Scaling Wearable Cognitive Assistance

Junjue Wang
junjuew@cs.cmu.edu

June 2018

Thesis Proposal

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Mahadev Satyanarayanan (Satya) (Chair)
Daniel Siewiorek
Martial Hebert
Roberta Klatzky
Padmanabhan Pillai (Intel Labs)

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy.

Contents

1 Introduction

Wearable Cognitive Assistance has emerged as a new genre of applications that pushes the boundaries of augmented cognition. These applications continuously process data from body-worn sensors and provide just-in-time guidance to help a user complete a specific task. For example, an IKEA Lamp assistant [6] has been built to assist the assembly of a table lamp. To use the application, a user wears a head-mounted smart glass that continuously captures her actions and surroundings from a first-person viewpoint. In real-time, the camera stream is analyzed to identify the state of the assembly. Audiovisual instructions are generated based on the detected state. The instructions either demonstrate a subsequent procedure or alert and correct a mistake.

Although Wearable Cognitive Assistance shares the vision of cognition enhancement with many previous research efforts [35] [39] [9] [49], its design goals advance the frontier of mobile computing in multiple aspects. First, wearable devices, particularly head-mounted smart glasses, are used to reduce the discomfort caused by carrying a bulky computation device. Users are freed from holding a smartphone and therefore able to interact with the physical world using both hands. The convenience of this interaction model comes at the cost of constrained computation resources. The small form-factor of smart glasses significantly limits their onboard computation capability due to size, cooling, and battery life reasons. Second, placed at the center of computation is the unstructured high-dimensional image and video data. Only these data types can satisfy the need to extract rich semantic information to identify the progress and mistakes a user makes. Furthermore, state-of-art computer vision algorithms used to analyze image data are both compute-intensive and challenging to develop. Third, many cognitive assistants give real-time feedback to users and have stringent end-to-end latency requirements. An instruction that arrives too late often provides no value and may even confuse or annoy users. This latency-sensitivity further increases their high demands of system resource and optimizations.

To meet the latency and the compute requirements, previous research leverages edge computing and offloads computation to a cloudlet. A cloudlet [47] is a small data-center located at the edge of the Internet, one wireless hop away from users. Researchers have developed an application framework for wearable cognitive assistance, named Gabriel, that leverages cloudlets, optimizes for end-to-end latency, and eases application development [6] [19] [8]. On top of Gabriel, several prototype applications have been built, such as Ping-Pong Assistance, Lego Assistance, Sandwich Assistance, and Ikea Lamp Assembly Assistance. Using these applications as benchmarks, [8] presents empirical measurements detailing the latency contributions of individual system components. Furthermore, a multi-algorithm approach was proposed to reduce the latency of computer vision computation by executing multiple algorithms in parallel and conditionally selecting a fast and accurate algorithm for the near future.

While previous research has demonstrated the technical feasibility of wearable cognitive assistants and meeting latency requirements, many practical concerns have not been addressed. First, previous work operates the wireless networks and cloudlets at low utilization in order to meet application latency. The economics of practical deployment preclude operation at such low utilization. In contrast, resources are often highly utilized and congested when serving many users. How to efficiently scale Gabriel applications to a large number of users remains to be answered. Second, previous work on the Gabriel framework reduces application development efforts by managing client-server communication, network flow control, and cognitive engine

discovery. However, the framework does not address the most time-consuming parts of creating a wearable cognitive assistance application. Experience has shown that developing computer vision modules that analyze video feeds is a time-consuming and painstaking process that requires special expertise and involves rounds of trial and error. Developer tools that alleviate the time and the expertise needed can greatly facilitate the creation of these applications.

This proposal lays out my plan to address these challenges. In order to meet latency requirements when utilization is high, restricting the freedom of using resources while taking account of workload characteristics is needed. The scarce resource can either be the wireless links or the cloudlets. First, upload bandwidth in cellular networks is limited compared to download bandwidth and has high variance. Existing wireless infrastructure cannot afford to continuously stream high-definition videos from many users. I plan to address this problem with application-level mechanisms that exploit the attributes of the workload to reduce bandwidth consumption. Second, accelerators, such as GPUs, on cloudlets are both limited and heterogeneous. Due to the high demands of accelerators from state-of-art computer vision algorithms, the intelligent discovery of accelerator resources and the usage coordination among applications are required to serve more users. I plan to work on these problems in an edge computing context to address how to discover appropriate cloudlets for offload and how to coordinate among applications with different latency requirements to share scarce accelerators.

In order to address the difficulty of development, I plan to build tools to reduce the expertise and time needed when creating wearable cognitive assistants. First, state-of-art computer vision uses Deep Neural Networks (DNNs) for critical tasks, including image classification, object detection, and semantic segmentation. DNNs champion end-to-end learning instead of hand-crafted features. The absence of manually created features provides an opportunity to build developer tools that replace ad-hoc trial and error development process. On the other hand, DNNs requires a significant amount of labeled data for training. I plan to build tools that help label examples and automate the creation of DNN-based object detectors.

My thesis is that these efforts can help to scale wearable cognitive assistance. Notably, we claim that:

Two critical challenges to the widespread adoption of wearable cognitive assistance are 1) the need to operate cloudlets and wireless network at low utilization to achieve acceptable end-to-end latency 2) the level of specialized skills and the long development time needed to create new applications. These challenges can be effectively addressed through system optimizations, functional extensions, and the addition of new software development tools to the Gabriel platform.

2 Motivation

2.1 Resource Constraints

2.1.1 Characteristics of Wearable Cognitive Assistance Workload

Wearable cognitive assistance applications are both latency-sensitive and compute intensive. Latency-sensitivity is inherent to applications. The arrival rate of instructions needs to match the time for humans to execute these instructions. For instance, humans can recognize a person's

	Pool	Work-out	Ping-pong	Face	Lego	Draw	Sandwich
Tight Bound (ms)	95	300	150	370	600		

Figure 1: Application Latency Requirements

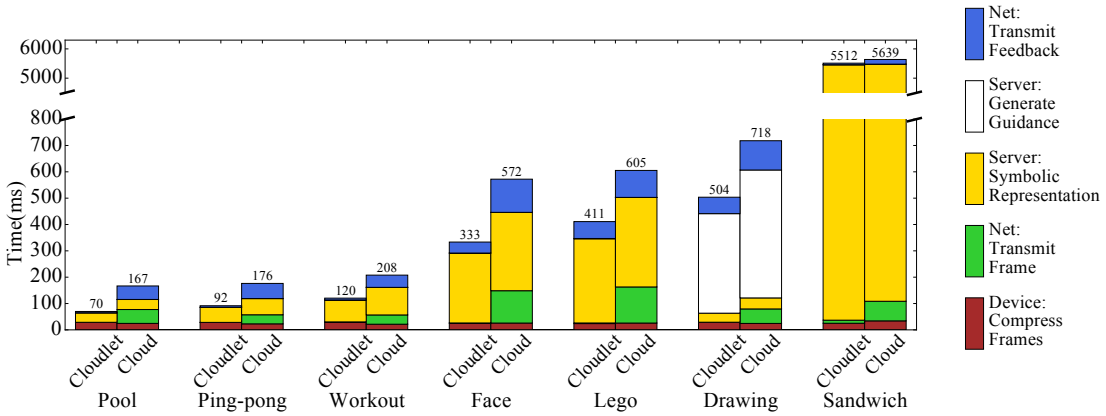


Figure 2: Mean Latency Breakdown - Cloudlet vs. Cloud for Phone over WiFi

face using around 1000 ms [34]. For a digital assistant that reminds a person who a person is, the speed needs to be much faster than this value.

The high compute demand of wearable cognitive assistance comes from the processing needs of modern computer vision algorithms. The accuracy of many important computer vision tasks, such as object detection, image segmentation, and face recognition, have been greatly improved since the advent of deep neural networks (DNNs). While accuracy has improved, the computation demand has also increased. Deep neural networks have tens to hundreds of perceptron layers and millions of parameters. Both DNN train and inferencing involves millions of multiply-and-add operations, which are implemented as matrix multiplication. These large matrix multiplications have a high computational demand.

Previous measurements on wearable cognitive assistants provide quantitative insights into both latency sensitivity and computation intensity. Figure 1 shows latency requirements for seven different applications. These applications serve a variety of purposes, from guiding a user how to aim in a game of pool to teaching a user how to make a sandwich. Chen et al. [8] describe how these bounds are obtained. These latency requirements highlight the latency-sensitivity of wearable cognitive assistants.

Figure 2 shows the time breakdown of these assistants. For the cloudlet case (left bars for each application), relatively little time is spent on network transmission, which is the benefit of edge computing. When offloading to the cloudlet, the server compute time consists of a significant portion, if not the most significant part of the end-to-end latency. For applications that use DNNs to process complex scenes, for example, Face and Sandwich, the server computation time dominates.

2.1.2 Wireless Network Characteristics

The sensed data, e.g. images and videos from mobile devices, are transmitted to a cloudlet through wireless communication. While IEEE 802.11 protocol is widely-used at home and enterprise settings, the cellular network, particularly LTE provides a better experience with its always-on ubiquitousness and large area coverage. To this end, I will focus on LTE network as the primary wireless communication protocol in the rest of my work.

LTE has unique characteristics in both latency and bandwidth. While previous works [28] and [21] have experimentally measured LTE's latency for mobile applications, the influence of LTE's bandwidth and capacity on wearable cognitive assistance applications is mostly unexplored.

The LTE uplink bandwidth is both limited and highly variant. LTE release 8 has a theoretical a peak uplink data rate of 75 Mbps compared to 300 Mbps peak downlink data rate. For LTE-Advanced, a major step toward 5G, the peak uplink and downlink data rate are 1500 Mbps and 3000 Mbps respectively. In addition, these theoretical peak data rates can only be achieved under idealized set-up condition. For example, the LTE cell in test needs to be well isolated from other cells while the test mobile device needs to be close to the base station. Actual deployment has less bandwidth available [10]. For a single user, the available bandwidth can be even less due to sharing. For example, a 2012 study measured median LTE downlink and uplink bandwidth in US commercial network to be 13 Mbps and 6 Mbps respectively [30]. In 2017, the highest average upload bandwidth is 18 Mbps among providers in US [40]. In addition, LTE bandwidth has high variance [53]. [30] observed high variation on LTE throughput not only for different users at different locations, but also for the same user at the same location across different runs.

Gabriel applications put a high bandwidth demand on the network since they continuously stream video data to cloudlets over wireless links. As a comparison, Netflix recommends an internet connection speed of 5 Mbps to watch its highly compressed HD video content. Gabriel applications would consume more bandwidth due to on-the-fly compression. Hundreds of users simultaneously streaming high-definition videos could easily saturate today's LTE uplink capabilities [43]. Therefore, effective reduction of the bandwidth consumption is essential to support concurrent running of many Gabriel applications.

2.2 Accelerators on Edge Nodes

The computation capability of cloudlets, especially hardware accelerators, also becomes a bottleneck with many users. The high demand for accelerator resources comes from the widespread use of DNNs to solve computer vision tasks. DNNs' superior learning ability enables them to advance the state-of-art accuracy. The large capacity to learn comes from the sheer number of model parameters. For example, the widely-used image classification network ResNet-101 [25] has more than 42 million parameters. Another classifier network Inception ResNet V2 which achieves higher accuracy on ImageNet [13] has more than 53 million parameters [29]. For more sophisticated tasks, for example object detection, additional layers are needed beyond image classifiers, resulting in even more parameters. In addition, these networks are deep, with tens to hundreds of layers in a sequence. Computational dependencies exist among the bottom and top layers. The combined effect of a large number of both parameters and layers is the high computational demands when using DNNs for inference. In order to shorten computational latency,

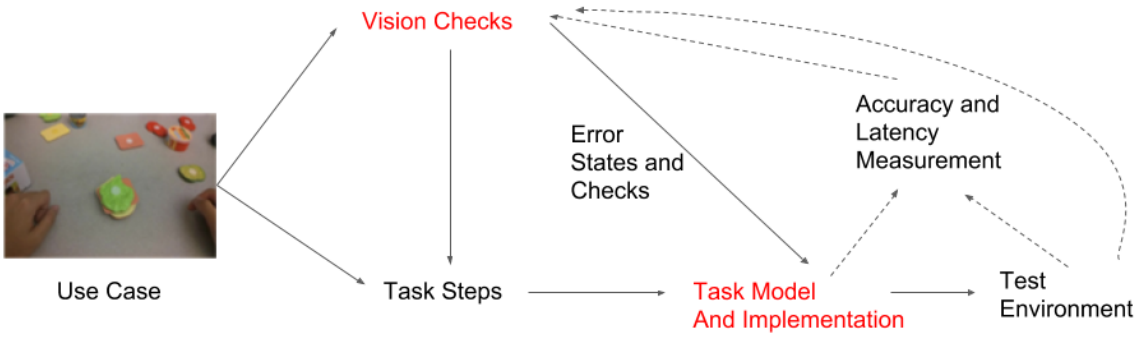


Figure 3: Development Workflow

hardware accelerators that can exploit parallel execution, for example GPUs, are commonly used for both DNN training and inference. ASIC accelerators to further improve speed have also been developed. For example, Google has deployed Tensor Processing Units (TPUs) into their cloud to meet DNNs' computational demands [33].

Despite many efforts in hardware design, the cost of DNN accelerators remains high. For example, a Nvidia Pascal Titan X graphic card, a top-of-line GPU for DNNs, costs around $1200 [41]. Nevertheless, it can only run the object detection network Faster-RCNN with ResNet101 at a speed of fewer than 10 frames per second (FPS) [29]. Although large batch sizes could theoretically improve the throughput, they often sacrifice application latency and therefore are not suitable for many Gabriel applications. To support more users with high demands on accelerators, Gabriel framework needs to efficiently manage and share scarce and expensive acceleration resources among concurrent users to reduce the cost of deployment.

2.3 Difficulty of Development

Gabriel applications are difficult to develop. Not only is the development process slow, but specialized computer vision experience is also needed. It took a researcher four months to build his first cognitive assistant for assembling LEGO pieces [6]. The majority of time is spent on learning the computer vision techniques and selecting robust algorithms to use through trial and error. New developers would face the similar obstacles when building these applications from scratch. As a result, the number of wearable cognitive assistants available is quite limited. Therefore, Gabriel needs to be extended with a toolchain to reduce the difficulty of development.

The overall development workflow of wearable cognitive assistance is shown in figure 3. After a use case is identified, developers would need to identify meaningful visual states that can be detected using computer vision. In the meantime, a task is divided into steps based on the use case and detectable visual states. Task procedures could be changed to reduce the difficulties of CV checks. In fact, since there is a human in the loop, relying on humans to do what they are good at is the main reason that wearable cognitive assistance can be implemented without solving perception and planning problems intrinsic to robotics. Task procedures together with error states form a task model. Developers implement the application according to the task model. After initial implementation, test runs and measurements are conducted to evaluate the robustness of computer vision checks and end-to-end application latency. This process is iterated

until developers are satisfied.

Among all the development procedures, creating the computer vision checks to detect user states consumes the most of development time and requires computer vision expertise and experience. With the adoption of DNNs, developers no longer need to spend days to select and tweak handcrafted features. Instead, the entire model is trained end-to-end using labeled data. However, DNNs, with millions of parameters to train, requires a significant amount of training data. Collecting and labeling data are time-consuming and painstaking. Besides, to craft and implement a DNN by hand is not trivial. Significant machine learning background is needed to tweak network architectures and parameters. Therefore, developer tools are needed to both help label the data and create deep neural networks.

In summary, implementing the workflow of cognitive assistance takes time and efforts. Ad-hoc implementation requires a team of domain experts, developers and computer vision experts. Such development model cannot scale to thousands of applications. Therefore, Gabriel needs to be extended with tools to reduce the effort of creating wearable cognitive assistants.

3 Bandwidth Reduction

Gabriel applications continuously stream sensor data to the cloudlet. The richer a sensing modality is, the more information can be extracted. Visual data from cameras is one of such rich sensing modalities that wearable cognitive assistance leverages. However, the last hop wireless bandwidth, especially LTE bandwidth cannot support thousands of users simultaneously streaming high definition videos. Therefore, to effectively scale wearable cognitive assistance, we need to reduce the bandwidth consumed. There are three techniques I will explore to reduce the bandwidth cost by exploiting the unique properties of the workload. We have shown the effectiveness of these techniques with drone footages. Further experiments with first-person videos will be done to show their effectiveness for wearable cognitive assistance.

3.1 Early Discard

Early discard refers to the technique that filters and discards contents in early stages of computation. An example system that uses early discard is an unindexed search system [48]. In the mobile computing context, [27] explored using image-level simple computer vision algorithms, for example blur and color detection, to suppress transmission of uninteresting video frames.

In Gabriel applications, an efficient early discard system that filters out uninteresting frames before transmission can reduce the bandwidth consumption. For example, in the Lego assistant, all the assembled Lego pieces are placed on a Lego board with black boundaries. Some frames do not contain the Lego board due to user movement. Without further analysis, the application knows these frames do not contain useful information since the board is not present. If an efficient contour detection of black boundaries can be performed on the wearable device, the application can save the unnecessary bandwidth consumed to transmit the uninteresting frames.

There exists a tradeoff between computation and network transmission for early discard. To one end, if the wearable device is powerful enough to perform all the computation on-device within latency constraints, the bandwidth cost would be minimal because there is no need to

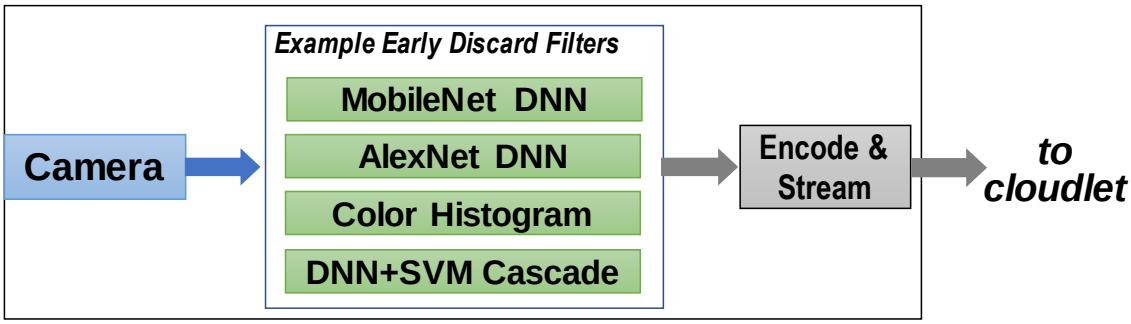


Figure 4: Early Discard Pipeline.

offload the computation. To the other extreme, if the wearable device is very weak, all the sensor data then needs to be shipped to the cloudlets for processing. In fact, image compression can be thought as an early discard technique at the pixel level. It uses the computation onboard to remove the duplicate data sent over the network.

To explore the effectiveness of early discard, we conducted experiments in a small drone setting. Drones typically can carry more computational power than wearable devices. Therefore they can be thought of futuristic wearable devices with large computation capability. In my thesis, I will run similar experiments on wearable devices with first-person videos. Below we will present our experiments on drone videos to demonstrate the effectiveness of early discard.

As shown in Figure 4, we envision a choice of weak detectors being available as early discard filters on a drone, with the specific choice of filter being mission-specific. We use image classification as early discard filters on the drone: it is not necessary to know exactly where in the frame a relevant object occurs. This suggests that MobileNet would be a good choice as a weak detector. Its speed of 352 ms per frame on Nexus 6 yields less than 3 fps. However, its speed of 13 ms per frame on Jetson yields more than 75 fps. If Jetson-class hardware was to be available on future smartphones, MobileNet would be usable at a full frame rate. We therefore use MobileNet on the drone for early discard in our experiments.

Our experiments on the EARLYDISCARD strategy used benchmark tasks suite described in figure 5. We used Nvidia Jetson TX2 as the drone platform. We use both frame-based and event-based metrics to evaluate the MobileNet filters. These experiments demonstrate that significant bandwidth saving can be achieved while maintaining final detection accuracy through interesting frames selection with filters running on the mobile device.

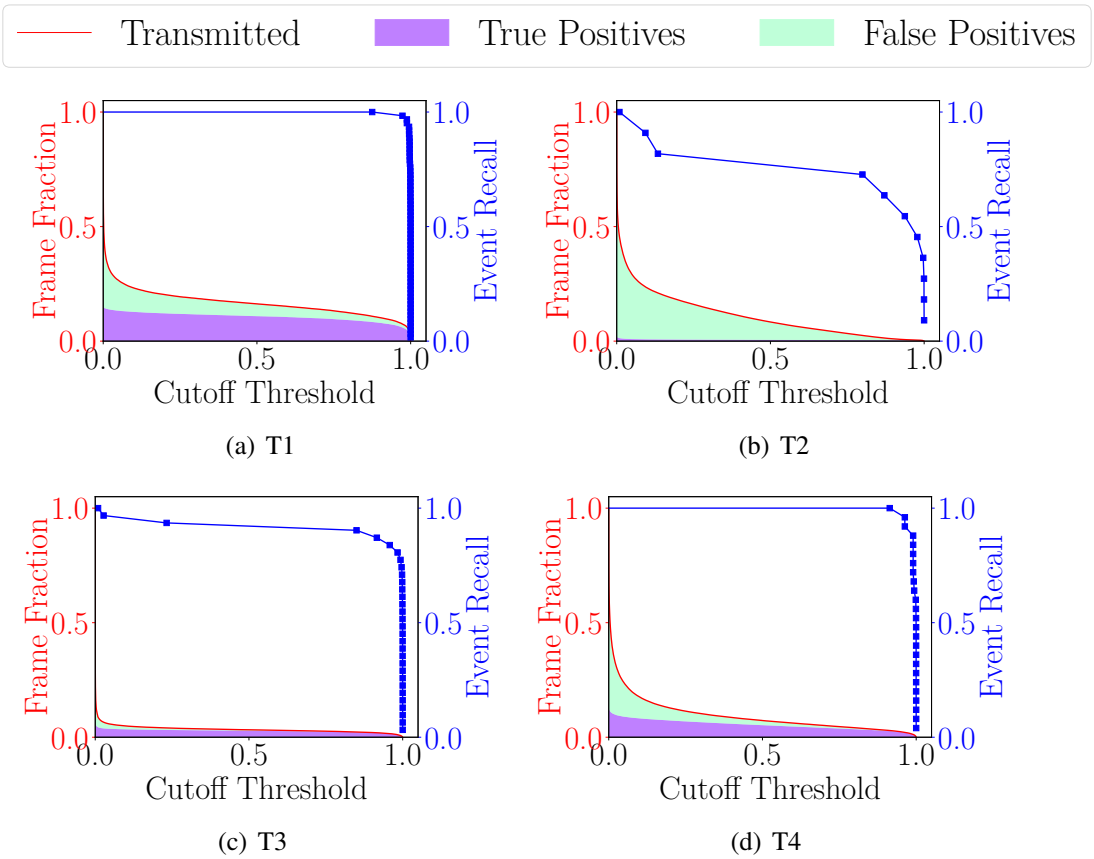
Drone Filter Accuracy: The output of a drone filter is the probability of the current tile being "interesting." A tunable *cutoff threshold* parameter specifies how interesting is "interesting enough" for transmission to the cloudlet. All tiles, whether deemed interesting or not, are still stored in drone storage for post-mission processing.

Events (such as detection of a raft in T3) occur in consecutive frames, all of which contain the object of interest. A correct detection of an event is defined as at least one of the consecutive frames being transmitted to the cloudlet. Blue lines in Figure 6 shows how the event recalls of drone filters for different tasks change as a function of cutoff threshold. As the figure shows, the MobileNet DNN filter is able to detect all the events for T1 and T4 at a high cutoff threshold. For T2 and T3, the majority of the events are detected. Achieving high recall on T2 and T3 (on

Task	Detection Goal	Data Source	Data Attributes	Training Subset	Testing Subset
T1	People in scenes of daily life	Okutama Action Dataset [3]	33 videos 59842 fr 4K@30 fps	9 videos 17763 fr	6 videos 20751 frames
T2	Moving cars	Stanford Drone Dataset [45]	60 videos 522497 fr 1080p@30 fps	16 videos 179992 fr	14 videos 92378 fr Combination of videos from each dataset. T1's test uses a subset that do not have unlabeled human.
T3	Raft in flooding scene	YouTube collection [1]	11 videos 54395 fr 720p@25 fps	8 videos 43017 fr	
T4	Elephants in natural habitat	YouTube collection [2]	11 videos 54203 fr 720p@25 fps	8 videos 39466 fr	

fr = "frames"
fps = "frames per second"
No overlap between training and testing subsets of data

Figure 5: Benchmark Suite of Video Traces



(a) T1 (b) T2

(c) T3 (d) T4

Figure 6: Where the Bandwidth Goes

	Total Events	Detected Events	Avg Delay (s)	Total Data (MB)	Avg B/W (Mbps)	Peak B/W (Mbps)
T1	62	100 %	0.1	441	5.10	10.7
T2	11	73 %	4.9	13	0.03	7.0
T3	31	90 %	12.7	93	0.24	7.0
T4	25	100 %	0.3	167	0.43	7.0

Figure 7: Recall, Event Latency and Bandwidth at Cutoff Threshold 0.5

the order of 0.95 or better) requires setting a low cutoff threshold. This leads to the possibility that many of the transmitted frames are actually uninteresting (i.e., false positives).

False negatives: As discussed earlier, false negatives are a source of concern with early discard. Once the drone drops a frame containing an important event, improved cloudlet processing cannot help. The results in the third column of Figure 7 confirm that there are no false negatives for T1 and T4 at a cutoff threshold of 0.5. For T2 and T3, lower cutoff thresholds are needed to achieve perfect recalls.

Result latency: The contribution of early discard processing to total result latency is calculated as the average time difference between the first frame in which an object occurs (i.e., first occurrence in ground truth) and the first frame containing the object that is transmitted to the backend (i.e., first detection). The results in the fourth column of Figure 7 confirm that early discard contributes little to result latency. The amounts range from 0.1 s for T1 to 12.7 s for T3. At the times cale of human actions such as dispatching of a rescue team, these are negligible delays.

Although the general approach of early discard would apply for wearable cognitive assistance, how to apply this technique to wearable devices still needs investigation. First, wearable devices in general have even less general-purpose computation capabilities. Due to the small form factor and heat dissipation constraints, the mobile CPUs in wearable devices are optimized for power instead of performance. In fact, many smart glasses today rely on a user's smartphone to process information. For instance, one of Google Glasses' main use cases when it was first released is to display notifications from smartphone applications, such as CNN and EverNote [51]. Second, there is a strong trend to put hardware DNN accelerators into embedded devices to perform some analysis of visual data. There are many efforts from both industry and academia. For example, Microsoft HoloLens has a special Holograhic Processing Unit [44] to map the environment in 3D. Recently released Google Pixel 2 smartphone have visual cores built-in for future-proofing although not all of them are being actively used now [5]. In academia, there are strong interests in DNN compression and optimization for embedded devices so that the inference can run natively on device [23] [24]. Therefore, it is likely that smart glasses' computation capabilities, especially their capabilities of executing DNNs, would vary greatly.

I would like to further explore how to create filters for a variety of heterogeneous smart glasses with or without accelerators in my thesis. I plan to formulate the problem as an optimization problem that optimizes for the least amount of bandwidth consumption constraint by

the application latency requirement. To its simplistic form, the problem can be written as

$$\begin{aligned} \text{minimize} \quad & AverageDataTransmittedPerImage \\ \text{subject to} \quad & OnBoardProcessingTime \\ & + NetworkTransmissionTime \\ & + ServerComputationTime \leq ApplicationLatencyRequirement \end{aligned}$$

I also plan to design tools that will generate filters given these constraints.

3.2 Just-in-time Learning

Just-in-time-learning (JITL) refers to the technique that tunes the processing pipeline to the characteristics of the current scenario in order to reduce transmitted false positives from the client, therefore reduce wasted bandwidth. When training DNNs to recognize and detect objects, machine learning experts do their best to obtain training examples from the same distribution of the test data. In other words, best efforts are made to acquire training examples from environments similar to test environment in terms of lighting, occlusion, and many other aspects. However, as a Gabriel application could be used in any environment, it is not realistic to assume all of them can be truthfully represented by the training data. For test environment that does not look similar to the training environment, the detection accuracy could be lower. Just-in-time-learning aims to alleviate the generalization problem by making each test environment represented in the training samples. Such goal can be achieved by quick collecting a small number of examples drawn from the actual test environment and iteratively training an existing model with the newly collected data in a short time.

I plan to focus on mechanical assembly tasks when applying just-in-time learning to wearable cognitive assistance. For mechanical assembly tasks, the environment a user is in usually does not change significantly. For example, a user might sit in front of a desk for most of the time. The relatively stable environment gives us opportunities to leverage the human in the loop to provide false positive examples. Before the application starts, a cognitive assistant could ask the user to look around her environment with the assembly kits put away for a short period of time. By construction, any positive predictions are false positives. They are excellent negative training examples because they are taken in the actual test environment. I plan to explore multiple methods to make use of these newly collected data. First, feature matching could be used to identify subsequent false positives. Specifically, a false positive feature pool can be built for the particular test environment with the newly collected negative data. For each prediction at application runtime, a matching is performed to see if it is close enough to items in the false positive feature pool. Second, researchers have shown positive results from train image segmentation DNNs with a small number of examples for a few iterations to fit the model better to the test environment. I plan to explore similar techniques in object detections.

3.3 Context-awareness

The essence of this approach is to leverage unique attributes of the current task to improve the speed and accuracy of image processing on the mobile device. By definition, this approach is ad

	Precision using DNN (%)	Precision using color filter (%)	Recall (%)
Video 1	92.4	95.3	89.1
Video 2	51.9	76.1	90.0
Video 3	41.3	84.3	88.6

(a) Accuracy

	Jetson		Joule		Nexus 6	
	DNN	Color	DNN	Color	DNN	Color
Video 1	13	6.2	37	9.8	352	27.5
Video 2		6.3		9.7		26.3
Video 3		9.5		12.3		36.1

(b) Processing time (ms)

Figure 8: Detection Results on T3 Using Color Filters

hoc in character. However, the wins can be significant. Consider the fuchsia colored screw in RibLoc cognitive assistant as an example. Since the color of the screw is rarely seen in everyday objects, cheap color filtering instead of DNNs can be used to filter interesting frames.

To demonstrate the effectiveness of this strategy, we apply it in the drone context using a simple color filter in T3. In each raft search video, we randomly pick a frame that contains a raft (true positive), and obtain the color of the most distinctive region of the raft. Using the hue, saturation, and value (HSV) color space attributes of this region, we apply a color filter to all the other frames of the video. If a significantly large area of a frame passes this filter, the frame is marked as positive. Otherwise, it is marked as negative.

Figure 8 shows the results of using this approach on three representative videos in T3. Keeping recall fixed at a high value (fourth column of Figure 8(a)), the second and third columns show the precision achieved using a DNN and a color filter. For all three videos, the precision using a color filter is better than the precision using a DNN. The difference is modest for Video 1, but considerable for Video 2 and Video 3. In other words, the context aware approach is consistently more accurate. This improvement in accuracy does not come at a sacrifice in speed. On the contrary, Figure 8(b) shows that the DNN on the Jetson takes 2–3 times the processing time of the color filter. On the Joule and the Nexus 6, it takes 8–10 times. These results show the high value of using context-aware knowledge. What the DNN provides is generality, combined with reasonable accuracy. At the beginning of a mission, when little is known about the context-specific search attributes of the target, the DNN is the only choice. As the mission progresses, the early results may hint at the attributes of a highly effective and cheap context-aware filter.

Although Gabriel applications do not have a human in the loop to specify filter parameters, context-awareness can still be applied to wearable cognitive assistance. In fact it could serve both as a mean to reduce bandwidth consumption and as a method to reduce server computation latency. To achieve the former goal, we can generate a suite of early-discard filters at training time using different algorithms. When the application first starts, we can ask the user to look around her environment and show us a few objects that will be used. All early-discard filters can be evaluated during this "warm-up" time by checking how many false positives and false

negatives they produce. A filter that performs the best in this test environment will be selected for application usage. To be able to achieve such context-awareness, many challenges exist, e.g. what metrics to use to evaluate these early discard filters, how to quickly identify the best one to use, where should the computation happen, and how to efficiently transmit context-aware updates to the client. I plan to explore these questions in my thesis. In addition, such dynamic selection of CV algorithms can be applied for processing at the edge nodes as well. We could build upon the multi-algorithm acceleration approach described by [8] to dynamically select a cheap algorithm to use from auto-generated models in order to reduce processing latency.

4 Gabriel Deployment System

Previous research on wearable cognitive assistance [8] has focused on meeting application latency requirements and assumed abundant resources on the cloudlets. However, when there are many users in the system and the utilizations of system resources are high, both the compute and the memory at the cloudlets can become scarce. Reduction of the application footprint on cloudlets is needed in order to scale better.

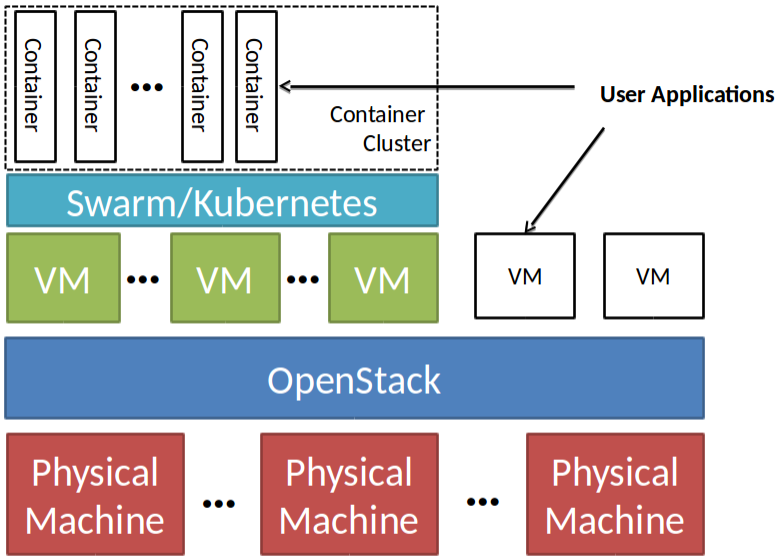


Figure 9: Container and Virtual Machine Virtualization on Cloudlets

Virtual machines (VMs) have been used to provide isolation among edge node tenants [19]. While VMs virtualize at the hardware level and provide strong isolation among tenants, they also have a large footprint since each virtual machine has its own kernel. In the meantime, some applications do not need the strong isolation provided by VMs. For example, applications published by the same developer or company would have stronger trusts about the integrity of the software. Therefore, for these applications, using more lightweight virtualization constructs, for instance, containers, can help reduce the application footprint on cloudlets.

Figure 9 shows an edge node implementation that combines lightweight container virtualization with strongly isolated virtual machines. It adopts a Container-on-top-of-VM approach.

Application providers on the cloudlets can create container clusters for managing multiple applications or instances of an application using the lightweight containers. In the meantime, different providers are isolated by VMs for better control. The edge node deployment system also supports applications to use a combination of virtual machines and containers. For example, an application might have some components in Windows VMs while other components run inside Linux containers. When this combination of virtualization is used, the system provides a DNS service to help resolve hostnames.

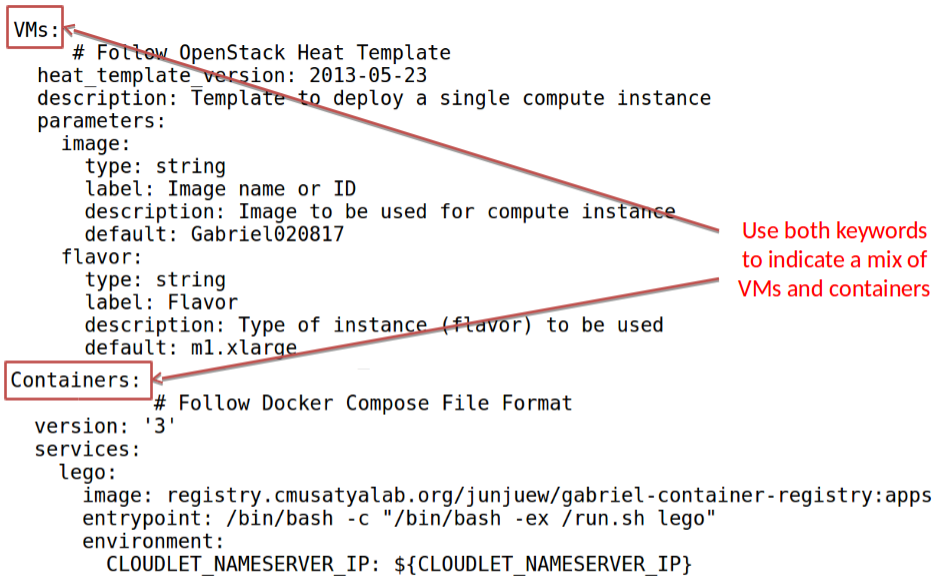


Figure 10: Application Deployment Template with Virtualization Type Specified

In order to specify virtualization techniques to use, developers only need to modify their existing orchestration templates to include their choice of virtualization. The system's annotation uses YAML [4] and follows the convention of orchestration frameworks, including OpenStack Heat and Docker Swarm. Figure 10 shows an application configuration file that adopts a mixed of virtual machine and container virtualization. Components with different virtualization are separated into different sections. Developers mark each section with the virtualization technique they want to use. Specifications inside each section correspond to the template consumed by the underlying orchestration frameworks.

5 Gabriel Acceleration Framework

DNNs have been widely adopted as the de facto algorithms to use for analyzing the image and video data. Therefore, running DNNs to analyze images is a common, if not the most common, workload Gabriel needs to support. Due to the parallel nature of DNN computation, such workload can be accelerated by special hardware, including GPUs, FPGAs, and custom ASICs.

On cloudlets, due to physical constraints and economic decisions, we are likely to see a wide range of heterogeneous accelerators. Some edge deployment may have a cluster of custom ASIC

DNN accelerators with proprietary drivers. Others might use commodity GPUs. Some may not have any accelerators at all. Such heterogeneity of acceleration capabilities has a direct impact on application latencies and the number of users an edge node can serve. While hardware capabilities vary, applications' needs on acceleration resources vary as well. Applications may use DNNs of different sizes based on the complexity of their task. For example, the Face Recognition Assistant developed uses a much smaller network than the Sandwich Assistant in [8]. As a result, the Face Assistant does not need a GPU to meet its latency requirement while the Sandwich Assistant does.

Given the heterogeneity in both hardware capabilities and application demands, I plan to study how to effectively federate cloudlets with different accelerators to better serve applications with different needs. The heterogeneity can be both inter-cloudlet and intra-cloudlet. I plan to address inter-cloudlet heterogeneity from a cloudlet discovery perspective. Specifically, I want to focus on how to maximize the number of users served by carefully discovering and selecting appropriate cloudlets to offload at run-time based on the availability of accelerators, their utilization, and the potentials to meet latency requirements. For instance, different applications from the same mobile client may be served by different cloudlets to accommodate their needs for accelerators and the requirements of latency.

In addition, within a cloudlet, there can be machines with and without accelerators. Virtualization for accelerators is difficult to develop and often cannot be easily adapted to new hardware. Instead, I plan to study how to enable the sharing of accelerators within a cloudlet on the application layer. One approach I have implemented is to expose the accelerator as an HTTP endpoint of fixed functions. This method is similar to model serving systems described in [42] [11]. Such approach faces a trade-off between throughput and latency. Specifically, the more frames a server batches, the higher throughput the server can achieve. I plan to explore such trade-off in wearable cognitive assistance context.

6 Gabriel Development Tool for Object Detection

Existing ad-hoc approach to develop wearable cognitive assistance not only takes a long time, but also requires computer vision expertise. A developer new to wearable cognitive assistance would need to spend months learning computer vision basics and acquire intuitions to determine what is achievable before developing an application. For instance, a researcher mentions the first application developed to help a user assemble LEGO pieces took him more than four months.

Figure 3 shows the ad-hoc development process. The most critical step in building wearable cognitive assistance is to identify task steps and the visual states of task steps. For example, for the Lego wearable cognitive assistance [8], the task steps are the sequence of shapes needed to achieve the final assembled lego shape. The visual states to recognize are the current shapes on the lego board. Identifying visual states and task steps takes significant time and expertise due to several reasons. First, a developer needs to be familiar with the state-of-art of computer vision algorithms to determine what visual states can be recognized reliably. Second, identifying task steps requires domain knowledge. Third, when visual states become too hard for CV, developers need to adjust the task steps to use other methods for confirmation. Often a redesign of task steps is required to compensate computer vision. For instance, when designing the RibLoc application,

a redesign of the task steps involves asking the user to read out a word on the gauge instead of performing optical character recognition on the lightly-colored characters.

I plan to work on building Gabriel development tools in my thesis. I want to focus on providing automation tools for the most time-consuming procedures in the development workflow – building vision checks.

TPOD (Tool for Painless Object Detection) is a web-based tool I developed to help quickly create DNN-based object detectors. It provides a tracking-assisted labeling interface for speedy labeling and transfer learning-based DNN training and evaluation backends that abstract the nuances of DNNs. Using TPOD to create object detectors is straight-forward. A user would first upload short videos of the object collected from varying lighting conditions and perspectives. Then, the user would label these objects using TPOD's labeling interface. TPOD assists labeling by tracking the labeled object across frames. Augmenting training data with synthetically generated data is also supported. A user then can start training from the web interface. TPOD backend uses transfer learning to finetune an object detector DNN from publicly available networks that have been trained with millions of images. When the training is done, a user can download the object detector as a container image to run the trained models for inference. TPOD also provides interfaces for evaluating and testing trained DNNs.

The initial prototype of TPOD has been used by researchers and students to build wearable cognitive assistance. For example, a group of master students in CMU mobile and pervasive computing class successfully used TPOD to build an assistant for using AED machines.

Going forward, I plan to open-source the tool after following optimizations. First, TPOD's backend needs to be easily extensible. With increasingly many DNN-based object detectors developed in the computer vision community, adding a new object detector to TPOD should be made easy. This design goal requires well-modularized DNN backends with clean and easy-to-use interfaces to query labeled datasets for training, standard serialization format to download, and stable APIs to detect objects using the trained models. Second, TPOD's labeling interface should be well isolated from the automated DNN training backend. With such isolation, individuals looking for labeling tools can leverage TPOD's front-end without setting up the training backend. This isolation requires serializing labeled datasets using widely-used formats, for example, Pascal VOC annotation format. Third, TPOD should be packaged well for installation. I plan to containerize TPOD to make it easy for others to install.

7 Timeline

Timeline		Plan
2018 Summer	May - Aug	Apply Bandwidth Reduction to Gabriel Applications
2018 Fall	Sept - Nov Dec	Complete Gabriel Acceleration Framework MobiSys submission
2019 Spring	Jan - April May	Build Gabriel Development Tools SEC submission
2019 Summer	May - Aug	Thesis Writing
2019 Fall	Sept - Nov Dec	Finish thesis dissertation Thesis Defense

8 Related Work

8.1 Edge Computing

Gabriel applications offload computation to a cloudlet, a small data-center at the edge of the internet [47]. The high bandwidth and low latency offered by cloudlets [17] [28] lay the foundation of wearable cognitive assistance [19]. Previous research has worked on VM synthesis [18] to enable rapid provisioning of applications onto a cloudlet. In addition, user mobility is supported through VM handoff [20], which migrates user states from one cloudlet to another. These pioneer work into cloudlet together with decade-long research into computational offload [12] [16] [14] provides the infrastructure support that makes wearable cognitive assistance applications feasible. Previous measurement of edge computing's impact on mobile applications tested compute-intensive and latency-sensitive applications when network utilization is low [8]. This work relaxes such assumption and focuses on supporting more users when the resource utilization is high.

8.2 Cognitive Assistance Applications

This work builds on top of existing efforts into creating wearable cognitive assistance [19] [7] [8]. While previous work has focused on identifying latency requirements and system optimizations to meet latency constraints, this work focuses on making wearable cognitive assistance economically feasible. Other earlier works have also explored providing cognitive assistance to users. For instance, [39] created a navigation system for the blind twenty years ago. Rhema [50] used Google Glass to help people with public speaking. [52] provided people with aphasia conversational cues on a head-worn display.

Most of these applications run solely on mobile devices and are highly constrained by the limited compute budget. Gabriel applications use an offload approach to leverage beefy static computational resources. Gabriel applications can use state-of-art DNN-based computer vision algorithms to perform complex CV tasks.

8.3 Mobile System Support for Computer Vision Workload

With the proliferation of smartphones and cameras, many research works have studied mobile system supports for computer vision workload. [37] designed and built a system to efficiently support many concurrent computer vision applications on a mobile device. [36] built a custom wearable device that supports continuous inference on sensor data using a low power budget. More recently, many more efforts have been focused on DNNs. [55] and [31] accelerated DNN inference on commodity mobile GPUs. [22] looked at where to schedule DNNs of different accuracy and speed tradeoff under resource constraints. [38] executed convolutional layers in analog domain before sensor read-out to reduce power consumption.

Besides, many researchers have worked on algorithmic optimizations to reduce DNN computation. [23] [32] compressed DNN models by pruning insignificant weights. [24] showcased a DNN hardware accelerator exploiting model compression techniques. [54] [15] studied weight quantization for faster inference. [26] [46] designed DNNs with depth-wise separable convolutions to decrease DNN parameters and increase inference speed on mobile devices.

While Gabriel applications can benefit from improvement on mobile system support for DNNs, we recognize the lasting trend in the performance gap between mobile devices and static elements. The offload approach we adopt enables us to leverage powerful computational resources at the edge of the internet. The improvement in mobile system support for DNNs can help us build better early discard filters and perform more dynamic adaptation to user context.

Bibliography

[1] YouTube Collection 1. https://www.dropbox.com/sh/zksp1pzc1ix5hlw/AAB3HEhx-yLAJVR1Q3HnFpsWa?dl=0,.

[2] YouTube Collection 2. https://www.dropbox.com/sh/3uly2qqwbzjasaa/AABiWSzPD-5uzmvCy3meqPKma?dl=0,.

[3] Mohammadamin Barekatain and et al. Okutama-action: An aerial view video dataset for concurrent human action detection. In *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 2153–2160, 2017.

[4] Oren Ben-Kiki, Clark Evans, and Brian Ingerson. Yaml ain't markup language (yaml) version 1.1. *yaml. org, Tech. Rep*, page 23, 2005.

[5] Android Central. The pixel 2 camera's secret weapon: A google-designed soc, the 'pixel visual core', 2017. Accessed April 16, 2018.

[6] Zhuo Chen. *An Application Platform for Wearable Cognitive Assistance*. PhD thesis, Carnegie Mellon University, 2018.

[7] Zhuo Chen, Lu Jiang, Wenlu Hu, Kiryong Ha, Brandon Amos, Padmanabhan Pillai, Alex Hauptmann, and Mahadev Satyanarayanan. Early implementation experience with wearable cognitive assistance applications. In *Proceedings of the 2015 workshop on Wearable Systems and Applications*, pages 33–38. ACM, 2015.

[8] Zhuo Chen, Wenlu Hu, Junjue Wang, Siyan Zhao, Brandon Amos, Guanhang Wu, Kiryong Ha, Khalid Elgazzar, Padmanabhan Pillai, Roberta L. Klatzky, Daniel P. Siewiorek, and Mahadev Satyanarayanan. 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*. ACM, 2017.

[9] Keith Cheverst, Nigel Davies, Keith Mitchell, Adrian Friday, and Christos Efstratiou. Developing a context-aware electronic tourist guide: some issues and experiences. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pages 17–24. ACM, 2000.

[10] Christopher Cox. *An introduction to LTE: LTE, LTE-advanced, SAE and 4G mobile communications*. John Wiley & Sons, 2012.

[11] Daniel Crankshaw, Xin Wang, Guilio Zhou, Michael J Franklin, Joseph E Gonzalez, and Ion Stoica. Clipper: A low-latency online prediction serving system. In *NSDI*, pages 613–627, 2017.

[12] Eduardo Cuervo, Aruna Balasubramanian, Dae-ki Cho, Alec Wolman, Stefan Saroiu, Ranveer Chandra, and Paramvir Bahl. Maui: making smartphones last longer with code offload. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pages 49–62. ACM, 2010.

[13] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

[14] Jason Flinn. Cyber foraging: Bridging mobile and cloud computing. *Synthesis Lectures on*

Mobile and Pervasive Computing, 7(2):1–103, 2012.

[15] Yunchao Gong, Liu Liu, Ming Yang, and Lubomir Bourdev. Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115*, 2014.

[16] Mark S Gordon, Davoud Anoushe Jamshidi, Scott A Mahlke, Zhuoqing Morley Mao, and Xu Chen. Comet: Code offload by migrating execution transparently. In *OSDI*, volume 12, pages 93–106, 2012.

[17] Kiryong Ha, Padmanabhan Pillai, Grace Lewis, Soumya Simanta, Sarah Clinch, Nigel Davies, and Mahadev Satyanarayanan. The impact of mobile multimedia applications on data center consolidation. In *Cloud Engineering (IC2E), 2013 IEEE International Conference on*, pages 166–176. IEEE, 2013.

[18] Kiryong Ha, Padmanabhan Pillai, Wolfgang Richter, Yoshihisa Abe, and Mahadev Satyanarayanan. Just-in-time provisioning for cyber foraging. In *Proceeding of the 11th annual international conference on Mobile systems, applications, and services*, pages 153–166. ACM, 2013.

[19] Kiryong Ha, Zhuo Chen, Wenlu Hu, Wolfgang Richter, Padmanabhan Pillai, and Mahadev Satyanarayanan. Towards wearable cognitive assistance. In *Proceedings of the 12th annual international conference on Mobile systems, applications, and services*, pages 68–81. ACM, 2014.

[20] Kiryong Ha, Yoshihisa Abe, Thomas Eiszler, Zhuo Chen, Wenlu Hu, Brandon Amos, Rohit Upadhyaya, Padmanabhan Pillai, and Mahadev Satyanarayanan. You can teach elephants to dance: agile vm handoff for edge computing. In *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, page 12. ACM, 2017.

[21] Ilija Hadžić, Yoshihisa Abe, and Hans C Woithe. Edge computing in the epc: a reality check. In *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, page 13. ACM, 2017.

[22] Seungyeop Han, Haichen Shen, Matthai Philipose, Sharad Agarwal, Alec Wolman, and Arvind Krishnamurthy. Mcdnn: An approximation-based execution framework for deep stream processing under resource constraints. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, pages 123–136. ACM, 2016.

[23] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[24] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. Eie: efficient inference engine on compressed deep neural network. In *Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual International Symposium on*, pages 243–254. IEEE, 2016.

[25] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[26] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[27] Wenlu Hu, Brandon Amos, Zhuo Chen, Kiryong Ha, Wolfgang Richter, Padmanabhan Pillai, Benjamin Gilbert, Jan Harkes, and Mahadev Satyanarayanan. The case for offload shaping. In *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications*, pages 51–56. ACM, 2015.

[28] Wenlu Hu, Ying Gao, Kiryong Ha, Junjue Wang, Brandon Amos, Zhuo Chen, Padmanabhan Pillai, and Mahadev Satyanarayanan. Quantifying the impact of edge computing on mobile applications. In *Proceedings of the 7th ACM SIGOPS Asia-Pacific Workshop on Systems*, page 5. ACM, 2016.

[29] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, et al. Speed/accuracy trade-offs for modern convolutional object detectors. In *IEEE CVPR*, 2017.

[30] Junxian Huang, Feng Qian, Alexandre Gerber, Z Morley Mao, Subhabrata Sen, and Oliver Spatscheck. A close examination of performance and power characteristics of 4g lte networks. In *Proceedings of the 10th international conference on Mobile systems, applications, and services*, pages 225–238. ACM, 2012.

[31] Loc N Huynh, Youngki Lee, and Rajesh Krishna Balan. Deepmon: Mobile gpu-based deep learning framework for continuous vision applications. In *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*, pages 82–95. ACM, 2017.

[32] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

[33] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, pages 1–12. ACM, 2017.

[34] Michal Kampf, Israel Nachson, and Harvey Babkoff. A serial test of the laterality of familiar face recognition. *Brain and Cognition*, 50(1):35–50, 2002.

[35] Cory D Kidd, Robert Orr, Gregory D Abowd, Christopher G Atkeson, Irfan A Essa, Blair MacIntyre, Elizabeth Mynatt, Thad E Starner, and Wendy Newstetter. The aware home: A living laboratory for ubiquitous computing research. In *International Workshop on Cooperative Buildings*, pages 191–198. Springer, 1999.

[36] Nicholas D Lane, Petko Georgiev, Cecilia Mascolo, and Ying Gao. Zoe: A cloud-less dialog-enabled continuous sensing wearable exploiting heterogeneous computation. In *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*, pages 273–286. ACM, 2015.

[37] Robert LiKamWa and Lin Zhong. Starfish: Efficient concurrency support for computer

vision applications. In *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*, pages 213–226. ACM, 2015.

[38] Robert LiKamWa, Yunhui Hou, Julian Gao, Mia Polansky, and Lin Zhong. Redeye: analog convnet image sensor architecture for continuous mobile vision. In *ACM SIGARCH Computer Architecture News*, volume 44, pages 255–266. IEEE Press, 2016.

[39] Jack M Loomis, Reginald G Golledge, and Roberta L Klatzky. Navigation system for the blind: Auditory display modes and guidance. *Presence*, 7(2):193–203, 1998.

[40] PC Magazine. Fastest mobile networks 2017, 2017. Accessed April 16, 2018.

[41] nvidia.com. Nvidia titan x, 2017. Accessed April , 2018.

[42] Christopher Olston, Noah Fiedel, Kiril Gorovoy, Jeremiah Harmsen, Li Lao, Fangwei Li, Vinu Rajashekhar, Sukriti Ramesh, and Jordan Soyke. Tensorflow-serving: Flexible, high-performance ml serving. *arXiv preprint arXiv:1712.06139*, 2017.

[43] Ozgur Oyman, Jeffrey Foerster, Yong-joo Tcha, and Seong-Choon Lee. Toward enhanced mobile video services over wimax and lte [wimax/lte update]. *IEEE Communications Magazine*, 48(8), 2010.

[44] The Register. Microsoft's hololens secret sauce: A 28nm customized 24-core dsp engine built by tsmc, 2016. Accessed April 16, 2018.

[45] Alexandre Robicquet, Amir Sadeghian, Alexandre Alahi, and Silvio Savarese. Learning social etiquette: Human trajectory understanding in crowded scenes. In *European Conference on Computer Vision*, 2016.

[46] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. *arXiv preprint arXiv:1801.04381*, 2018.

[47] Mahadev Satyanarayanan, Paramvir Bahl, Ramón Caceres, and Nigel Davies. The case for vm-based cloudlets in mobile computing. *IEEE pervasive Computing*, 8(4), 2009.

[48] Mahadev Satyanarayanan, Rahul Sukthankar, Adam Goode, Nilton Bila, Lily Mummert, Jan Harkes, Adam Wolbach, Larry Huston, and Eyal de Lara. Searching complex data without an index. *International Journal of Next Generation Computing*, 1(2), 2010.

[49] Enrico Tanuwidjaja, Derek Huynh, Kirsten Koa, Calvin Nguyen, Churen Shao, Patrick Torbett, Colleen Emmenegger, and Nadir Weibel. Chroma: a wearable augmented-reality solution for color blindness. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pages 799–810. ACM, 2014.

[50] M Iftekhar Tanveer, Emy Lin, and Mohammed Ehsan Hoque. Rhema: A real-time in-situ intelligent interface to help people with public speaking. In *Proceedings of the 20th International Conference on Intelligent User Interfaces*, pages 286–295. ACM, 2015.

[51] The Verge. Google glass becomes more useful with new facebook, twitter, and tumblr apps, 2013. Accessed April 16, 2018.

[52] Kristin Williams, Karyn Moffatt, Denise McCall, and Leah Findlater. Designing conversation cues on a head-worn display to support persons with aphasia. In *Proceedings of the*

33rd Annual ACM Conference on Human Factors in Computing Systems, pages 231–240. ACM, 2015.

[53] Keith Winstein, Anirudh Sivaraman, Hari Balakrishnan, et al. Stochastic forecasts achieve high throughput and low delay over cellular networks. In *NSDI*, volume 1, pages 2–3, 2013.

[54] Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng. Quantized convolutional neural networks for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4820–4828, 2016.

[55] Shuochao Yao, Shaohan Hu, Yiran Zhao, Aston Zhang, and Tarek Abdelzaher. Deepsense: A unified deep learning framework for time-series mobile sensing data processing. In *Proceedings of the 26th International Conference on World Wide Web*, pages 351–360. International World Wide Web Conferences Steering Committee, 2017.