference on Mobile Computing and Networking, Los Cabos, Mexico, 21–25 October 2019.
6. Mathur, A.; Lane, N.D.; Bhattacharya, S.; Boran, A.; Forlivesi, C.; Kawsar, F. DeepEye: Resource Efficient Local Execution of Multiple Deep Vision Models Using Wearable Commodity Hardware. In Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys2017), Niagara Falls, NY, USA, 19–23 June 2017; pp. 68–81.
7. Ha, K.; Chen, Z.; Hu, W.; Richter, W.; Pillai, P.; Satyanarayanan, M. Towards Wearable Cognitive Assistance. In Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services, Bretton Woods, NH, USA, 16–19 June 2014; pp. 68–81.
8. Iyengar, R.; Dong, Q.; Nguyen, C.; Pillai, P.; Satyanarayanan, M. Offload Shaping for Wearable Cognitive Assistance. In Proceedings of the 2023 IEEE International Conference on Edge Computing and Communications (EDGE), Chicago, IL, USA, 2–8 July 2023; pp. 183–189. [CrossRef]
9. Satyanarayanan, M.; Gao, W.; Lucia, B. The Computing Landscape of the 21st Century. In Proceedings of the 20th International Workshop on Mobile Computing Systems and Applications, HotMobile'19, New York, NY, USA, 27–28 February 2019; pp. 45–50. [CrossRef]
10. Eshratifar, A.E.; Esmaili, A.; Pedram, M. BottleNet: A Deep Learning Architecture for Intelligent Mobile Cloud Computing Services. In Proceedings of the 2019 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED), Lausanne, Switzerland, 29–31 July 2019; pp. 1–6.
11. Matsubara, Y.; Baidya, S.; Callegaro, D.; Levorato, M.; Singh, S. Distilled Split Deep Neural Networks for Edge-Assisted Real-Time Systems. In Proceedings of the 2019 MobiCom Workshop on Hot Topics in Video Analytics and Intelligent Edges, Los Cabos Mexico, 21 October 2019; pp. 21–26.
12. Hsu, K.J.; Bhardwaj, K.; Gavrilovska, A. Couper: DNN Model Slicing for Visual Analytics Containers at the Edge. In Proceedings of the 4th ACM/IEEE Symposium on Edge Computing, Arlington, VA, USA, 7–9 November 2019; pp. 179–194.
13. Li, G.; Liu, L.; Wang, X.; Dong, X.; Zhao, P.; Feng, X. Auto-tuning Neural Network Quantization Framework for Collaborative Inference Between the Cloud and Edge. In Proceedings of the Artificial Neural Networks and Machine Learning (ICANN 2018), Rhodes, Greece, 4–7 October 2018; pp. 402–411.
14. Jeong, H.J.; Jeong, I.; Lee, H.J.; Moon, S.M. Computation Offloading for Machine Learning Web Apps in the Edge Server Environment. In Proceedings of the IEEE 38th International Conference on Distributed Computing Systems (ICDCS), Vienna, Austria, 2–6 July 2018; pp. 1492–1499.
15. Choi, H.; Bajic, I.V. Deep Feature Compression for Collaborative Object Detection. In Proceedings of the 2018 25th IEEE International Conference on Image Processing (ICIP), Athens, Greece, 7–10 October 2018; pp. 3743–3747.
16. Matsubara, Y.; Levorato, M.; Restuccia, F. Split Computing and Early Exiting for Deep Learning Applications: Survey and Research Challenges. *ACM Comput. Surv.* **2022**, *55*, 1–30. [CrossRef]
17. Noble, B.; Satyanarayanan, M.; Narayanan, D.; Tilton, J.; Flinn, J.; Walker, K. Agile Application-Aware Adaptation for Mobility. In Proceedings of the 16th ACM Symposium on Operating Systems Principles, Saint-Malo, France, 5–8 October 1997; pp. 276–287.
18. Flinn, J.; Satyanarayanan, M. Energy-aware Adaptation for Mobile Applications. In Proceedings of the 17th ACM Symposium on Operating Systems and Principles, Charleston, SC, USA, 12–15 December 1999; pp. 48–63.
19. Satyanarayanan, M. Pervasive Computing: Vision and Challenges. *IEEE Pers. Commun.* **2001**, *8*, 10–17. [CrossRef]
20. Balan, R.; Flinn, J.; Satyanarayanan, M.; Sinnamohideen, S.; Yang, H. The Case for Cyber Foraging. In Proceedings of the 10th Workshop on ACM SIGOPS European Workshop, Saint-Emilion, France, 1 July 2002; pp. 87–92.
21. Flinn, J.; Narayanan, D.; Satyanarayanan, M. Self-Tuned Remote Execution for Pervasive Computing. In Proceedings of the 8th IEEE Workshop on Hot Topics in Operating Systems, Schloss Elmau, Germany, 20–22 May 2001; pp. 61–66.
22. Flinn, J.; Park, S.; Satyanarayanan, M. Balancing Performance, Energy Conservation and Application Quality in Pervasive Computing. In Proceedings of the 22nd International Conference on Distributed Computing Systems, Vienna, Austria, 2–5 July 2002; pp. 217–226.
23. Narayanan, D.; Satyanarayanan, M. Predictive Resource Management for Wearable Computing. In Proceedings of the MobiSys 2003: The First International Conference on Mobile Systems, Applications, and Services, San Francisco, CA, USA, 5–8 May 2003; pp. 113–128.
24. Goyal, S.; Carter, J. A Lightweight Secure Cyber Foraging Infrastructure for Resource-constrained Devices. In Proceedings of the 6th IEEE Workshop on Mobile Computing Systems and Applications, Windermere, UK, 3 December 2004; pp. 186–195.
25. Ya-Yunn, S.; Flinn, J. Slingshot: Deploying Stateful Services in Wireless Hotspots. In Proceedings of the 3rd International Conference on Mobile Systems, Applications, and Services, Seattle, WA, USA, 6–8 June 2005; pp. 79–92.
26. Ok, M.; Seo, J.W.; Park, M.s. A Distributed Resource Furnishing to Offload Resource-Constrained Devices in Cyber Foraging Toward Pervasive Computing. In *Network-Based Information Systems*; Lecture Notes in Computer Science; Enokido, T.; Barolli, L.; Takizawa, M., Eds.; Springer: Berlin/Heidelberg, Germany, 2007; Volume 4658.

27. Balan, R.; Gergle, D.; Satyanarayanan, M.; Herbsleb, J. Simplifying Cyber Foraging for Mobile Devices. In Proceedings of the 5th International Conference on Mobile Systems Applications and Services, San Juan, Puerto Rico, 11–13 June 2007; pp. 272–285.
28. Kristensen, M.D. Execution Plans for Cyber Foraging. In Proceedings of the MobMid '08: Proceedings of the 1st Workshop on Mobile Middleware, Leuven, Belgium, 1–5 December 2008; pp. 1–6.
29. Flinn, J. *Cyber Foraging: Bridging Mobile and Cloud Computing via Opportunistic Offload*; Morgan & Claypool Publishers: Melbourne, Australia, 2012.
30. Cuervo, E.; Balasubramanian, A.; Cho, D.k.; Wolman, A.; Saroiu, S.; Chandra, R.; Bahl, P. MAUI: Making Smartphones Last Longer with Code Offload. In Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services, San Francisco, CA, USA, 15–18 June 2010; pp. 49–62.
31. Ra, M.; Sheth, A.; Mummert, L.; Pillai, P.; Wetherall, D.; Govindan, R. Odessa: Enabling Interactive Perception Applications on Mobile Devices. In Proceedings of the International Conference on Mobile Systems, Applications, and Services (MobiSys 2011), Bethesda, MD, USA, 28 June 2011; pp. 43–56.
32. Chun, B.G.; Ihm, S.; Maniatis, P.; Naik, M.; Patti, A. CloneCloud: Elastic Execution between Mobile Device and Cloud. In Proceedings of the EuroSys 2011, Salzburg, Austria, 10–13 April 2011; pp. 301–314.
33. Gordon, M.S.; Jamshidi, D.A.; Mahlke, S.; Mao, Z.M.; Chen, X. COMET: Code Offload by Migrating Execution Transparently. In Proceedings of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12), Hollywood, CA, USA, 8–10 October 2012; pp. 93–106.
34. Sobel, I.; Feldman, G. A 3×3 isotropic gradient operator for image processing. Presented at a Talk at the Stanford Artificial Project. In *Pattern Classification and Scene Analysis*; John Wiley and Sons: New York, NY, USA, 1968; pp. 271–272.
35. Szeliski, R. *Computer Vision: Algorithms and Applications*; Springer: Berlin/Heidelberg, Germany, 2010.
36. Zauner, C. Implementation and Benchmarking of Perceptual Image Hash Functions. Ph.D. Thesis, University of Applied Sciences Hagenberg, Mühlkreis, Austria, 2010.
37. Monga, V.; Evans, B. Perceptual image hashing via feature points: Performance evaluation and tradeoffs. *IEEE Trans. Image Process.* **2006**, *15*, 3452–3465. [CrossRef]
38. Kozat, S.S.; Venkatesan, R.; Mihçak, M.K. Robust perceptual image hashing via matrix invariants. In Proceedings of the ICIP'04: International Conference on Image Processing, Singapore, 24–27 October 2004; pp. 3443–3446.
39. Wang, J.; Feng, Z.; Chen, Z.; George, S.; Bala, M.; Pillai, P.; Yang, S.W.; Satyanarayanan, M. Bandwidth-efficient Live Video Analytics for Drones via Edge Computing. In Proceedings of the 2018 IEEE/ACM Symposium on Edge Computing (SEC), Seattle, WA, USA, 9 December 2018; pp. 159–173.
40. Kang, Y.; Hauswald, J.; Gao, C.; Rovinski, A.; Mudge, T.; Mars, J.; Tang, L. Neurosurgeon: Collaborative Intelligence Between the Cloud and Mobile Edge. *ACM Sigarch Comput. Archit. News* **2018**, *45*, 615–629. [CrossRef]
41. Chen, K.; Li, T.; Kim, H.S.; Culler, D.E.; Katz, R.H. MARVEL: Enabling Mobile Augmented Reality with Low Energy and Low Latency. In Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems, SenSys'18, New York, NY, USA, 4–7 November 2018; pp. 292–304. [CrossRef]
42. Liu, L.; Li, H.; Gruteser, M. Edge Assisted Real-Time Object Detection for Mobile Augmented Reality. In Proceedings of the 25th Annual International Conference on Mobile Computing and Networking, MobiCom'19, New York, NY, USA, 21–25 October 2019. [CrossRef]
43. Chen, Z.; Hu, W.; Wang, J.; Zhao, S.; Amos, B.; Wu, G.; Ha, K.; Elgazzar, K.; Pillai, P.; Klatzky, R.; et al. An Empirical Study of Latency in an Emerging Class of Edge Computing Applications for Wearable Cognitive Assistance. In Proceedings of the Second ACM/IEEE Symposium on Edge Computing, San Jose, CA, USA, 12–14 October 2017; pp. 1–14.
44. Satyanarayanan, M.; Davies, N. Augmenting Cognition through Edge Computing. *IEEE Comput.* **2019**, *52*, 37–46. [CrossRef]
45. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep Residual Learning for Image Recognition. In Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 26 June–1 June 2016; pp. 770–778.
46. Li, P.; Xie, J.; Wang, Q.; Gao, Z. Towards Faster Training of Global Covariance Pooling Networks by Iterative Matrix Square Root Normalization. In Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR), Salt Lake City, UT, USA, 18–22 June 2018; pp. 947–955.
47. Ren, S.; He, K.; Girshick, R.; Sun, J. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Trans. Pattern Anal. Mach. Intell.* **2016**, *39*, 1137–1149. [CrossRef]
48. Tan, M.; Pang, R.; Le, Q.V. EfficientDet: Scalable and Efficient Object Detection. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Seattle, WA, USA, 14–19 June 2020; pp. 10781–10790 .
49. Gebru, T.; Krause, J.; Wang, Y.; Chen, D.; Deng, J.; Fei-Fei, L. Fine-Grained Car Detection for Visual Census Estimation. In Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, San Francisco, CA, USA, 4–9 February 2017; pp. 4502–4508.
50. Wang, J.; Feng, Z.; George, S.; Iyengar, R.; Pillai, P.; Satyanarayanan, M. Towards Scalable Edge-Native Applications. In Proceedings of the Fourth IEEE/ACM Symposium on Edge Computing (SEC 2019), Arlington, VA, USA, 7–9 November 2019; pp. 152–165.
51. TensorFlow. Running TF2 Detection API Models on Mobile. Available online: https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/running_on_mobile_tf2.md (accessed on 19 May 2023).

52. PyTorch. Autodiff for User Script Functions Aka torch.jit.script for autograd.Function. Available online: https://github.com/pytorch/pytorch/issues/22329 (accessed on 19 May 2023).
53. Lugaresi, C.; Tang, J.; Nash, H.; McClanahan, C.; Uboweja, E.; Hays, M.; Zhang, F.; Chang, C.L.; Yong, M.; Lee, J.; et al. MediaPipe: A Framework for Perceiving and Processing Reality. In Proceedings of the Third Workshop on Computer Vision for AR/VR at IEEE Computer Vision and Pattern Recognition (CVPR) 2019, Long Beach, CA, USA, 15–20 June 2019.
54. CMUSphinx. PocketSphinx. Available online: https://github.com/cmusphinx/pocketsphinx (accessed on 19 May 2023).
55. Azure Cognitive Services. Speech Containers. Available online: https://learn.microsoft.com/en-us/azure/cognitive-services/cognitive-services-container-support#speech-containers (accessed on 19 May 2023).
56. Lin, T.Y.; Maire, M.; Belongie, S.; Hays, J.; Perona, P.; Ramanan, D.; Dollár, P.; Zitnick, C.L. Microsoft COCO: Common Objects in Context. In Proceedings of the Computer Vision—ECCV 2014, Zurich, Switzerland, 6–12 September 2014; Fleet, D.; Pajdla, T.; Schiele, B., Tuytelaars, T., Eds.; Springer: Cham, Switzerland, 2014.